PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
University of Aboubakr Belkaïd - Tlemcen -
Faculty of TECHNOLOGY
Department of Industrial Engineering

MASTER'S THESIS

Submitted in partial fulfillment of the requirements for the Master's degree

In: Industrial Engineering
Specialization: Supply chain

Blockchain Based Solution for the last mile delivery

By: BENYELLES Imene Yasmine

Publicly defended on 06/01/2025 in front of the jury composed of:

Mr Ahmed Hassam	MCB	Univ. Tlemcen	President
Mr Khalid Mekamcha	MCB	Univ. Tlemcen	Examiner
Mr Abdessamad Ait El Cadi	Prof	UPHF Valenciennes	Examiner
Mr Mehdi Souier	Prof	Univ. Tlemcen	Thesis Director
Mr Abdelghani Bekrar	MC.	UPHF Valenciennes	Thesis Co-director

Academic Year: 2024/2025

Gratitude

"...With great power comes great responsibility" -Uncle Ben from Spider-Man 2002

To my supervisors, thank you for teaching me to wield the power of knowledge with purpose, integrity, and responsibility.

To Prof. Mehdi Souier, thank you for believing in me from the very beginning and encouraging me to me to take on this challenging path.

To Dr. Abdelghani Bekrar and Dr. Yassine Idel Mahjoub, thank you for your constant support, patience, and invaluable guidance throughout this journey.

I would like to express my sincere gratitude to Dr. Ahmed Hassam, for the great honor he has bestowed upon me by presiding over the defense of my master's thesis. His presence is deeply appreciated and his role invaluable.

My heartfelt thanks go to Dr. Khalid Mekamcha, for his time, attention, and insightful evaluation of my work. I am truly grateful for his contribution to this important academic milestone.

I am also profoundly thankful to Prof. Abdessamad Ait El Cadi from UPHF, Valenciennes, for accepting to evaluate this modest work. His expertise and constructive remarks are highly appreciated.

My gratitude extends to all my teachers and mentors who have shaped my academic journey and played a key role in the quality of my education.

Finally, I thank all those who contributed, directly or indirectly, to the completion of this thesis. Your support has been essential.

Acknowledgements

All praise is due to Allah for guiding me, strengthening me, and allowing me to complete this journey.

To my dear parents, thank you for your unconditional love, endless sacrifices, and for being my constant source of strength and comfort.

To my brothers Amine and Adil, thank you for the laughter, support, and chaos that made this journey lighter and more memorable.

To my best friends Nesrine Benyelles and Soumia Djafour, thank you for being the sisters I got to choose and for walking this life with me with love and loyalty.

To Anfel Bennekrouf and Meryem Belmahi, thank you for being my unexpected blessing and for bringing joy, weirdness, and warmth into my life.

To my university friends from all over Algeria: Rabab, Fouad, Fethi, Houssem, Aymen, Nesrine, Taha, Mehdi, and my classmates...thank you for the laughter and every shared moment that made this ride unforgettable.

To my beloved grandmother Khadidja, may Allah grant you Jannah; I hope I made you proud.

Ineternational Conference

The 2025 International Conference on the Leadership and Management of Projects in the digital age (ICLAMP2025)

13-14 April 2025, Kingdom of Bahrain (Hybrid)

A Blockchain-Based Hybrid Vehicle Routing Framework with Time Window for Last-Mile Delivery.

Imene Yasmine Benyelles, Mehdi Souier, Abdelghani Bekrar, Yassine Idel Mahjoub.

Abstract

This thesis presents a hybrid framework for optimizing last-mile delivery through the integration of mathematical modeling and blockchain technology. A heterogeneous fleet vehicle routing problem with time windows is first formulated and solved using exact and metaheuristic approaches. Then, a Solana-based smart contract is developed to ensure secure, transparent, and decentralized task management. The solution enhances cost efficiency, traceability, and sustainability across the delivery chain.

Keywords: Last-mile delivery, Vehicle Routing Problem, Simulated Annealing Metaheuristic, Blockchain, Solana, Smart Contracts, Sustainability

Résumé

Ce mémoire propose un cadre hybride pour l'optimisation de la livraison du dernier kilomètre en combinant la modélisation mathématique avec la technologie blockchain. Un problème de routage de véhicules à flotte hétérogène avec fenêtres temporelles est d'abord formulé et résolu à l'aide d'approches exactes et métaheuristiques. Ensuite, un contrat intelligent basé sur Solana est développé pour garantir une gestion sécurisée, transparente et décentralisée des livraisons. La solution améliore l'efficacité, la traçabilité et la durabilité de la chaîne logistique.

Mots-clés : Livraison du dernier kilomètre, Problème de routage de véhicules, Métaheuristique du recuit simulé, Blockchain, Solana, Contrats intelligents, Durabilité

الملخص

يقدم هذا البحث إطاراً هجينيًا لتحسين عملية التوصيل في الميل الأخير من خلال دمج النمذجة الرياضية مع تكنولوجيا البلوكشين. تم نمذجة مشكلة توجيه المركبات لأسطول غير متجانس مع نوافذ زمنية، وحلها باستخدام منهجيات دقيقة وطرق ميتاهيوريستيكية. بعد ذلك، تم تطوير عقد ذكي على شبكة سولانا لضمان إدارة آمنة وشفافة ولا مركزية لعمليات التوصيل. تسهم هذه المنظومة في تعزيز الكفاءة، التتبع، والاستدامة ضمن سلسلة التوريد.

الكلمات المفتاحية: التوصيل في الميل الأخير، مشكلة توجيه المركبات، خوارزمية التلدين المحاكى، البلوكشين، سولانا، العقود الذكية، الاستدامة

Contents

Introduction **1**

1 Supply Chain: Last Mile Delivery **3**
1.1 Definition of logistics . . . 4
1.2 City Logistics . . . 4
1.3 Definition of Supply chain . . . 5
1.3.1 Supply chain components . . . 6
1.3.2 Supply chain stages . . . 8
1.4 Last Mile . . . 8
1.4.1 Last Mile Logistics . . . 9
1.4.2 Last Mile Distribution . . . 9
1.4.3 Last Mile Fulfillment . . . 9
1.4.4 Last Mile Transport . . . 9
1.4.5 Last Mile Delivery . . . 9
1.5 Typology of Last Mile Delivery . . . 10
1.6 Last Mile Delivery Challenges . . . 10
1.7 Vehicle Routing Problem in Last-Mile Delivery . . . 11
1.7.1 Definition of the Vehicle Routing Problem . . . 11
1.7.2 Categories of VRP's in Last Mile Delivery Problem . . . 14
1.7.3 Crowdshipping (Occasional Drivers) . . . 15

2 Blockchain **17**
2.1 The growth of Internet . . . 18
2.1.1 Internet of Connection-Web 1.0 . . . 18
2.1.2 Internet of Information-Web 2.0 . . . 18
2.1.3 Internet of Value-Web 3.0 . . . 19
2.2 Blockchain Technology . . . 19
2.2.1 Definition . . . 19
2.2.2 Blockchain Features . . . 21
2.3 Key Concepts of a Blockchain . . . 22
2.3.1 Blocks . . . 22
2.3.2 Ledger . . . 23
2.3.3 Transactions . . . 23
2.3.4 The Hash . . . 24
2.3.5 Nodes and miners . . . 24
2.3.6 Peer-to-peer (P2P) . . . 24
2.4 Types of Blockchain . . . 24
2.4.1 Public Blockchain . . . 24

2.4.2 Private Blockchain . . . 25
2.4.3 Consortium Blockchain . . . 25
2.4.4 Hybrid Blockchain . . . 25
2.5 Cryptography in Blockchain . . . 25
2.5.1 Symmetric Key Cryptography . . . 26
2.5.2 Asymmetric (Public-Key) Cryptography . . . 26
2.5.3 Zero-Knowledge Proofs . . . 27
2.5.4 Cryptographic Hash Functions . . . 27
2.5.5 Digital Signatures . . . 28
2.6 Consensus algorithms . . . 28
2.6.1 Proof of Work . . . 28
2.6.2 Proof of Stake . . . 28
2.6.3 Proof of History . . . 29
2.7 Smart Contract . . . 29
2.7.1 Structure of a Smart Contract . . . 30
2.7.2 Key benefits of Smart Contracts . . . 30
2.7.3 Programming languages used for Smart contracts . . . 30
2.8 Challenges and Limitations of The Blockchain . . . 31
2.9 Blockchain in Supply chain . . . 32
2.9.1 Smart Supply chain . . . 32
2.9.2 Overcoming Supply Chain Limitations . . . 32
2.9.3 Current Applications . . . 33
2.10 Blockchain in Last Mile Delivery . . . 35
2.10.1 Research Gap and Contribution . . . 39

3 Optimization of the VRP-TW with a Heterogeneous Fleet 41
3.1 Problem description . . . 42
3.2 Hypothesis . . . 42
3.3 Problem formulation . . . 43
3.3.1 Parameters . . . 43
3.3.2 Decision Variables . . . 44
3.3.3 Objective Function . . . 44
3.3.4 Constraints . . . 44
3.3.5 Constraints Explanation . . . 46
3.4 Resolution approach . . . 47
3.4.1 Exact Resolution . . . 47
3.4.2 Metaheuristic . . . 48
3.5 Application . . . 56
3.5.1 Data . . . 56
3.5.2 Scenarios . . . 58
3.5.3 Results and discussion . . . 58

4 Blockchain Framework for the Last Mile delivery 65
4.1 Overview of Solana Blockchain . . . 66
4.2 Solana Programs . . . 66
4.2.1 Key Characteristics . . . 66
4.2.2 Development of Solana Programs . . . 66
4.3 Tokens on Solana . . . 68

4.4 Offchain and Onchain storage 68
4.5 Framework Overview and assumption 68
4.6 Functions and Data Structures 70
4.7 System Interactions 77
4.8 System Configuration and Operating System 81
4.8.1 Solana playground 81
4.8.2 NodeJS and Mocha 81
4.8.3 Phantom wallet 82
4.8.4 Python SDK for Solana 82
4.8.5 Solana Explorer 83
4.9 Implementation 83
4.9.1 Environment configuration 83
4.10 Last-Mile delivery Smart contract 85
4.10.1 Platform Initialization and Registration 85
4.10.2 Order Lifecycle Management 88
4.10.3 Assignment and Pickup 90
4.10.4 Delivery and Payment 92
4.10.5 Sustainability 95
4.10.6 Ratings 98
4.11 Building and deploying the smart contract 103
4.12 Smart contract Testing 104
4.12.1 Scenario 1 104
4.12.2 Scenario 2 105
4.12.3 Scenario 3 106
4.12.4 Client Interaction and Workflow Validation 107
4.13 Result of the Offchain integration 108

List of Figures

1.1 Scopes and perimeters of city logistics [41] 5
1.2 Supply Chain activities [36] 6
1.3 SCOR-DS [9] 7
1.4 Last Mile Framework [107] 8
1.5 Vehicle Routing Problem 12
1.6 Algorithm classification of VRP [151] 13
1.7 Crowdsourcing 15

2.1 Internet of Connection 18
2.2 Internet of Information 19
2.3 Internet of Value 19
2.4 Representation of a chain of Blocks 20
2.5 Blockchain Workflow [2] 20
2.6 Blockchain and block structure [72] 22
2.7 Centralized and Decentralized ledger [27] 23
2.8 Symmetric Cryptography 26
2.9 Asymmetric Cryptography 27
2.10 Structure of a smart contract [58] 30
2.11 Comparison between traditional and Blockchain powered Supply Chain Management (SCM) [147] 33
2.12 Cyber-physical scheduling control framework for smart contract design[42] 35
2.13 Overview of packchain framework and its actors [44] 36
2.14 Blockchain-Based Proof of Delivery of Physical Assets With Single and Multiple Transporters [65] 36
2.15 CrowdBC Framework [88] 37
2.16 Digital Twin and blockchain framework for the Last Mile Delivery [45] . 38
2.17 Workflow and Components of Decentralized Crowdsourced Delivery Model [99] 39

3.1 Resolution approach 48
3.2 Simulated Annealing 49
3.3 Flowchart of the simulated annealing algorithm 50
3.4 Flowchart for SA for VRP-TW-HF 55
3.5 Temperature decrease with $\alpha = 0.995$ starting from $T_0 = 1000$ until reaching $T_{\min} = 10^{-3}$. 55
3.6 Geographic map illustrating client locations (in yellow) and the depot location (in red) 57
3.7 Objective function comparison between Gurobi and SA 60
3.8 30 Nodes with Hard time window- Mixed Fleet 61

3.9 30 Nodes with Light time window- Mixed Fleet 61
3.10 30 Nodes with Hard time window- Company only 62
3.11 30 Nodes with Light time window-Company only 62
3.12 30 Nodes with Hard time window- Crowd Only 63
3.13 30 Nodes with Light time window- Crowd Only 63

4.1 Framework of the Blockchain-based Last Mile Delivery 69
4.2 QR Code - Full System Sequence Diagram (IPFS using Pinata) 78
4.3 System interaction part 1 78
4.4 System interaction part 2 79
4.5 System interaction part 3 80
4.6 Solana Playground 81
4.7 Logo of Mocha and Node.js 81
4.8 Phantom Wallet 82
4.9 Solana.py 83
4.10 Solana workplace 84
4.11 Program id by default 84
4.12 Connecting the wallet 84
4.13 Import in the programs and Program ID 85
4.14 Building and deploying the program 103
4.15 Deployement verification 104
4.16 Successed Transaction 104
4.17 Package Return 105
4.18 Packaging return for circular economy 106
4.19 Delivery by Company Vehicle with Return and Reward 106
4.20 Transaction Overview 107
4.21 Program Instruction Logs 107
4.22 Client interaction using client.ts 108
4.23 Off-chain integration: querying a vehicle account from Solana using AnchorPy 109
4.24 Generation of CID in pinata IPFS of the VRPTW result 109
4.25 Generation of CID in Pinata which contains the result of the VRPTW . . 110
4.26 Result of the routing on IPFS pinata 110

List of Tables

1.1 Notation for VRP Model 14

2.1 Programming Languages Used in Blockchain Development 31

3.1 Summary of Input Data Used for the Vehicle Routing Problem 57
3.2 Scenario configuration based on fleet type, number of nodes, and time window conditions. 58
3.3 Simulated Annealing Algorithm Parameters 58
3.4 Performance Comparison of Gurobi and SA Algorithms 59

4.1 Glossary of Terms Used in Smart Contract Implementation 70
4.2 Main Structures 71
4.3 Enumeration Types 71
4.4 Error Codes 71
4.5 DeliveryPlatform Structure 72
4.6 Depot Structure 72
4.7 Vehicle Structure 73
4.8 Driver Structure 73
4.9 Order Structure 74
4.10 PackagingReturn Structure 75
4.11 Function Descriptions – Initialization and Registration 75
4.12 Function Descriptions – Order Lifecycle Management 76
4.13 Function Descriptions – Rating System 76
4.14 Function Descriptions – Return and Rewards 77
4.15 Function Descriptions – Metadata Assignment 77
4.16 Function Description – Carbon Offset Calculation 77

General introduction

In recent years, the increasing complexity of urban logistics and the rising expectations for fast, transparent, and sustainable delivery services have pushed researchers and practitioners to rethink last-mile delivery systems. As e-commerce continues to grow, the last mile which is known as the last step of the delivery process from a distribution center to the end customer, has become one of the most critical and costly segments of the supply chain. Traditional centralized delivery platforms, though operationally effective, raise growing concerns related to scalability, data privacy, delivery trust, and system vulnerability. These issues are especially pronounced in heterogeneous and large-scale urban environments subjected to dynamic delivery requests, limited resources, demands variations, and high transparency.

This Master's thesis aims to address these challenges by proposing an integrated framework that combines mathematical optimization, metaheuristic adaptation, and blockchain technology to improve the efficiency, sustainability, and trustworthiness of last-mile delivery (LMD) systems. The first part of the present work focuses on modeling the studied LMD problem as a Hybrid Vehicle Routing Problem with Time Windows (HVRPTW), considering a mixed fleet of company-owned vehicles and crowdsourced drivers. Using both exact (Gurobi) and metaheuristic (Simulated Annealing) methods, this research delivers a robust solution approach capable of handling real-world constraints such as limited capacity, time windows, routing costs, and open/closed routes for different vehicle types.

The second part of the thesis builds on this optimization foundation by integrating a decentralized architecture based on the Solana blockchain. A full-featured smart contract is developed using the Anchor framework to manage depot registration, vehicle and driver onboarding, order creation, assignment, pickup, delivery, payments via SPL tokens, sustainability incentives (carbon tracking and packaging return), and performance ratings. Off-chain metadata is stored using IPFS (Pinata) to ensure scalability and traceability while maintaining on-chain integrity. This blockchain-based approach ensures trusted execution, real-time visibility, and decentralization of key decision processes in last-mile logistics.

The key motivations driving this work include: (i) the limitations of centralized systems in ensuring data transparency and fairness, (ii) the opportunity to use blockchain and smart contracts for decentralized coordination, and (iii) the need to develop practical optimization tools to manage complex routing constraints in urban logistics. The proposed system demonstrates the feasibility and benefits of using a hybrid optimization-blockchain solution to address the operational, technological, and environmental challenges of last-mile delivery.

The main contributions of this thesis are as follows:

- Development of a mixed-integer linear programming model (MILP) for the HVRPTW incorporating both company and crowd drivers.

- Implementation of a Simulated Annealing algorithm to provide scalable and near-optimal routing solutions.
- Design and deployment of a Solana-based smart contract for decentralized order management and delivery task allocation.
- Integration of sustainability features including carbon emission tracking, and packaging return reward mechanisms.
- Demonstration of a complete workflow using client scripts and unit testing, validating the interaction between users and the smart contract.

Research Questions

This research is guided by the following key questions and associated hypotheses:

Research Questions

- **RQ1 (Optimization Focus):** How can a hybrid fleet vehicle routing model be optimized based on last-mile delivery costs while satisfying time window and capacity constraints?
- **RQ2 (Blockchain Focus):** How can blockchain-based smart contracts be utilized to ensure transparency, traceability, and trust in last-mile delivery task allocation and execution?
- **RQ3 (Integrated Focus):** How can vehicle routing optimization techniques be effectively combined with blockchain technology to enhance the performance, transparency, and sustainability of last-mile delivery systems?

Structure of the Thesis

The remainder of the thesis is structured as follows:

- **Chapter 1: Supply Chain: Last Mile Delivery**
  Provides an overview of supply chain logistics, with a focus on the definitions, challenges, and typologies of last-mile delivery systems. It also introduces the vehicle routing problem in urban logistics.
- **Chapter 2: Blockchain**
  Introduces the evolution of the internet toward decentralized systems, defines blockchain and its components, and reviews its applications in supply chains, including specific challenges and gaps in last-mile delivery.
- **Chapter 3: Optimization of the Vehicle Routing Problem with Time Windows and a Heterogeneous Fleet**
  Presents the formulation of the HVRPTW, the resolution through exact and meta-heuristic approaches, and includes empirical results comparing performance and solution quality.
- **Chapter 4: Blockchain Framework for the Last-Mile Delivery**
  Details the technical implementation of a decentralized last-mile delivery system using Solana. It describes the smart contract logic, deployment process, testing scenarios, off-chain integration, and discusses future plans for DApp development.

Chapter 1

Supply Chain: Last Mile Delivery

Introduction

In the dynamic, increasingly complex world of global commerce and city logistics, efficiency, transparency, and responsiveness are demanded to a greater extent than ever. As consumers anticipate deliveries to be faster and updates to be in real time, especially if urban areas are densely populated, the last mile of the supply chain, the final step in the delivery process from the distribution center to the end consumer, has materialized as a vital opportunity together with a lasting challenge.

Last-mile delivery is considered as the most resource-intensive part of the logistics process. This part frequently exceeds 50% of total delivery expenses and is costly. Problems such as poor urban infrastructure, traffic congestion, failed delivery attempts, and growing pressure to reduce environmental impact make the last mile even more difficult. In addition, the rapid growth of e-commerce and omnichannel retail has made these challenges worse, turning last-mile delivery into a key area for competition and customer satisfaction.

To deal with these challenges, traceability has become a key strategic factor. The ability to track the movement of goods and confirm their authenticity in real time is now considered essential, not only at the warehouse level but all the way to the customer's doorstep. It supports transparency, accountability, and builds customer trust. Traceability also plays a vital role in ensuring product quality, meeting regulations, and managing risks, while improving overall visibility in the supply chain.

In this chapter, we begin by providing fundamental definitions of logistics and supply chains. We then examine the operational constraints and planning considerations related to deliveries in urban environments through a detailed exploration of urban and city logistics. Following this, we outline the main components and stages of supply chains. The focus then shifts to last-mile delivery, where we analyze various classifications and examine its key functional elements: distribution, fulfillment, transportation, and final delivery that.

Additionally, we address the most pressing challenges associated with last-mile delivery, such as high operational costs, lack of infrastructure, and service fragmentation. We then introduce the Vehicle Routing Problem (VRP) and its variants, which are central to optimizing delivery flows and resource allocation in urban networks. Finally, we explore the emergence of crowdsourcing and participatory logistics models, offering new avenues for decentralized last-mile fulfillment.

1.1 Definition of logistics

The term Logistics was originally used in the military context, it was used as early as *Sun Tzu* in his book *The Art of War* and practiced by leaders such as *Alexander The Great*. Historically, logistics used to refer the planning and movement of troops as well as the transportation of provisions to support the armed forces. Over time, the core principles of military logistics were adapted to enhance the organization and distribution of resources in multiple sectors, eventually evolving into the modern discipline of supply chain management[59].

Logistics is a process of supply chain management that makes sure products move efficiently from the point of origin to the end customer. It involves planning, coordination, and execution of activities such as inventory management, transportation and packaging[143]. Effective logistics not only minimizes operational costs and keeps companies in competitive positions, but also helps to improve customer satisfaction and long-term loyalty, directly influencing supply chain performance.[40]

1.2 City Logistics

The term "logistics" is seen as the discipline that focuses on analyzing, planning, and managing integrated and coordinated physical, informational, and financial flows, a value network that often involves multiple partners.[19].In this viewpoint, City logistics was defined as:

> *'The process of optimizing both logistics and transport activities done by private companies in urban areas while considering the traffic environment, traffic congestion and energy consumption within the framework of a market economy' [130]*

Its basic philosophy is to plan, manage, and control freight movements within a metropolitan area through coordinated actions among all involved stakeholders, with the goal of creating more productive and attractive urban spaces [5, 129].

The figure 1.1 represents the different aspects and components that define the scope and boundaries of city logistic.It highlights that city logistics is often considered the last step in supply chain management, with keywords related to supply chain frequently appearing in the literature. However, it also emphasizes that while freight transport exists in different stages of the supply chain, city logistics specifically focuses on freight transport within urban areas [41].

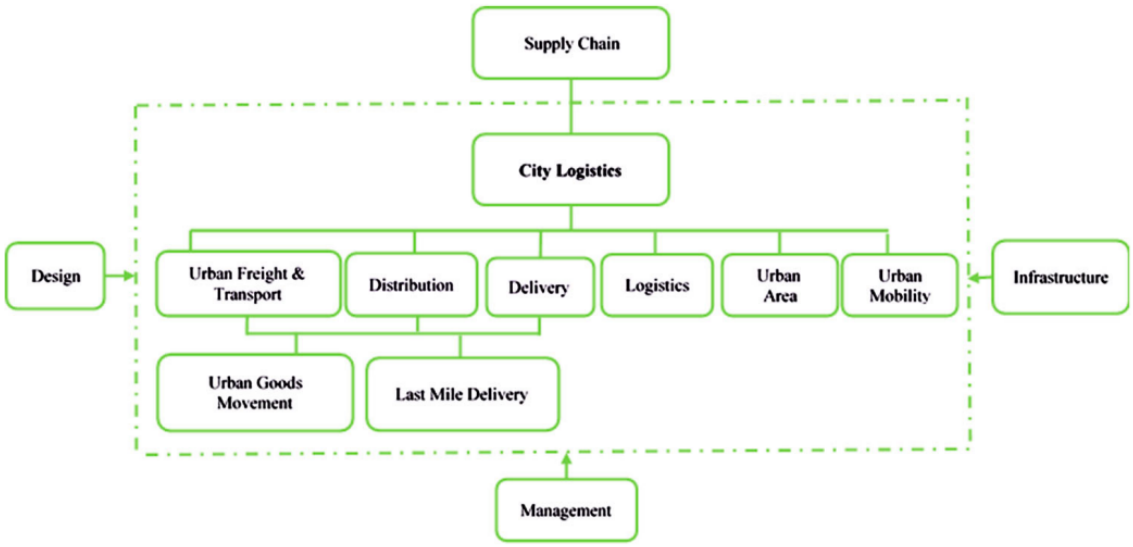


Figure 1.1: Scopes and perimeters of city logistics [41]

City logistics is split into three prespectives: Design (left), Management (bottom). These views guide the planning and operation of logistics systems. Running in parallel to these views is Infrastructure (right), highlighting the importance of physical and organizational structures that support city logistics.

Below City Logistics, specific operational and spatial categories are laid out, each relating to different aspects of urban freight flow [41]. These categrories include:

- **Urban Freight and Transport:** Addresses vehicle routing, congestion, and multimodal transport.
- **Distribution:** Focuses on storage, handling, and moving goods to the next stage.
- **Delivery:** Centers on last-mile logistics, scheduling, and customer service.
- **Logistics:** Covers general supply chain processes, order fulfillment, and reverse logistics.
- **Urban Area:** Relates to the geographical and regulatory elements that influence freight movement within city boundaries.
- **Urban Mobility:** Concerned with how people and goods share transport networks.

Under Urban frieght and transport, distribution and delivery, two elements are highlighted which are: Urban Goods Movement and the Last Mile Delivery. Urban Goods Movement covers the overall process of moving freight around the city. It involves using multimodal coordination, grouping shipments to save trips, and adjusting to city regulatory constraints. Last Mile focuses on meeting high service expectations, the challenge of congested urban areas, and the growing influence of online shopping. Together, these elements demonstrate that effective city logistics need to find a balance between fast, safe transportation and smart planning for goods movement in cities.

1.3 Definition of Supply chain

Supply chain is a wider concept of logistics and building on this framework, it is defined as a network of all parties involved directly or indirectly to meet customer's satisfaction. It

covers not only productions and suppliers but also transport companies, warehouses, retailers, and even the customers themselves. Within any organization, such as a manufacturer, the supply chain includes all activities required to receive and complete a customer order, including product development, marketing, operations, distribution, finance, and customer service.[30].In other words, it is the strategic coordination of all activities involved in the production and delivery of goods and services from sourcing raw materials to manufacturing, logistics, and distribution to the end customer. It involves managing the flow of products, information, and finances among all stakeholders (suppliers, manufacturers, distributors, retailers, and customers) to optimize efficiency, reduce costs, and enhance customer satisfaction[38]. Figure 1.2 illustrate the sequential flow of supply chain activities from raw materials to the end consumer.

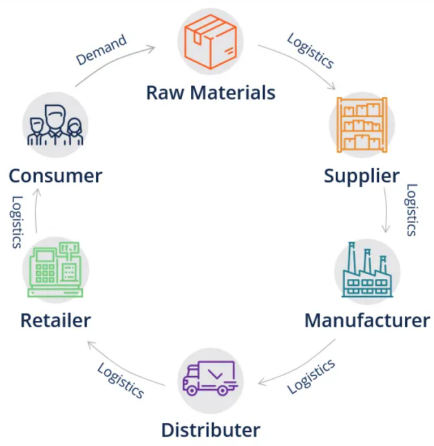


Figure 1.2: Supply Chain activities [36]

1.3.1 Supply chain components

Supply Chain is viewed as a fully integrated network of interdependent activities that work together to create and deliver value to the customer. [34]. According to SCOR-DS (Supply Chain Operation Research-Digital Standard), which is a model that has been developed to describe the business activities associated with all phases of satisfying customer demand. The model itself contains multiple tabbed sections and is organized around the seven primary management processes of Orchestrate, Plan, Order, Source, Transform, Fulfill, and Return [9]. Figure 1.3 represents the SCOR-DS model, that captures the modern supply chain's continuous, interconnected, and cyclical characteristics, highlighting the seven essential processes. The SCOR-DS illustation shows the balance of Supply and Demand in a horizontal infinity loop and Synchronize and Regenerate in a vertical infinity loop.

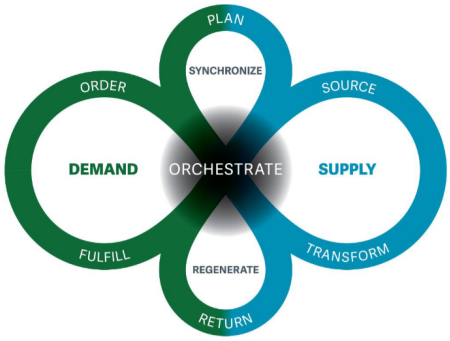


Figure 1.3: SCOR-DS [9]

SCOR-DS is organized around seven major management processes [124] as follows:

- **Orchestrate:** It describes activities that integrate and enable supply chain strategies. This involves establishing business rules and planning, managing human resources, designing networks and deploying technology like the *Blockchain* for traceability, leveraging data analytics, and formalizing contracts and agreements. It also covers regulatory compliance, risk management, ESG (Environment, Social, and Governance) initiatives, circular supply chain practices, and performance management, among other key functions.
- **Plan:** It is the process of developing detailed roadmaps for operating the supply chain. It covers planning for the Order, Source, Transform, Fulfill, and Return functions. This involves identifying requirements; collecting and analyzing data on available resources; balancing them with demand to assess planned capabilities and reveal any gaps while identifying actions how to correct it.
- **Order:** It refers to the processes customers's purchace of a product or a service, capturing details such as location, payment method, pricing, fulfillment status, and all other pertinent order data.
- **Source:** It is defined as the processes involved in obtaining products and services. This includes not only procuring and ordering but also scheduling those orders, managing deliveries, receiving goods, and overseeing the transfer of products and services.
- **Transform:** It represents the activities involved in creating and modifying products. This includes scheduling production activities; carrying out assembly and disassembly; and executing maintenance, repair, and overhaul operations, among other related functions.
- **Fulfill:** It involves all the activities required to execute customer orders or deliver services. This includes planning the delivery schedule, picking and packing items, shipping, as well as installing, commissioning, and invoicing.
- **Return:** It covers the processes related to the reverse flow of goods, services, and associated components from the customer back into the supply chain. It involves assessing the condition of returned items, evaluating their eligibility, and determining whether they should be reintroduced into the production process (Transform) or allocated to other circular activities.

1.3.2 Supply chain stages

It is useful to decompose the supply chain into manageable segments in order to optimize and effectively analyze the challenges at each stage of the overall network. These segments, commonly defined as the first mile, middle mile, and last mile, have specific operational challenges and startegies that contribute to the efficiency of the supply chain.

- **First Mile**: It refers to the initial phase of the distribution network [17] where goods are collected from their point of origin which are the supplier or manufacturer and transported to a warehouse or a distribution center. [126]
- **Middle Mile**:It is the segment in the logistics network that connects regional fulfillment centers or large scale distribution hubs with local distribution centers or sorting facilities. It involves transporting shipments over longer distances within a region, thereby bridging the gap between the initial (first mile) pickup of goods and the final (last mile) delivery to the customer [6].
- **Last Mile**: It is the final part of the supply chain that handles all logistics activities related to getting shipments which are generally small to mid-sized packages ordered online, from a starting point in an urban area (such as a central depot) directly to the final customer's preferred destination. This stage is known for its high costs and complexity, as it involves navigating urban congestion, tight delivery time windows, and often diverse delivery methods. [21]

1.4 Last Mile

Generally, the last mile, alias the movement of goods from a distribution center to the end customer's location, is often seen as the most expensive, inefficient, and polluting part of the supply chain [53]. The framework of Last mile (Figure 1.4) can be examined from two complementary angles: a back-end perspective that addresses the sender's side, and a front-end perspective oriented toward the receiver [107].

Last Mile covers all the activities involved in the final segment of the network, it can be classified into five different typologies: : last mile logistics, last mile distribution, last mile fulfillment, last mile transport, and last mile delivery , as listed in the following sections.

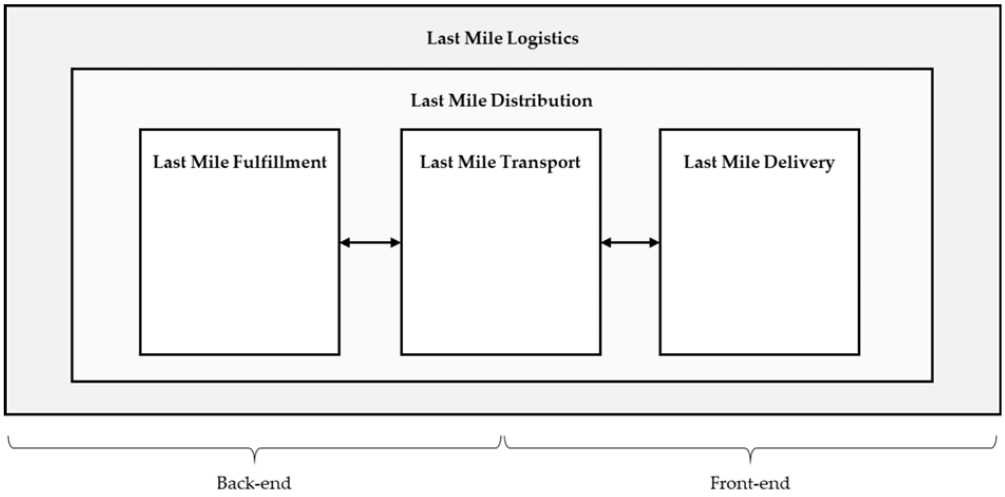


Figure 1.4: Last Mile Framework [107]

1.4.1 Last Mile Logistics

Last mile logistics addresses strategic and long-term planning challenges while providing a comprehensive overview of the system and its surrounding environment. Based on literature, it can be described as the process of planning, implementing, and controlling efficient transportation and storage of goods from the order entry point to the final customer.[107]

1.4.2 Last Mile Distribution

Last Mile Distribution is the process of managing, transporting, and storing products until they reach the end consumer through various distribution channels. It involves tactical planning (mid-term horizon) and operational optimization, often integrating multiple components of the last mile system (fulfillment, transport, and delivery). It covers key operational aspects such as routing [144, 155, 156], transport planning [57], scheduling [16], and facility location[74] like hub-and-spoke systems or urban freight terminals to optimize the movement of goods. Also, it integrates eco-logistics systems[46] and implement collaborative models like drones working alongside trucks to enhance efficiency and sustainability [81].

1.4.3 Last Mile Fulfillment

As it is shown in Figure1.4 Last Mile Logistic consists of 3 differents components and each of them targets differents aspects and highlighting varying themes in the litterature. Last Mile Fulfllment is the process of executing an order by making it ready for delivery [107].In other words, it focuses on transforming orders into shippable packages and involves coordinating activities such as picking, packing, and inventory management. This components ensures accuracy and efficiency in processing customers order through methods like re-engineering fulfillment processes in distribution centers [87] It also involves facility operations, such as managing urban freight terminals [86] or mobile depots [93] to streamline order readiness and reduce bottlenecks.

1.4.4 Last Mile Transport

Last Mile Transport which represents the interface between last mile fulfillment and last mile delivery, refers the movement of goods in the last mile, using different means such as electric vehicles [97],light goods vehicles, bicycles, or drones [49].[107]. Key aspects include operational challenges such as urban congestion, deliveries with tight time windows, and fragmented shipments, as well as innovative solutions such as electric vehicle routing algorithms [138], crowd navigation [47], multi modal deliveries [110].

1.4.5 Last Mile Delivery

Last mile delivery covers the activities required to physically transport goods to the final destination selected by the receiver. It can be viewed as the front-end interface where the supply chain concludes at the consumer. Due to the close connection between last mile delivery and last mile transport, these topics are frequently studied together, with a focus on innovative and emerging technologies especially those related to efficient goods reception solutions [107]. Recent advancements include the use of Blockchain to automate processes like offer management, payment settlement, and proof of delivery. By

integrating this technology, we can ensure transparency and reduce disputes between clients and carriers [44]. Also, recent research includes the application of data-driven optimization and crowdsourcing approaches [25] that integrate smart scheduling algorithms [17], the use of innovative routing models such as the use of occasional drivers for the delivery [8]. In addition, consumer-centric delivery strategies that emphasize attributes such as low delivery fees, speed, and real-time tracking have become essential in enhancing customer satisfaction and loyalty[102].

1.5 Typology of Last Mile Delivery

The typology of last-mile delivery models is becoming increasingly important due to the boom in online retailing and the challenges of e-fulfillment in urban areas. A comprehensive classification framework helps to characterize and classify the various operational setups employed by e-retailers globally. [146]

From case studies, a total of 18 variables can characterize any given last-mile e-commerce delivery model. These variables are organized into five groups [146]:

- **Order placement and payment:** This group deals with the initial interaction of the customer with the e-commerce system. It defines where the orders are placed (online or offline), when the payment is made (at the time of purchase or on delivery), and how the payment is made (credit/debit, online wallets, or cash). These parameters lay the foundation for the fulfillment process and can influence later stages such as product exchange.
- **Warehousing and order preparation:**This section addresses the back-end activities that prepare orders for shipment. It covers the management of the warehousing,either internal or outsourced via third-party logistics,and the degree of automation in order preparation and facility location (ranging from regional hubs to local centers). These choices directly affect efficiency and order lead times.
- **Distribution:** It is the stage at which routing is planned with shipments potentially being combined or delivered on-demand, with or without an intermediate transshipment stop. Distribution management can also include *crowd-sourced* or local courier networks that provide flexible and effective means of last-mile delivery.
- **Customer service performance:** It addresses the order lead time, the specific delivery time-windows offered, and any value-added services.
- **Product exchange:** This group addresses the physical delivery of the product to the customer. It specifies where the product is delivered,be it at home, an automatic locker, a pick-up point, or an urban e-fulfillment center,and whether the delivery is attended or unattended.

1.6 Last Mile Delivery Challenges

The Last Mile delivery is no longer plane transportation for just one product type to one destination; rather, it involves delivery of many different products and services to many different receivers with high expectations on service level and a precise time window. This stage is often represented as the bottleneck of e-commerce [142] or the logistics service nightmare [123]. Below we can find different challenges related to the Last Mile Delivery:

- **Security and Transparency Issues**: One of the key challenges of last-mile delivery is ensuring that delivery data is secured and fully traceable, as the need for customers to protect and reliably track the flow of goods and information throughout the delivery process continues to grow. This challenge is further amplified by the involvement of multiple stakeholders, from suppliers and logistics providers to the final customer, each requiring access to accurate and tamper-proof data. [116][94]
- **High Operational Costs and Resource Allocation**: The last mile leg is often the most expensive part of the supply chain; according to the literature, it represents 13% and 75% of the total supply chain costs [52]. Key challenges include inefficient routing, which increases fuel consumption and vehicle maintenance costs [28], managing multiple delivery options, such as home and locker deliveries, which make route planning and allocation more challenging"[133] and coordinating occasional drivers, which requires compensation schemes and flexibility constraints to balance cost savings and reliability[8].
- **Increasing volume and urbanization**: The rapidly growth of e-commerce, combined with rapid urbanization, has led to a significant rise in parcel volumes delivered to densely populated areas . It creates complex issues such as increased traffic congestion, limited parking spaces, and added pressure on urban infrastructures. As more consumers order online and cities continue to expand, traditional delivery methods often fall short in meeting the intensified demand [21].
- **Sustainability and environmental impact**: The growth of e-commerce combined with urbanization has led to increased vehicle usage and higher carbon emissions, which contribute significantly to air pollution and climate change. Traditional delivery methods often result in inefficient routes and redundant trips that further increase these environmental issues [149, 116]
- **Time pressure and variability**: There are significant challenges in last mile delivery due to the unpredictable nature of urban traffic, fluctuating customer availability, and extremely tight delivery windows. These factors force logistics providers to continuously adjust routing and scheduling plans in real-time, often resulting in increased operational costs and service disruptions [98]. Moreover, even minor deviations or delays can have an impact on the system, reducing overall reliability and customer satisfaction. [117, 35]

1.7 Vehicle Routing Problem in Last-Mile Delivery

The Vehicle Routing Problem (VRP) is one of the most significant problems in logistics[80]. Dantzig and Ramser's [39] introduction of the Truck Dispatching Problem in 1959 introduced research on vehicle routing and delivery scheduling. It is seen as the classic illustration of the vehicle routing problem (VRP) related to the delivery of goods from a central depot to multiple customers.

1.7.1 Definition of the Vehicle Routing Problem

The Vehicle Routing Problem (VRP) is a class of optimization problems in which a fleet of vehicles located at one or more depots must serve a set of geographically dispersed customers as shown in Figure 1.5. The customers have certain demands (e.g., certain

amounts of goods or services required), and the vehicles have certain constraints such as capacity or scheduling constraints. The objective is typically to construct a fleet of vehicle routes, with each route specifying the exact customer visit order for one vehicle in order to minimize one or more cost-related metrics (often distance, travel time, or operating expense), subject to satisfying all operational constraints (e.g., not exceeding vehicle capacity, meeting time-window commitments, etc.).[135]

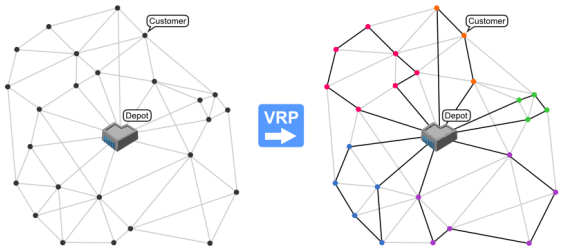


Figure 1.5: Vehicle Routing Problem

The Vehicle Routing Problem (VRP) is NP-hard (non deterministic polynomial-time hard), which means that as the problem instance grows in size whether in terms of the number of customers, vehicles, or constraints, the time and computational resources required to find an optimal solution increase exponentially[101]. As a result, finding exact solutions becomes complex for realistic problem sizes. Consequently, researchers often turn to specialized exact optimization procedures for smaller instances (e.g., branch-and-cut) and various heuristics or metaheuristics (e.g., tabu search, genetic algorithms, or large neighborhood search) for larger or more complex instances[83]. These methods aim to produce high-quality solutions within practical computation times rather than guaranteeing mathematical optimality [135] as shown in Figure 1.6.

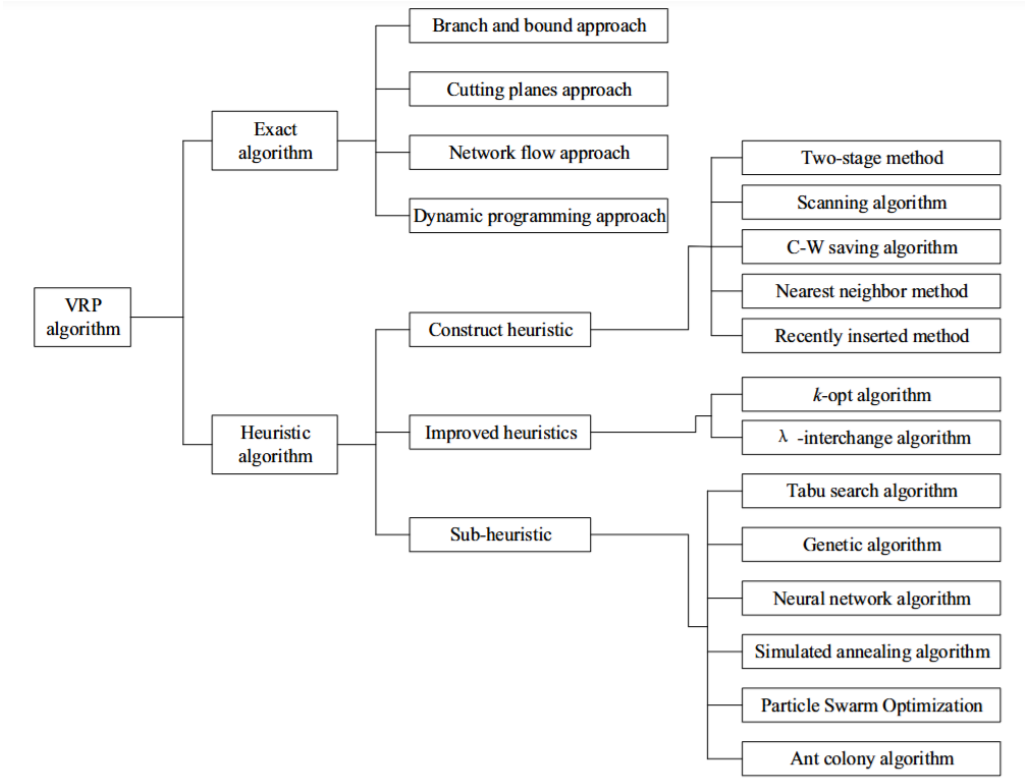


Figure 1.6: Algorithm classification of VRP [151]

According to Haifei Zhang et al [151], the classical VRP can be described as follows:

A depot (node 0) has to supply goods to m customer points (represented by $1,2,\ldots,m$). There is a demand g_i for each customer i, and the depot sends n vehicles with load capacity C_n (where $g_i \leq C_n$).The objective is to construct routes so as to minimize the total cost of transportation, which is directly proportional to the total distance traveled. Key constraints include:

1. Each vehicle's total load must not exceed its capacity.
2. Each customer point is served by only one vehicle.
3. The length of each route must not exceed the vehicle's maximum allowable travel distance.
4. Certain customer points must be visited in a predefined order (e.g., point B before point A).

Let the depot be indexed as 0, customer points as $i = 1,2,\ldots,m$, and vehicles as $s = 1,2,\ldots,n$. The objective is to minimize the overall cost of transportation, which is directly proportional to the distance traveled. A shorter trip conserves fuel, driver time, and resources.

The objective function (1.1) minimizes the total transportation cost, where c_{ij} represents the cost (e.g., distance, fuel consumption) between nodes i and j.

$$\text{Minimize} \quad J = \sum_{i=0}^{m} \sum_{j=0}^{m} \sum_{s=1}^{n} c_{ij} x_{ij}^{s} \tag{1.1}$$

subject to:

$$\sum_{i=0}^{m} x_{ij}^{s} = y_{j0}^{s} \qquad \forall j \in \{1,\ldots,m\}, \forall s \in \{1,\ldots,n\} \tag{1.2}$$

$$\sum_{j=0}^{m} x_{ij}^{s} = y_{0i}^{s} \qquad \forall i \in \{1,\ldots,m\}, \forall s \in \{1,\ldots,n\} \tag{1.3}$$

$$\sum_{i=1}^{m} g_i y_{0i}^{s} \leq q_s \qquad \forall s \in \{1,\ldots,n\} \tag{1.4}$$

$$\sum_{s=1}^{n} y_{0i}^{s} = \begin{cases} 1 & \text{if } i = 1,2,\ldots,m \\ n & \text{if } i = 0 \end{cases} \tag{1.5}$$

$$x_{ij}^{s} \in \{0,1\}, \quad y_{0i}^{s} \in \{0,1\} \qquad \forall i,j,s \tag{1.6}$$

- Constraints (1.2) and (1.3) enforce flow conservation: vehicles begin and terminate at the depot, and routes are linked.
- Constraint (1.4) enforces vehicle capacity restrictions.
- Constraint (1.5) ensures that every customer is visited by precisely one vehicle, and the depot is visited by all n vehicles.

Table 1.1: Notation for VRP Model

Symbol	Description	Type
i, j	Node indices (customers or depot)	Index
s	Vehicle index	Index
m	Number of customers	Parameter
n	Number of vehicles	Parameter
g_i	Demand of customer i	Parameter
q_s	Capacity of vehicle s	Parameter
c_{ij}	Transportation cost from node i to j	Parameter
x_{ij}^{s}	Binary variable: 1 if vehicle s travels from i to j	Decision Variable
y_{0i}^{s}	Binary variable: 1 if vehicle s serves customer i	Decision Variable

1.7.2 Categories of VRP's in Last Mile Delivery Problem

Recent technology advancements and pandemics' influence have prompted fervent research in Last-Mile Delivery (LMD) vehicle routing in the last eight years. The literature has been divided into four general classes according to their problem focus. [75]

- **Crowdshipping (Occasional Drivers)**: Logistic providers complement their professional fleet with non-professional drivers available occasionally to make parcel deliveries.
- **Parcel Locker Delivery**: Packages are delivered to secure, centralized locker locations. Customers retrieve their shipments from assigned lockers using personal access codes or electronic keys.
- **Sidekick-Assisted Delivery**: A main truck delivers parcels and smaller supporting vehicles (sidekicks) to tactical intermediate points. The sidekicks finish the last mile of delivery to customer locations.

- **Flexible Delivery Points**: Customers give a number of possible delivery points (e.g., home, office, or next-door neighbor), and the delivery system chooses the best destination dynamically based on efficiency or as per the customer's preference.

1.7.3 Crowdshipping (Occasional Drivers)

Crowdsourced Logistics (CSL) is an emerging sharing economy business model in which individuals, as independent contractors, make deliveries using their own vehicles. [25] Shippers are connected with drivers through mobile or web apps, which allows companies to save on fixed costs and eliminate empty vehicle trips. [122]. Crowdshipping, a transportation model employing occasional drivers (ODs) to deliver parcels, has become increasingly popular since 2011 and was largely advertised by Amazon in 2015[64]. It is a complement to traditional logistics by leveraging non-professional drivers, which tends to be less expensive than company-owned fleets, particularly in periods of peak demand[118].

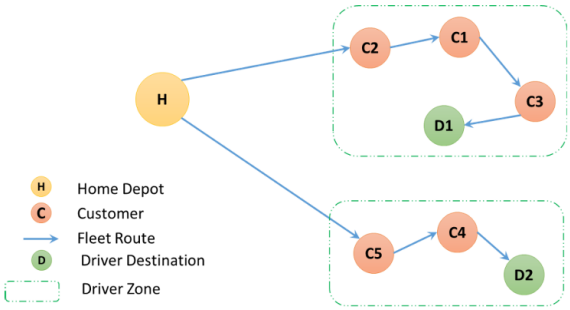


Figure 1.7: Crowdsourcing

Although crowdshipping falls under the open VRP framework,drivers are not required to go back to depots as shown in Figure 1.7, it comes with certain challenges[29, 134]:

- **Time Constraints:** In addition to customer time windows, which are a priority in traditional VRP, crowdshipping has to consider ODs' availability windows.
- **Accessibility Constraints:** Segmentation takes into account constraints like vehicle constraints (e.g., size, vehicle type) or driver preferences that limit access to certain customer locations.

Conclusion

This chapter highlighted the growing challenges in today's supply chains, especially in cities where last-mile delivery has become a major concern. Problems like high delivery costs, traffic congestion, and poor infrastructure make it harder to meet customer expectations. These issues show the need for smarter and more flexible solutions in logistics.

One important solution discussed is traceability, which is the ability to track goods in real time from the warehouse all the way to the customer's door. Traceability helps to improve transparency, build trust, and manage risks. It also plays a key role in improving monitoring and control of supply chains.

We also explored how supply chains work, from their main parts and stages to the specific tasks involved in last-mile delivery, such as distribution, fulfillment, transport, and

final delivery. We looked at different delivery models and introduced the Vehicle Routing Problem (VRP), which is a key issue in planning efficient delivery routes.

We also dive into the use of crowdsourcing in logistics, often called crowdshipping. This approach involves using independent drivers or individuals to deliver packages, often coordinated through digital platforms. Crowdshipping offers new possibilities for flexible last-mile delivery, especially in urban areas. It can help reduce delivery times and costs while providing more scalable and responsive solutions to meet growing consumer expectations.

This chapter builds the foundation for the rest of the thesis, where we will explore how blockchain technology can improve traceability in supply chains. In the next chapters, we will look at how blockchain can make supply chains more secure, transparent, and efficient by allowing better tracking and sharing of information throughout the delivery process.

Chapter 2

Blockchain

Introduction

Just as the internet transformed how we communicate, shop, and do business, a new technology is emerging with the potential to reshape trust, transactions, and transparency in the digital age: Blockchain.

At first, the term "blockchain" may seem distant or technical, much like how the word "Internet" sounded to many in the early 1990s. Today, the internet is everywhere: we use it to send messages, pay bills, check the weather, and receive goods at our doorstep. Blockchain is following a similar trajectory. It is often associated with cryptocurrencies like Bitcoin, but its real value extends far beyond digital money. Blockchain offers a new way to record, share, and verify transactions securely and transparently, without relying on a central authority.

The basic idea behind blockchain is simple: it is a digital ledger, shared across a network, where every participant agrees on a common version of the truth. Instead of sending copies of data (like email attachments), blockchain allows us to send original, tamper-proof records. This eliminates the problem of "double spending" and builds a foundation for trust between parties who may not know or trust each other directly.

Blockchain's power lies in combining cryptography, distributed networks, and consensus mechanisms to ensure that once data is recorded, it cannot be changed without the agreement of the network. This creates a system where transparency and security are built into the technology itself.

As supply chains become more complex and global, the need for trust, traceability, and transparency has grown. Traditional systems often rely on centralized databases and separated information, making it hard to track goods in real time or verify the authenticity of data. Blockchain offers a solution by creating a shared, immutable record of every transaction and movement across the supply chain, from the origin of goods to their delivery to the final customer.

In this chapter, we will explore the evolution of the internet leading to the rise of blockchain (Web 3.0), define what blockchain is and how it works, examine its key features and technical components, and study how it can be applied in supply chain management, particularly in solving traceability challenges in last-mile delivery. We will also discuss smart contracts, consensus algorithms, and the role of cryptography in securing blockchain systems, before addressing current limitations and real-world applications.

Through this chapter, we aim to understand why blockchain is not just a passing trend, but a foundational technology that may transform how organizations collaborate, exchange

value, and build trust, especially in the field of logistics.

2.1 The growth of Internet

The Internet can be divided into three distinct phases, each defined by how users interact with the technology.

2.1.1 Internet of Connection-Web 1.0

Web 1.0 represents the first stage of the Internet, characterized by static, read-only web pages controlled by central authorities, where users could only consume content without interacting or contributing. It is relied on core Internet protocols like TCP/IP and HTTP, with the latter being stateless(unable to retain session information)limiting interactivity. Email emerged as the first widely used application, and early web-based services laid the groundwork for online commerce. Technologies such as cookies and server-side sessions were later introduced to address the limitations of HTTP, enabling basic stateful interactions (Figure 2.1). This era also witnessed rapid advances in computing power and Internet speed, which, alongside the rise of companies like Netscape and the dot-com boom, marked the beginning of the Internet as a commercial and informational platform[136].

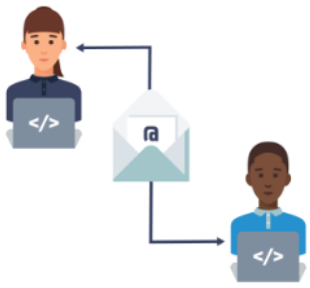


Figure 2.1: Internet of Connection

2.1.2 Internet of Information-Web 2.0

Web 2.0 marks the second generation of the Internet, defined by a shift from static content to dynamic, user-generated content and interactive platforms. Unlike the read-only nature of Web 1.0, Web 2.0 is read-write, allowing users not only to consume information but also to create, share, and engage with it through blogs, social media, and collaborative tools. This era transformed users into both consumers and producers of data, with online behavior generating vast digital footprints (Figure 2.2). Data itself became a valuable commodity, collected, isolated, and monetized by powerful intermediaries. These entities, including traditional financial institutions and new digital platforms, began to dominate online transactions and interactions, often acting as gatekeepers. Web 2.0 also facilitated the rise of digital marketplaces that efficiently connected users across the globe, while the growing centralization of data control raised concerns about privacy, surveillance, and corporate dominance[136].

Figure 2.2: Internet of Information

2.1.3 Internet of Value-Web 3.0

Web 3.0, often referred to as the "Internet of Value," represents the next evolution of the Internet, where value (not just information) is exchanged seamlessly and securely between users without the need for centralized middlemen. This phase is defined by decentralization, community interaction, and increased intelligence through the integration of blockchain technology, smart contracts, machine learning, and artificial intelligence. Unlike Web 2.0, where data is siloed and controlled by large companies, Web 3.0 empowers individuals to own and manage their own identity and data, often through cryptographic tools such as public-private key pairs. Digital assets(both fungible (like cryptocurrencies) and non-fungible (NFTs)) can be transferred directly between people using decentralized networks, removing the need for traditional third parties like banks or sales agents (Figure 2.3). This technological shift enables the creation and exchange of digital versions of real-world and intangible items, making transactions more transparent, secure, and efficient. Web 3.0 thus marks the transition to a more connected, open, and user-empowered Internet [136].

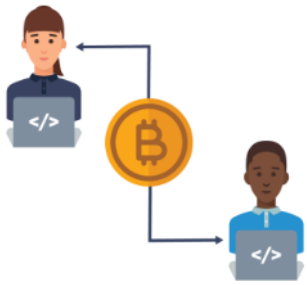


Figure 2.3: Internet of Value

2.2 Blockchain Technology

2.2.1 Definition

Blockchain technology was introduced in 2008, under the pseudonym Satoshi Nakamoto, as the foundation of Bitcoin[100], a decentralized digital currency that was designed to operate independently of government or banking control[139]. While designed for Bitcoin transactions, blockchain's properties for immutability, decentralization, and cryptographic security have enabled its adoption across a wide range of industries, from finance to supply chain management and digital identity authentication[131].As defined by Seebacher and Schüritz [125]:

> *"A blockchain is a distributed database, which is shared among and agreed upon a peer-to-peer network. It consists of a linked sequence of blocks (a storage unit of transaction), holding timestamped transactions that are secured by public-key cryptography (i.e., "hash") and verified by the network community. Once an element is appended to the blockchain, it cannot be altered, turning a blockchain into an immutable record of past activity."*

Blockchain is a decentralized, immutable distributed ledger in which transactions are recorded in a chain of blocks[45]. Each block contains transaction data and is linked to the previous block using cryptographic hashes ensuring data integrity, allowing for fast detection of tampering as shown in Figure 2.4. Since copies of the blockchain are distributed over a peer-to-peer network, transparency and trust are automatically assured[77].

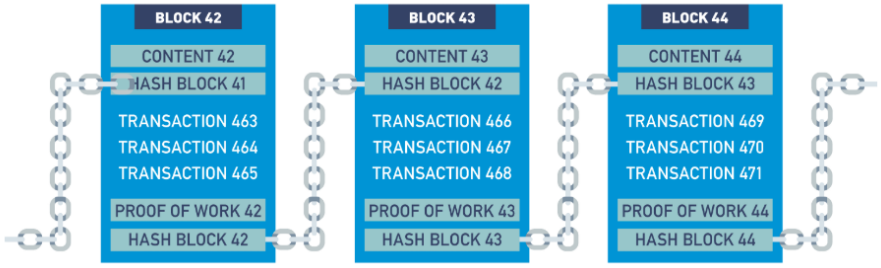


Figure 2.4: Representation of a chain of Blocks

Figure 2.5 represents how do the blockchain work, each part of this workflow will be explained in the next sections.

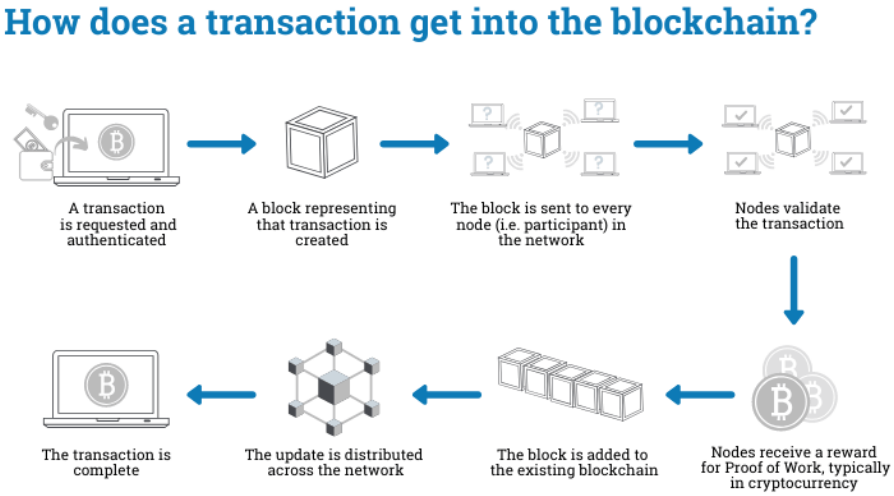


Figure 2.5: Blockchain Workflow [2]

2.2.2 Blockchain Features

As its stated in [100, 43], blockchain technology is characterized by several key features: immutability, transparency, authenticity, decentralization, anonymity, and the absence of intermediaries[92]. These features drive the adoption of this technology in innovative and modern applications.

- **Immutable**: It refers to the inability to delete or modify data after it has been recorded. Blockchain architecture is structured to allow "append-only" access where we can only add data and don't have the ability to change any previous records[24]. It is achieved through the use of cryptography: each transaction is electronically signed, and each block includes a hash of the previous, thus linking the blocks together[67]. For parcel shipment delivery, past transaction data should not be changed to maintain data integrity; if an error is made, it can be corrected through the lense of adding a new transaction that states the error is being corrected, similarly to accounting for booking corrections[68]

- **Transparent**: Awaysheh and Klassen[12] defined transparency as the extent to which information is directly accessible to both parties involved in the exchange, as well as to external stakeholders observing the process. In blockchain systems, transactions occur between cryptographic addresses and can be seen by anyone who has access to the system. When a change is entered into one copy, all other copies are automatically updated. [11].

- **Authentic**: Blockchain ensures that any participant in the network has access to information that has the following attributes : it is *accurate*[51] which means that it is free from falsifications, *timely* as it has been updated in real-time to reflect the current state[15],*consistent* meaning that it remains the same across all nodes in real-time without mismatches [152], and *complete* [48] which means that it includes all information needed to make a proper assessment without omissions.

- **Decentralized**: Centralized systems (e.g., banks) rely on transactions being confirmed by a single central server, which creates performance bottlenecks with high communication overhead and computational costs. Blockchain technology is built on a decentralized peer-to-peer (P2P) network that allows parties to perform transactions directly without a central authority, adding efficiency and minimizing dependence on intermediaries [157].

- **Anonymous**: Each user can interact with the blockchain through a generated address, which does not expose the real identity of the user. [153]. Transactions are recorded using the public address of a person's wallet, which protects their identity[141]. The owner digitally signs each transaction with a private key for source authentication and identification[114].

- **No intermediaries**: Blockchains support peer-to-peer networks by allowing members to transact without relying on trusted intermediaries, which is accomplished through smart contracts, which are self-executing scripts on the blockchain that automate multi-step (interactions) processes. Smart contracts reduce third parties by encoding contractual clauses in code and embedding them in property and self-enforcement[33].

2.3 Key Concepts of a Blockchain

2.3.1 Blocks

A block describes any bounded collection of bits or bytes that forms a distinct unit of data. This concept is applied in a variety of fields, including database management, word processing, and network communication.

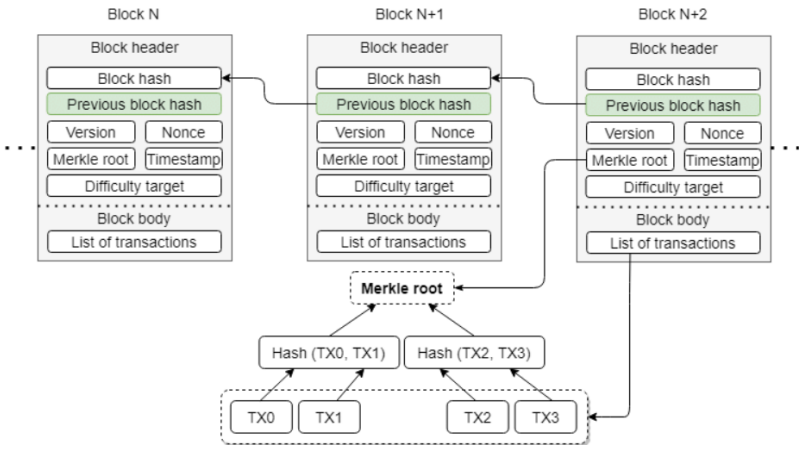


Figure 2.6: Blockchain and block structure [72]

Figure 2.6 represents the block structures linked together to form a blockchain. The first block (not mentioned in the figure), referred to as the *genesis* block, does not have a parent (i.e., no previous block) and has a predetermined hash.

Blocks are the basic units of the blockchain, [115] that holds a collection of transactions, from money exchanges to the execution of arbitrary code through smart contracts[147]. A block consists of two parts:"Block Header" and "Block Body"[157].

The block header holds the metadata necessary for validating, linking, and ensuring the integrity of the block. According to Zheng. et al [154],it includes:

- **Block Version**: A numeric value that specifies the set of consensus rules and validation rules to apply to the block. This allows for both backward and forward compatibility as the blockchain protocol evolves[89, 72].
- **Parent (previous) block hash**: A 256-bit cryptographic hash value that points to the header of the previous block (i.e., Block N-1). Thus, a chain of blocks is created that is immutable and provides tamper detection and integrity for the blockchain. [89, 72]
- **Merkle Root Hash**: A hash of 256 bits derived from the Merkle tree, which is a binary data structure where the leaf nodes contain the values to be stored, and each internal node is the hash of its two children,and it is used for fast and efficient validation of data. Merkle trees summarize the entire set of data in a block by creating a root hash of that data. The Merkle root represents a compact cryptographic signature of all transactions in the block, allowing for efficient verification that a transaction is included in the block without having to inspect the entire block [89, 72].
- **Timestamp**: The Unix epoch time (in seconds since 1970-01-01T00:00 UTC) that represents the date the block is created. This value creates a chronological order of blocks, and prevents a users ability to change the ordering of blocks [154, 89, 72]
- **nBits (Difficulty target)**: a hashing target indicating the difficulty of mining[154]

- **Nonce**: A 4-byte (32-bit) number that starts at zero and increments with each hash iteration while mining. [154, 89, 72]

Block body contains a transaction count and transactions. The transaction counter shows the maximum number of transactions allowed in a block and is based on the size of the block and transaction sizes [157].

2.3.2 Ledger

A *Ledger* is a physical or electronic record-keeping system that records the movement of values and allows the person to always see exactly what value resides where at any moment. Blockchains are digital ledgers that validate and store all data and transactions which occur within a blockchain [85].

Distributed ledger technology (DLT) is a type of database that records the transactions such as assets or data, in a decentralized manner at independent nodes throughout a network. In contrast to a centralized system, DLT does not require a central authority, intermediary, or centralized data store and instead distributes data among many participants in multiple locations, as shown in Figure 2.7. Within this decentralized network, individuals are able to record, share, replicate, and synchronize transactions and data while managing the ledger through *consensus mechanisms* that validate and authorize updates. Each transaction is secured with a unique *cryptographic* signature and *timestamp*, ensuring the records to be auditable and immutable. [27]

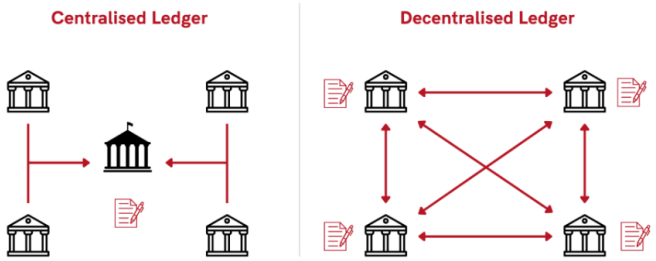


Figure 2.7: Centralized and Decentralized ledger [27]

2.3.3 Transactions

Transactions signify the transfer of value from one node to another made available to the entire network for verification. Once recorded on the blockchain, these transactions are considered confirmed. For a transaction to transfer funds, it must be digitally signed with a private key, allowing the recipient to use the associated public key to verify its authenticity and validate the transfer. Most cryptocurrencies also impose a network fee for processing transactions[84]. A blockchain transaction is recorded as a data block and can include details like: Parties involved identified by cryptographic addresses, the nature of the exchange, the timestamp, the digital addresses or network nodes participating in the transaction, the purpose of the transaction, the quantity of assets exchanged, and the number of pre-conditions satisfied during the transaction [1].

2.3.4 The Hash

Hashing is a process that converts arbitrary input (text, numbers or data) into a fixed-length string of characters. In cryptocurrency systems, like Bitcoin, the transactions are facilitated through a hashing algorithm (Bitcoin uses SHA 256). The hashing algorithm takes the transaction data and produces a large, unique fixed-sized output of 32 bytes. The output then acts as a digital fingerprint for the transaction data, ensuring security and immutability in the blockchain [37].

2.3.5 Nodes and miners

In Bitcoin: A Peer-to-Peer Electronic Cash System, nodes are defined as the entities that are connected to the blockchain network. A node stores a copy of the blockchain ledger, confirms transactions, and helps maintain the integrity of the network. While they can also validate and add blocks, nodes can operate a blockchain network while simply being observers and not directly involved in mining. Miners are considered as special nodes involved in the mining process. Mining is the process of validating and adding blockchain blocks using a process called proof-of-work, which requires solving complex mathematical puzzles. In return for their work and resources expended, miners receive some cryptocurrency that serves as an incentive for them to help to secure the network[100].

2.3.6 Peer-to-peer (P2P)

Peer-to-peer (P2P) technology is a decentralised model featuring a group nodes which collaborate to save and share records. Within a P2P network, each node or device functions as an individual peer, easing communication without the need for central control , thus allowing equal participation among all nodes in task execution[141]. This system can be classified into three quite special models: structured networks, where nodes are organized to enable efficient data searches; unstructured networks, which rely on randomly formed connections; and hybrid systems integrating both client-server and P2P models [137]. Within the blockchain sector, P2P technology supports many cryptocurrencies, as well as enabling such a decentralized ledger to be sustained through independent verification combined with validation of blocks by linked nodes without any dependency upon a central authority [132] .

2.4 Types of Blockchain

Blockchains, much like the other database systems, can be categorized as public, private, consortium, or hybrid variants, depending on how they are being applied [104].

2.4.1 Public Blockchain

We can see the public blockchain like the Internet, where there is no barrier to entry, and anyone with a cell phone and a digital wallet can transact on the blockchain[90].

A public blockchain is open to everyone and operates without any restrictions. Anyone with an internet connection can initiate transactions (including reading, writing, or auditing data) and also has the potential to become a validator. This blockchain variety is totally clear

and open to everyone. Since absolutely anyone can participate, a very key question arises, that is, who actually confirms those transactions? The definitive answer lies well within decentralized consensus mechanisms such as Proof of Work (PoW) and Proof of Stake (PoS), and those enable all participating nodes to collectively validate transactions[115]. Public blockchains represent the general principle of being *"for the people, by the people, and of the people."* Standard examples include Bitcoin, Solana, and Ethereum.

2.4.2 Private Blockchain

Private Blockchain are more synonymous with an Intranet, i.e. we have to control who can access our network, like no company would allow anyone to access their private network[90].

Private blockchains, also known as permissioned blockchains, are designed by or for specific people or organizations and require an invitation from network administrators in order to join. Complete access to participation and complete validation is greatly restricted, and this situation is making this specific type of blockchain ideal for users who specifically need to protect sensitive data from public exposure. Private blockchains are commonly used for accounting as well as record-keeping while maintaining control over data access. As opposed to public blockchains, consensus in a private blockchain is managed by only a central authority, and it determines who, if anyone, is allowed to participate in mining. While this structure introduces a certain level of centralization that contrasts with the customary decentralized nature of blockchain, it remains cryptographically secure [115].

2.4.3 Consortium Blockchain

A consortium blockchain is a type of semi-decentralized , permissioned network in which multiple companies or people collaborate so as to operate nodes. This group, frequently called a consortium or federation, jointly makes decisions that help the whole network [115].

2.4.4 Hybrid Blockchain

A hybrid blockchain integrates a collection of features from both public and private blockchains. It offers the flexibility to designate certain data as public, in addition to keeping other information private [115].

2.5 Cryptography in Blockchain

The term cryptography is derived from two ancient greek terms, *"kryptos"* which means "hidden" and *"graphein"* which means "to write". Cryptography is the science of sending messages back and forth from one party to another in the presence of participants who may wish to corrupt or change the content of a message[90].

Blockchain builds a layer of trust among untrusted parties, allowing safe records along with transactions without any need of a third-party intermediary. Through employment of many cryptographic techniques along with collaborative processes, it eliminates reliance upon centralized institutions. Instead, information is stored on a ledger securely using cryptography [62]. There are several types of cryptography used in the blockchain; among

them, we have symmetric key cryptography, asymmetric (public-key) cryptography, zero-knowledge proofs, hash functions, and digital signatures.

2.5.1 Symmetric Key Cryptography

Symmetric key algorithms are classic encryption methods that use a single private key for encryption and also decryption, as illustrated in Figure 2.8. This requires that both sender as well as the receiver agree on the key beforehand. These methods tend to be faster than asymmetric ones. Their security, however, relies entirely on keeping the key secret. If the key disappears, the encryption algorithm may be cracked with ease. Likewise, by using a complex mathematical formula on an encrypted data (called plaintext), it is possible to find the key used for decryption [121].

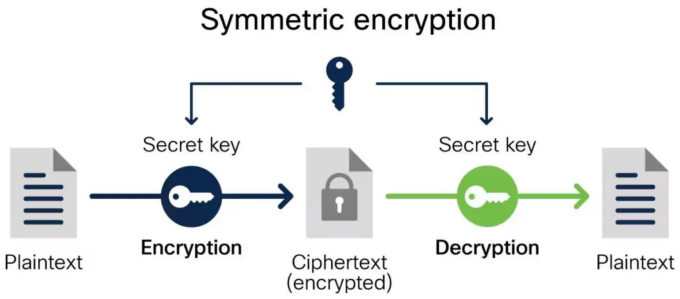


Figure 2.8: Symmetric Cryptography

2.5.2 Asymmetric (Public-Key) Cryptography

A public key is used to verify that a transaction was carried out by the rightful owner. In blockchain systems, the private key is securely stored in a digital wallet. This wallet can be a physical device, known as a hardware wallet, or a software-based wallet, such as a mobile app, desktop application, or web wallet.[62].

Asymmetric encryption (Figure 2.9), also known as public-key cryptography, was developed for addressing limitations with symmetric key algorithms. It employs two mathematically linked but distinct keys, a public key for encryption and a private key for decryption. The separation of these keys improves all security, as a public key can be openly shared while a private key stays confidential. Every transaction gets signed using the sender's private key, and sent to the recipient's public key, so the recipient can verify the transaction by authenticating its digital signature. Only the legitimate owner possessing the private key can authorize cryptocurrency transfers via this mechanism. Therefore, trust and security throughout the blockchain ecosystem is maintained [119].

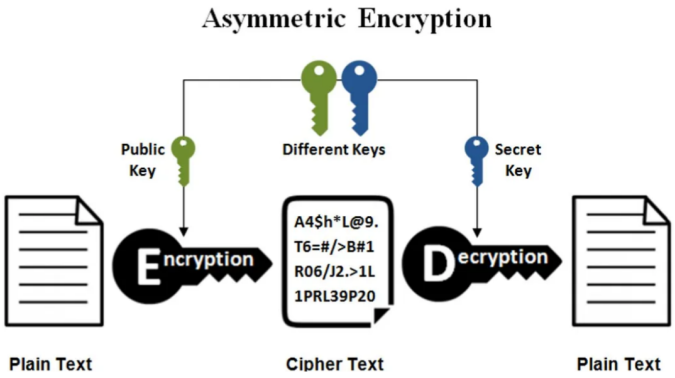


Figure 2.9: Asymmetric Cryptography

2.5.3 Zero-Knowledge Proofs

A zero-knowledge proof (ZKP) is a crypto protocol where one party (prover) can show a statement's truth to another party (verifier) without the prover revealing extra information. In authentication, a user can confirm their possession of the correct password. This happens without ever transmitting or storing any version of the password itself (neither the password nor its hash) thereby preventing exposure to potential security risks[84, 32, 82].

Use Case in Blockchain: A user initiates a request in order to send funds to another user. Prior to committing this transaction, the blockchain must completely verify that the sender has quite a sufficient balance. The blockchain must perform such verification. However, there is no need to know of the sender's identity or of their exact total holdings. Being capable of answering the question "Does the sender have enough funds to complete this transaction?" without revealing who the sender is or how much they own is a primary use case for zero-knowledge proofs in blockchain.

2.5.4 Cryptographic Hash Functions

When using blockchain, the need to trust a central authority to verify the accuracy of data is removed and replaced by trust in a cryptographic hashing function. With data integrity guaranteed by algorithms, trust becomes part of the system [91, 90]. Hash functions are mathematical algorithms that receive inputs of arbitrary lengths of data and return fixed-size numeric outputs, referred to as hash values. The hash value size is independent of the input size, be it a single character or a large document; the result will always be the same length. This algorithm for generating a hash from string or file data is called hashing[62].

The Secure Hash Algorithm (SHA) family such as SHA-256, is the most commonly used in blockchains. These hash functions are primarily employed to ensure data integrity, often alongside digital signatures[92]. For example, if we apply the hash function H=SHA256 to:

The word "Master" the result of the hash will be
"a2bb34780af80cf92faf0872e60567400a910e72543b8de3f47cdb90fc99ffb4"
The sentence "This is Master thesis entitled Blockchain based solution for the last mile delivery." the result of the hash will be
"3894570793a31ae8489574b15ae29a9b2aad94fd06bed4f6188c8cea69f8f110"

2.5.5 Digital Signatures

A digital signature is an asymmetric cryptography method that guarantees the authenticity and integrity of digital messages or transactions. The digital signature consists of two parts: a signature algorithm that generates a signature over the hashed message using the signer's private key, and a verification algorithm to verify the signature created by the signature algorithm using a public key. The signer keeps their private key and signature algorithm secret, and the public key and verification algorithm are made public for others to use to verify that the message has been digitally signed by someone with the private key, and that the message's content has not been changed [121, 150].

2.6 Consensus algorithms

Consensus algorithm is a common collective decision-making process that involves multiple participants reaching and approving allocations that meet the collective goals of the group. It allows distributed nodes each of which share the same values and goals to come to an agreement about the state of the system and is a key element to the success of blockchain networks. Different consensus mechanisms are available, but the most common ones utilized in blockchain are Proof of Work (PoW) and Proof of Stake (PoS) and Proof of History (PoH)[84, 115].

2.6.1 Proof of Work

Proof of Work (PoW) is one of the earliest and most widely adopted consensus mechanisms in blockchain technology, first popularized by Bitcoin [100]. In PoW, network participants known as miners compete to add new blocks by solving a computationally intensive cryptographic puzzle. Each miner performs repeated hash computations until they find a value below a target threshold set by the network; the first miner to find a valid solution shares it with the network. While PoW secures the network by making attacks costly, it can lead to temporary *forks* when multiple miners find valid solutions simultaneously, which the network resolves by accepting the longest chain (i.e., the chain with the most accumulated work) [18, 20].

2.6.2 Proof of Stake

In comparison with Proof of Work, Proof of Stake (PoS) offers an energy-efficient alternative. Rather than expending vast computational resources to solve a cryptographic puzzle, PoS relies on participants holding a sufficient economic stake in the network to become validators [120]. The probability of being selected to validate the next block is directly proportional to the amount of stake (i.e. wealth) a node possesses [78]. By tying block-creation rights to stake rather than raw compute power, PoS discourages malicious behavior (since validators risk losing their stake if they act dishonestly) and eliminates the resource-intensive competition inherent in Proof of Work. Successful validators earn transaction fees, and because no new coins are minted as block rewards, PoS achieves higher throughput and lower latency with far less energy consumption [13].

2.6.3 Proof of History

Accurately tracking time is challenging in distributed systems, especially when high-speed transaction processing requires fine-grained timestamps. Many Blockchain platforms (such as Ethereum) depend on external services to provide a "median" timestamp for ordering transactions. However, relying on a centralized time source undermines decentralization. Solana addresses this by integrating timestamps directly into the ledger via Proof of History, which uses a verifiable delay function (VDF) to generate and embed timeproofs natively. *Anatoly Yakovenko*, co-founder of Solana Labs said:

> "Every block producer has to crank through the VDF, this proof of history, to get to their assigned slot and produce a block"

Solana implements this by inserting each new data element into the PoH sequence appending the hash of the data of the previously generated states. By publishing the state, input data, and count, the system prevents tampering or alternate histories. This setup up provides clear bounds on event ordering, while the VDF doesn't give an actual clock time like it's 10:10:01 AM, it accurately positions every transaction within the blockchain's timeline[50].

Proof-of-History (PoH) is a scalability solution that allows compact blockchains with a verifiable history to be created and secured[113]. Proof of History (PoH) is a cryptographic technique that embeds a verifiable sequence of timestamps directly into a blockchain, creating a tamper-resistant record of *when* events occurred. Introduced by *Anatoly Yakovenko* for *Solana*, PoH uses a cryptographically secure, sequential hash function (typically SHA-256) run in a single, uninterruptible chain. Each new hash incorporates the previous output and an event's data or its hash, producing a verifiable "fingerprint" that proves the passage of *time* between events, as the co-founder of Solana said *"gives the ledger this interesting property where you can infer when events occurred when you examine it"*. External nodes can independently verify segments of this chain in parallel by re-computing hashes on separate cores. By building these timestamps into the ledger itself via a verifiable delay function, PoH eliminates the need for inter-node clock synchronization or external time authorities, reducing communication overhead and enabling high throughput and fast finality.[148]

2.7 Smart Contract

First conceived by Szabo in the 90's [128], smart contracts combine computer protocols with user interfaces to automatically execute predefined contracts, potentially replacing intermediaries like lawyers or banks. Smart contract is deployed at a unique blockchain address, it is triggered by a transaction sent to it and then self-executes on every node based on the embedded code and data. A smart contract is an executable script that can be stored on a blockchain, such that the script can be examined by all participants. Smart contracts are similar to stored procedures in regular databases, but once deployed, they cannot be bypassed [105, 147].

Smart contracts are programs that use "if/when…then…" logic. Once the specified conditions are met and verified by the network, the contract automatically carries out actions, such as releasing funds, registering assets, sending alerts, or issuing tickets, and records the outcome immutably on the blockchain. Only authorized parties can view the results [70].

Participants first agree on how transactions and their data will be represented, define the "if/when... then..." rules, consider potential exceptions, and establish dispute-resolution procedures. Developers then encode these terms into the smart contract. To streamline this process, many blockchain platforms now offer templates, user-friendly interfaces, and online tools for creating and deploying contracts without extensive coding[70].

2.7.1 Structure of a Smart Contract

A smart contract consists of two main components: a set of executable functions and a collection of state variables. When a user sends a transaction to one of these functions (including any required input parameters) the function's logic runs and may update the contract's state variables accordingly. Developers write smart contracts in high-level languages such as Solidity or Python , then use a language-specific compiler (e.g., the Solidity or Serpent compiler) to translate the code into bytecode. Once compiled, the bytecode is deployed to the blockchain network, which assigns it its unique address. From that point, any network participant can trigger the contract's functions by sending transactions to its address. Each time a function executes, every node in the network runs the same code as part of block verification, ensuring consensus on the contract's behavior [14]. Figure 2.10 illustrates this structure.

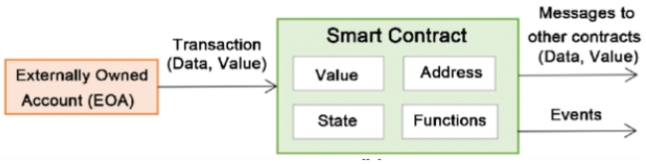


Figure 2.10: Structure of a smart contract [58]

2.7.2 Key benefits of Smart Contracts

Smart contracts offer several advantages that contribute to their growing use across industries. Accuracy is achieved by replacing human intermediaries with executable code, ensuring that processes are consistently performed in the same way. Cost savings result from eliminating intermediaries, which often leads to significant reductions in operational expenses. Efficiency improves as the removal of process intermediaries streamlines operations and accelerates execution. Backup is inherently provided by the blockchain, which maintains a permanent, tamper-proof record that supports auditing, traceability, and insight, even if the original creator is no longer active. Finally, autonomy is enabled as smart contracts can be developed and deployed by anyone, removing the need for third-party agents such as lawyers, or auditors [90].

2.7.3 Programming languages used for Smart contracts

Blockchain demand has surged recently, driven in part by the global pandemic's push toward digital transformation. Organizations are seeking secure, efficient ways to migrate traditional processes online, and blockchain's built-in security and transparency make it an ideal solution.[73]

Among its applications, smart contracts stand out for their diverse functionalities . Hense, smart contract programming languages has been increasing recently. The following table 2.1 provides an overview of some languages that has been used for developing smart contracts[90].

Table 2.1: Programming Languages Used in Blockchain Development

Language	Description
Solidity	An object-oriented, Turing-complete, high-level language for Ethereum smart contracts. Runs on the Ethereum Virtual Machine (EVM) and leverages libraries of reusable code.
Viper	A Python-influenced language built around auditability, security, and simplicity to provide a solid foundation for smart contract development.
Rust	A low-level, statically typed language that is fast and memory-efficient, making it highly scalable. For example, Solana,the high-throughput blockchain, is built in Rust.
JavaScript	A core Web technology enabling dynamic, user-friendly sites. Its vast ecosystem and frameworks simplify integration of front-end and back-end resources with blockchain APIs.
Golang	An open-source language from Google. Much of Hyperledger Fabric's chaincode is written in Go, benefiting from its simplicity and concurrency support.

2.8 Challenges and Limitations of The Blockchain

Blockchain is famous across multiple industries as a transformative technology capable of operating without central authorities or intermediaries.It securely stores tamper-proof data and keeps long transaction histories efficiently. Yet, like other emerging technologies, blockchain has inherent limitations and may not suit every business model [96]. The following sections outline its key challenges and trade-offs:

- **Performance and Scalability:** As the number of nodes grows, blockchains suffer from low throughput (Bitcoin's 6–10 TPS) and high latency (10 min per block) due to resource-intensive consensus (PoW) and limited block sizes (1 MB) . Even energy-efficient protocols like PBFT (Practical Byzantine Fault Tolerance) face quadratic messaging overhead, making them impractical for hundreds or thousands of transactions per second. [140]
- **Privacy:** Although users transact via pseudonymous addresses, all balances and transaction histories are publicly visible. Analyses can link addresses to real identities or IPs, undermining transactional privacy unless privacy requirements built in from the start[66].
- **Energy Consumption:** Proof-of-Work mining consumes huge amounts of electricity (Bitcoin alone uses over 50 TWh annually, comparable to the consumption of some countries)resulting in a substantial carbon footprint and raising sustainability concerns[106].

- **Fairness and Security:** Vulnerabilities such as 51% attacks and selfish-mining allow attackers controlling a majority (or even a minority of hashing power to double) spend or censor transactions, challenging the network's integrity and fairness[55].

2.9 Blockchain in Supply chain

2.9.1 Smart Supply chain

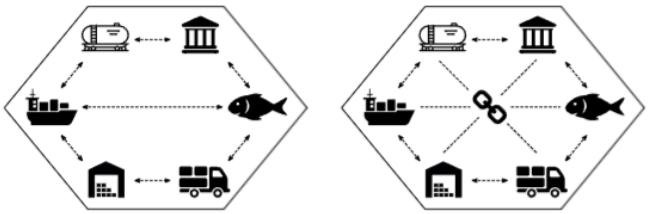
The definition of a smart environment is *"A physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in everyday objects of our lives, and connected through a continuous network"*[145]. In a smart supply chain, each participant (whether retailer, manufacturer, or distribution center) can interact with and communicate about every product, both in storage and in transit, creating a fully integrated, self-regulating system. The supply chain becomes "smart" when its components continuously share data and use it to make autonomous decisions. In this context, blockchain technology enhances smart contracts by having its distributed nodes verify and validate contract conditions in a secure, transparent environment [10].

2.9.2 Overcoming Supply Chain Limitations

Blockchain technology presents transformative potential for supply chain management by standardizing immutable, transparent data exchanges and automating processes within a secure framework. This capability could redefine traditional trust-based models in the field[23]. Despite advances in IT-driven integration of supply chain information flows, significant challenges persist, and grow more pronounced as the number of global participants increases [79, 63].Key issues include bureaucratic inefficiencies in international logistics and each participant's unique way of organizing data. These issues are prevents by a lack of trust between organizations, which prevents them from adopting unified data-sharing interfaces. Additionally, effective solutions must enable secure, decentralized data sharing that automatically follows regulations and contractual agreements[108]. Furthermore,

> "...the data sharing must be secured, distributed (e.g., for optimizing the subsystems locally) and with some automated actions related to the different regulations and negotiations. Thus, Blockchain appears as a natural technology for implementing these common issues"[110]

Blockchain helps overcome the supply chain challenges in four main areas: reducing bureaucracy through automation, preventing counterfeit products via transparency, improving traceability with immutable records, and supporting sustainability by encouraging ethical and environmentally responsible practices[23, 63].



(a) Traditional SCM. (b) Blockchain powered SCM.

Figure 2.11: Comparison between traditional and Blockchain powered Supply Chain Management (SCM) [147]

Figure 2.11 illustrates the different interaction frameworks between entities in traditional and blockchain-enabled supply chain management (SCM) that represent the transformation from the traditional supply chain to the blockchain-based supply chain. In blockchain-based SCM, all transactional interactions among stakeholders are consolidated into a unified blockchain ledger, ensuring full visibility of product-related records across all participants[31].

Several companies advertise to provide blockchain based solutions to improve the efficiency of supply chain management solutions. Walmart uses IBM's Hyperledger blockchain to achieve end-to-end traceability of food products, enabling precise tracking from the origin to the retailers. Similarly, Everledger uses a hybrid blockchain model (combining public and private chains) in the diamond industry, offering permission to stakeholders while maintaining an immutable, auditable transaction history [31].

2.9.3 Current Applications

Blockchain technology is being applied to various use cases within the supply chain to improve efficiency, transparency, and security.

- **Agriculture and Fresh Food**: Guido Perboli et al.[112] outlines a blockchain implementation in a European e-commerce fresh food retailer's supply chain to address challenges like inaccurate data, delivery delays, contamination risks, and transparency gaps. The implementation focused on creating a network of producers, distributors, certifiers, and retailers to track batches of products and certify regulatory compliance. Using Hyperledger Fabric [69] hosted on AWS (Amazon Web Service)[7], the retailer created a permissioned blockchain to tokenize product batches, track transformations (e.g., certifications, cold-chain compliance), and automate legal documentation via smart contracts. The solution is integrated with existing ERPs and IoT sensors to monitor critical parameters such as temperature without replacing legacy systems. Guided by the GUEST methodology, a five-step framework (GO, UNIFORM, EVALUATE, SOLVE, and TEST)[111] aligning digital strategy with business goals, key stakeholders (producers, warehouses, distributors) gained visibility into demand forecasts, reducing the bullwhip effect and optimizing inventory. The results included 850 saved working hours per year in inbound operations, a 10% increase in revenue from reduced counterfeits, and significant cost savings from faster contamination recalls and waste reduction. Challenges included initial resistance from distributors and partial data migration due to scalability constraints, though continuous monitoring revealed unexpected efficiencies, such as distributors adopting blockchain for internal optimizations.

- **Carrefour:**As part of its "Carrefour 2022" transformation plan, the retail giant adopted blockchain technology to boost supply chain transparency and digitalization. Starting with its organic product line [60], Carrefour became the first European retailer to use blockchain for full product traceability. For example, customers buying Spanish organic oranges could scan a QR code on the packaging to access detailed information [26], including:

  - **Origin and Journey**: Producer's name, geographic location, transport routes, and packing details.
  - **Quality Data:** Harvest date, variety, seasonality, and lab analysis results.
  - **Certifications:** Organic certification details, conversion dates, and sustainability initiatives.

  This blockchain system allowed consumers to verify the authenticity and ethical standards of products, ensuring trust in Carrefour's organic claims. The technology also streamlined supply chain operations, cutting costs and aligning with the company's broader goal of digital innovation.

- **TradeLens by Maersk and IBM:** Blockchain is implemented here to digitize global trade processes by creating a shared, decentralized ledger accessible to stakeholders (shippers, carriers, ports, customs), as reported by [71], the partnership between supply chain and blockchain sought to *"vastly reduce the cost and complexity of trading"*. The platform replaces paper-based documentation with automated smart contracts, enabling real-time tracking of shipments and reducing delays. Transactions (e.g., approvals, cargo status updates) are recorded immutably on the blockchain, enhancing transparency and trust. The system streamlines workflows by eliminating redundant data entry and enabling instant verification, cutting costs and inefficiencies linked to manual processes [76].

- **Scheduling:** Alexander Dolgui et al. [42] propose a dynamic scheduling approach for supply chain operations that embeds smart contracts on a blockchain to both design and execute supply chain processes 2.12 . Specifically, they model the supply chain as a flexible flow shop (with multiple logistics/service providers and alternative production routes) and treat each physical operation as a "virtual operation" inside the start and finish of a blockchain-logged information service. By linking the physical process with on-chain events, the blockchain records when each operation starts, waits, and completes, and updates state variables accordingly. This allows fully automated confirmation of operation completions (or disruptions) and triggers re-scheduling if needed. Their mathematical framework merges optimal control (continuous optimization for time-based operations) and discrete assignment (choosing which provider performs which step), so that smart contract transactions effectively monitor and validate each operation in real-time, ensuring transparency, traceability, and the ability to adapt to changing conditions.

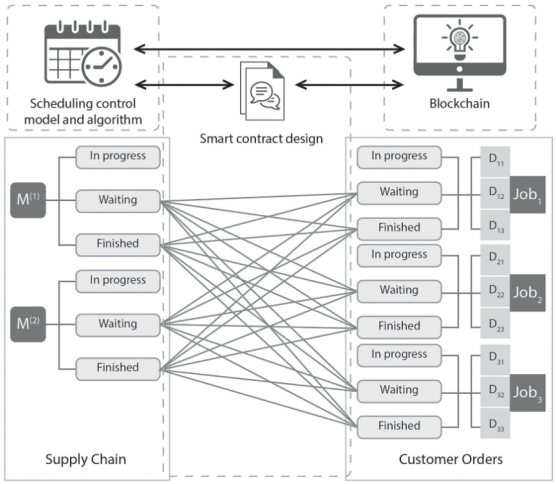


Figure 2.12: Cyber-physical scheduling control framework for smart contract design[42]

2.10 Blockchain in Last Mile Delivery

The literature indicates that numerous studies suggest blockchain technology can enhance the process of delivering physical assets, such as parcels and packages[3].

Riham AlTawy et al. [4] proposed Lelantos, a blockchain-based platform designed to enable anonymous delivery services of physical goods by protecting customer identities and private information from all contractual parties, including shippers and merchants. The system leverages an on-chain smart contract, known as the Lelantos smart contract (Lsc), which handles core functions such as message relays, anonymous order placements, and fair distribution of payments, while off-chain components (like user applications and a web server) support its operations. Inspired by onion routing (a technique designed to ensure anonymous communication within a network. It works by encapsulating messages in layers of encryption, similar to the layers of an onion) [56] and Crowds techniques, Lelantos ensures that each delivery company only knows the adjacent locations in the delivery chain, and it even allows customers to dynamically choose to pick up packages along the route. Customers encrypt and upload their real addresses to the blockchain and must select at least two couriers to engage in an untrusted, yet secure, delivery agreement without relying on a centralized third party. Once a delivery is underway, the system uses secure hashing to prove the delivery, although this design increases computational costs and complexity due to the requirement of multiple delivery companies and the enforcement of a cancellation-free, Point-of-No-Return mechanism.

Soufiane El Moudaa et al. [44] proposed PackChain, a blockchain-based platform designed to manage last-mile delivery through a decentralized, crowdsourced framework. The system uses the Ethereum blockchain to execute core functions via smart contracts written in Solidity, with Web3.js facilitating interactions between clients and carriers, as shown in Figure 2.13. In PackChain, users register using their unique Ethereum addresses, after which clients can place orders containing package details stored off-chain on SWARM which is a decentralized storage solution that only retains a corresponding hash on the blockchain to reduce costs. Carriers then post offers for these orders by specifying required prepayments and deposits. Upon order acceptance, the smart contract processes payments, and a proof-of-delivery mechanism are employed to verify successful delivery through the client's signature, which subsequently triggers the release of the carrier's deposit. This

design ensures transparency, traceability, and tamper-resistance in the delivery process while addressing issues like high costs and centralization present in traditional systems.

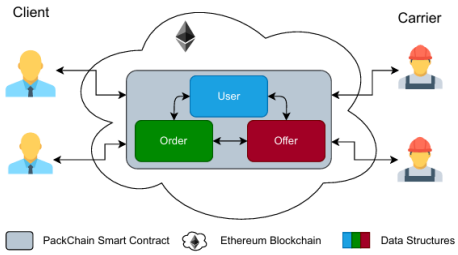


Figure 2.13: Overview of packchain framework and its actors [44]

Hasan and Salah [65] implemented blockchain in their proof-of-delivery solution by developing an Ethereum-based system where all participants (seller, buyer, transporter, and arbitrator 2.14) are assigned unique Ethereum addresses and required to deposit collateral to promote honest behavior. Their solution leverages smart contracts, written in Solidity and attested by a Smart Contract Attestation Authority, to enforce the signed terms and conditions (stored off-chain via an IPFS (InterPlanetary File System) hash) and manage the entire delivery lifecycle. The seller issues two keys, one for the transporter and one for the buyer, which are later exchanged during the delivery process; the smart contract then computes and verifies the key hashes using a secure hashing algorithm (Keccak256) to confirm successful delivery. Upon a match, payments and collateral are settled automatically, while any discrepancies or delays trigger an off-chain dispute resolution through the arbitrator.

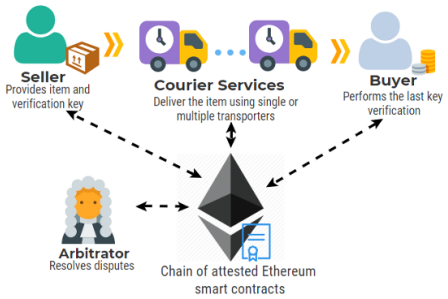


Figure 2.14: Blockchain-Based Proof of Delivery of Physical Assets With Single and Multiple Transporters [65]

In [109], the researchers Patil et al. integrated blockchain technology with the vehicle routing problem (VRP) to optimize short-distance donation logistics. They implemented a permissioned blockchain environment where donation service requests are initiated through an application, after which a VRP module calculates the most efficient routes based on factors such as distance, vehicle capacity, and time constraints. The system records route allocations and transactional details on a decentralized ledger, ensuring transparency, immutability, and trust among stakeholders. Smart contracts are used to automatically manage vehicle assignments and transaction processes, while a consensus mechanism validates each transaction and updates the ledger uniformly across the network. Simulation results demonstrated that this blockchain-optimized approach reduced average journey times by approximately 33 percent compared to traditional routing methods, thus significantly enhancing logistics efficiency for philanthropic activities.

The authors proposed CrowdBC[88], a fully decentralized crowdsourcing framework that leverages blockchain technology to eliminate the need for any central trusted authority. They implemented blockchain by designing a three-layer architecture 2.15:

- Application Layer: User interface (CrowdBC Client) for task management and interaction.
- Blockchain Layer: Handles consensus, state transitions, and smart contract execution.
- Storage Layer: Stores large task data (e.g., encrypted solutions) in distributed systems IPFS .Only hashes and metadata are stored on-chain to ensure data integrity and reduce blockchain storage costs.

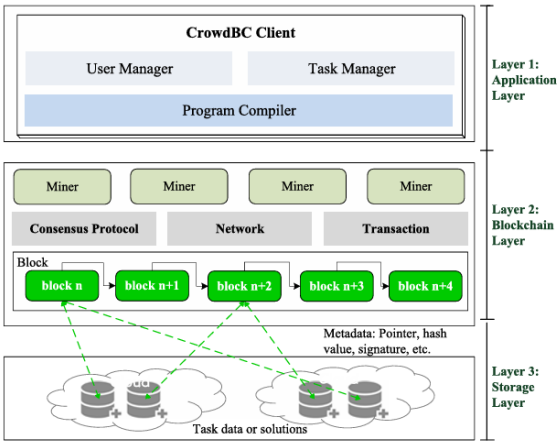


Figure 2.15: CrowdBC Framework [88]

To operationalize the framework, they developed and deployed several smart contracts on a blockchain platform. Specifically, the User Register Contract, User Summary Contract, and Requester-Worker Relationship Contract, each of which governs crucial aspects such as user registration, task posting and receiving, reward assignment, and reputation management. Additionally, the system incorporates a time-locked deposit protocol to enforce fairness and deter malicious behavior, ensuring that both requesters and workers commit collateral that is only released upon task completion verified through automated smart contract evaluations. The framework also uses a state machine to monitor task lifecycle stages, providing transparency and allowing users to query the current state of any task [88].

The authors Feruz Elmayet al . [45] developed a decentralized crowdsourced last-mile delivery framework that leverages Digital Twins (DTs) and dynamic Non-Fungible Tokens (NFTs) on the Ethereum blockchain to enhance real-time package monitoring, secure task allocation, and reputation management 2.16. In their solution, smart contracts are used to manage the entire crowdsourcing process, from task creation to worker allocation and feedback, with two specific contracts: the Tasks Manager Contract (TMC) for handling package-related tasks and NFT minting and the Last-Mile Delivery Contract (LMDC) for registering and managing worker information. Digital Twins are created for both delivery packages and workers; these DTs continuously monitor critical package attributes (such as temperature, humidity, and other sensor readings) and simulate the future state of the package using physics-based models. The DT data, which is dynamic and constantly updated, is secured through the minting of dynamic NFTs whose metadata, stored off-chain via IPFS

and synchronized via IPNS (Inter-Planetary Naming System), provides a tamper-proof, real-time record of each package's history and current status. This holistic approach not only improves data security and integrity but also enables a precise evaluation of delivery quality (Quality of Delivery or QoD), which feeds back into updating workers' reputations. The implementation was carried out using Solidity for smart contract development and deployed on a test Ethereum blockchain via Remix, with DTs hosted on a cloud platform.

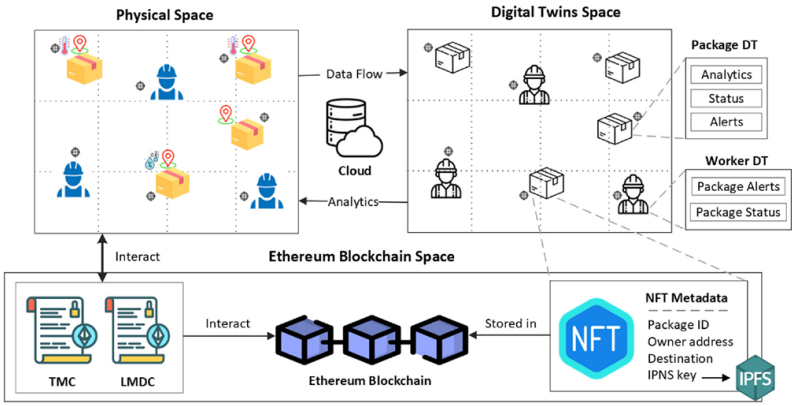


Figure 2.16: Digital Twin and blockchain framework for the Last Mile Delivery [45]

Francesca Guerriero et al. [61] implement a semi-decentralized last-mile delivery system by performing most computationally heavy tasks (Vehicle Routing Problem with Occasional Drivers) off-chain and using blockchain to handle trust, confirmations, and data integrity. Specifically, after customers request deliveries and an algorithm assigns the tasks to occasional drivers, these drivers confirm their shipment assignments on-chain; once deliveries are completed, customers also confirm receipt on-chain, triggering the automatic update of each driver's "score" (reliability metrics) in a smart contract. This ensures transparency and trust without excessive costs for ongoing heavy computations, since only these key confirmation steps require blockchain transaction fees.

Kadim Lahcen Nadime et al. propose a decentralized crowdsourced delivery system aimed at optimizing last-mile logistics in the e-commerce sector [99] (Figure 2.17). The use case revolves around enabling e-commerce companies to outsource deliveries to a dynamic pool of independent carriers, improving scalability, cost-efficiency, and delivery reliability. The project addresses critical challenges such as lack of transparency, trust, and accountability in traditional centralized logistics models. To solve these, the researchers implemented a blockchain-based application built on the Ethereum network, using solid-state-written smart contracts and employing the ERC-721 standard for uniquely identifying products. The system automates order creation, carrier bidding, delivery tracking, and payment processes through smart contracts while also generating tamper-proof proofs of delivery (PoD) and proofs of return (PoR). Blockchain's immutability ensures verifiable traceability and fraud prevention throughout the supply chain. The implementation was successfully tested using the Truffle and Ganache environments, confirming the feasibility and robustness of this decentralized approach. Therefore, the study not only defines a practical use case but also realizes a functional blockchain solution that contributes to the digital transformation of e-commerce logistics.

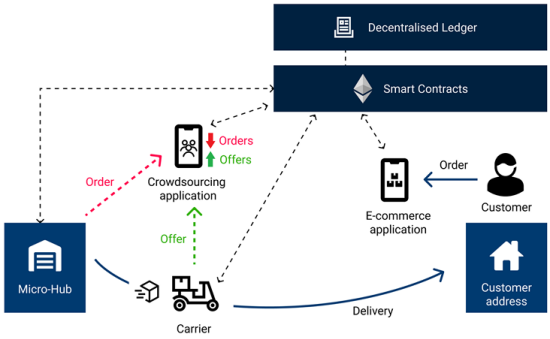


Figure 2.17: Workflow and Components of Decentralized Crowdsourced Delivery Model [99]

2.10.1 Research Gap and Contribution

Despite the increasing interest in leveraging blockchain technology for last-mile delivery, current research reveals several significant gaps when examined through the lens of vehicle routing optimization with hybrid fleets. Existing blockchain-based delivery systems primarily focus on decentralizing task allocation, ensuring traceability through smart contracts and NFTs, and enhancing trust via transparent transactions. However, they largely omit the integration of the Vehicle Routing Problem (VRP), particularly in scenarios involving a hybrid fleet composed of company-owned and crowdsourced vehicles. Only one study incorporates VRP principles, but consideration for fleet heterogeneity. Furthermore, economic efficiency and environmental sustainability (key concerns in hybrid fleet logistics) are insufficiently explored. These gaps underscore the need for an approach that combines blockchain, VRP algorithms, and hybrid fleet coordination to improve cost, performance, and sustainability in last-mile delivery.

This research develops a blockchain-based solution that integrates vehicle routing optimization with hybrid fleet management for last-mile delivery. Using Solana blockchain and Rust smart contracts, the system coordinates company-owned and crowdsourced vehicles while optimizing routes. The platform handles order creation to fulfillment, incorporating token-based payments, carbon tracking, and IPFS storage. Unlike existing approaches that separate blockchain and routing, this work demonstrates their practical integration to enhance transparency, reduce costs, and improve sustainability in delivery operations.

Conclusion

This chapter introduced blockchain as a powerful and emerging digital technology with the potential to reshape the way transactions and information are recorded, verified, and shared. Starting from the evolution of the internet, we saw how blockchain represents a shift toward a more decentralized and transparent digital world, often called the Internet of Value.

We explored how blockchain works, including its core components such as blocks, ledgers, transactions, hash functions, and peer-to-peer networks. We also looked at different types of blockchain systems and the important role of cryptography in protecting data and ensuring trust. In addition, we discussed consensus mechanisms like Proof of Work

and Proof of Stake, which allow blockchain networks to agree on shared records without needing a central authority.

The chapter also introduced smart contracts, which make it possible to automate agreements and processes securely. While blockchain offers many benefits such as transparency, security, and traceability, it also faces limitations, including scalability, energy consumption, and regulatory challenges.

Finally, we examined the role of blockchain in supply chains, with a focus on how it can improve traceability, reduce fraud, and increase trust between different actors. In particular, blockchain holds strong potential for solving problems in last-mile delivery by enabling real-time tracking and secure data sharing.

Chapter 3

Optimization of the VRP-TW with a Heterogeneous Fleet

Introduction

In this chapter, we focus on the Vehicle Routing Problem with Time Windows and Heterogeneous Fleet (H-VRP-TW), addressing the optimization of last-mile delivery using a mixed fleet composed of company-owned vehicles and crowdsourced drivers. This hybrid model reflects real-world practices where logistics operators combine internal resources with external on-demand delivery agents to meet rising customer expectations and handle variable delivery demand efficiently.

The primary objective of this study is to minimize total delivery costs while meeting key operational constraints such as vehicle capacities, time windows, and service time. The integration of crowdsourcing (also known as crowdshipping) into the routing model introduces additional decision-making complexity, due to varying cost structures, fixed activation fees, and limited load capacity.

We begin by formulating a mathematical model that captures the VRPTW in this hybrid context. The model incorporates several key parameters: vehicle capacities, delivery demands, travel distances and times, time windows for service at each client location, and cost differences between company-owned and crowd vehicles. The decision variables include route assignments for both vehicle types, visit sequences, customer arrival times, and activation variable for crowd drivers. The objective function is designed to minimize the total cost of travel and service across the fleet, accounting for both variable and fixed operational costs.

To solve this mono-objective optimization problem, we implement a Simulated Annealing (SA) Algorithm. SA is a robust approach for solving combinatorial optimization problems such as VRP, especially when dealing with heterogeneous fleets and strict delivery constraints. The algorithm generates and improves candidate solutions through local modifications (neighbor operations), guided by a probabilistic acceptance mechanism that allows escape from local minima.

We test our approach effectiveness using geographic and demand data based on a real urban setting in Tlemcen, Algeria, simulating deliveries in a city environment with multiple customer locations. The data set includes coordinates, distance matrices, travel time, service times, and time windows. Several scenarios are analyzed to evaluate the performance of the model, including the effects of crowd vehicle use, route balancing, and time feasibility.

Comparative analyses are conducted against solutions obtained from Gurobi which is an exact optimization solver, to evaluate the performance of the Simulated Annealing (SA) algorithm in terms of solution quality, total delivery cost, and computational efficiency when applied to the Vehicle Routing Problem with Time Windows using a heterogeneous fleet composed of company-owned and crowdsourced vehicles.

3.1 Problem description

In the proposed model, we are working on a last-mile delivery vehicle routing problem in the context of last mile delivery with a mixed fleet configuration: crowdsourced vehicles and company-owned vehicles. This problem addresses the operational challenge of routing a set of delivery vehicles to efficiently meet customer demands within specific time windows while optimizing resource utilization and minimizing transportation costs.

Customer demands are expressed as discrete units to be delivered within predefined service time intervals. The delivery operation is supported by a central depot where all vehicles start their routing and must return upon completion of their routes. Each vehicle type has its own capacity constraints, cost structures, and operational roles, with company vehicles typically handling closed routes starting and ending at the depot, while crowdsourced vehicles follow open routes and finish at their last customer to deliver. Crowdsourced vehicles incur both fixed and distance-based costs, whereas company vehicles' traveling costs are primarily related to the distance traveled.

The model aims to minimize the overall delivery cost, which includes both fixed and variable costs, while ensuring service quality by meeting time-window requirements and respecting capacity constraints. The number of vehicles used, their load levels, and delivery schedules are modeled as decision variables, enabling performance evaluation under various operational scenarios.

Given a set of n customer locations $\mathscr{C} = \{1, \ldots, n\}$ with associated demands and service time windows, a fleet of k crowdsourced vehicles c $\mathscr{V}_c = \{v_1, \ldots, v_k\}$, and a fleet of l company vehicles p $\mathscr{V}_p = \{v_1, \ldots, v_l\}$, the model constructs optimal delivery plans. These plans respect the individual capacity limits of each vehicle type, ensure that each customer is served exactly once, and guarantee feasibility within the specified time constraints.

3.2 Hypothesis

The proposed model addresses the last-mile delivery problem using a mixed fleet of company-owned and crowdsourced vehicles within an urban logistics framework. Before detailing the structure and components of the model, we outline the following key assumptions that guide its formulation:

- Delivery operations are managed from a single central depot, which serves a set of customers $C = \{c_1, \ldots, c_n\}$.
- The fleet is divided into two types of vehicles:
  - Crowdsourced vehicles: $\mathscr{V}_c = \{v_1^c, \ldots, v_k^c\}$, which operate under an open vehicle routing problem structure. Each crowdsourced vehicle starts its route from the depot and ends at the location of the last customer it serves.

  - Company-owned vehicles: $\mathscr{V}_p = \{v_1^p, \ldots, v_l^p\}$, which follow a closed vehicle routing problem structure. Each company vehicle starts and ends its route at the central depot.

- Each customer must be visited exactly once by a single vehicle and deliveries must be completed within predefined time frames.

- Each vehicle has a fixed capacity that must not be exceeded at any point during its route.

- Customers are not pre-assigned to any vehicle type, allowing routing decisions to be made dynamically based on current operational conditions and cost efficiency.

- Crowdsourced vehicles incur both a fixed usage cost and a variable cost based on the distance traveled. Company-owned vehicles incur only a variable cost per kilometer.

- All service times and travel durations are considered in the scheduling, ensuring that time window constraints are satisfied.

- The primary objective is to minimize the total delivery cost, which includes all fixed and variable cost related with distance, while ensuring feasible routing and timely delivery across the mixed fleet.

3.3 Problem formulation

3.3.1 Parameters

n	Number of customers
$\mathscr{C} = \{1, \ldots, n\}$	Set of customer locations
$\mathscr{V}_c = \{v_1^c, \ldots, v_k^c\}$	Set of crowdsourced vehicles
$\mathscr{V}_p = \{v_1^p, \ldots, v_l^p\}$	Set of company-owned vehicles
d_0	Depot location
d_i	Demand of customer i, $\forall i \in \mathscr{C}$
$[e_i, l_i]$	Time window during which customer i must be served
Q_c	Capacity of each crowdsourced vehicle
Q_p	Capacity of each company vehicle
c_v^{fix}	Fixed cost of using crowdsourced vehicle $v \in \mathscr{V}_c$
c_{ij}^{dist}	Distance cost between nodes i and j
t_{ij}	Travel time between nodes i and j
s_i	Service time required at customer i
M	A large constant

3.3.2 Decision Variables

$$x^c_{ijv} = \begin{cases} 1 & \text{if crowdsourced vehicle } v \text{ travels from node } i \text{ to } j \\ 0 & \text{otherwise} \end{cases}$$

$$x^p_{ijv} = \begin{cases} 1 & \text{if company vehicle } v \text{ travels from node } i \text{ to } j \\ 0 & \text{otherwise} \end{cases}$$

$$y^c_{iv} = \begin{cases} 1 & \text{if customer } i \text{ is served by crowdsourced vehicle } v \\ 0 & \text{otherwise} \end{cases}$$

$$y^p_{iv} = \begin{cases} 1 & \text{if customer } i \text{ is served by company vehicle } v \\ 0 & \text{otherwise} \end{cases}$$

$$z_v = \begin{cases} 1 & \text{if crowdsourced vehicle } v \text{ is used} \\ 0 & \text{otherwise} \end{cases}$$

$$\delta^c_{iv} = \begin{cases} 1 & \text{if customer } i \text{ is the last stop of crowdsourced vehicle } v \\ 0 & \text{otherwise} \end{cases}$$

u^c_{iv} Load carried by crowdsourced vehicle v after visiting customer i
u^p_{iv} Load carried by company vehicle v after visiting customer i
t^c_{iv} Arrival time of crowdsourced vehicle v at customer i
t^p_{iv} Arrival time of company vehicle v at customer i

3.3.3 Objective Function

$$\min \quad \sum_{v \in \mathscr{V}_c} \sum_{i \in \mathscr{N}} \sum_{\substack{j \in \mathscr{N} \\ j \neq i}} c^{\text{dist}}_{ij} \cdot x^c_{ijv} + \sum_{v \in \mathscr{V}_c} c^{\text{fix}}_v \cdot z_v + \sum_{v \in \mathscr{V}_p} \sum_{i \in \mathscr{N}} \sum_{\substack{j \in \mathscr{N} \\ j \neq i}} c^{\text{dist}}_{ij} \cdot x^p_{ijv} \tag{3.1}$$

The objective function 3.1 in this model is designed to minimize the total cost of last-mile delivery by optimally assigning delivery routes to two types of vehicle fleets: crowd-sourced vehicles and company-owned vehicles.

The first component relates to crowd-sourced vehicles, where each vehicle incurs a cost proportional to the distance traveled.

The second component is a one-time fixed activation cost incurred when a crowd-sourced vehicle leaves the depot, regardless of the number of deliveries it performs. This cost represents the compensation and logistical effort required to recruit and send a crowd worker to the depot for pickup.

The third component pertains to company-owned vehicles and includes a variable cost also proportional to the distance traveled. Unlike crowd-sourced vehicles, company-owned vehicles do not incur a fixed cost per use, as they are assumed to be part of the company's regular fleet operations.

3.3.4 Constraints

$$\sum_{v \in \mathscr{V}_c} y^c_{iv} + \sum_{v \in \mathscr{V}_p} y^p_{iv} = 1 \quad \forall i \in \mathscr{C} \tag{3.2}$$

$$\sum_{j \in \mathscr{C}} x^c_{d_0 jv} \leq 1 \quad \forall v \in \mathscr{V}_c \tag{3.3}$$

$$\sum_{\substack{i \in \mathscr{C} \cup \{d_0\} \\ i \neq j}} x^c_{ijv} - \sum_{\substack{i \in \mathscr{C} \cup \{d_0\} \\ i \neq j}} x^c_{jiv} = \delta^c_{jv} \quad \forall j \in \mathscr{C},\ \forall v \in \mathscr{V}_c \tag{3.4}$$

$$\sum_{\substack{i \in \mathscr{C} \cup \{d_0\} \\ i \neq j}} x^c_{ijv} = y^c_{jv} \quad \forall j \in \mathscr{C},\ \forall v \in \mathscr{V}_c \tag{3.5}$$

$$\sum_{j \in \mathscr{C}} x^p_{d_0 jv} \leq 1 \quad \forall v \in \mathscr{V}_p \tag{3.6}$$

$$\sum_{j \in \mathscr{C}} x^p_{d_0 jv} = \sum_{i \in \mathscr{C}} x^p_{id_0 v} \quad \forall v \in \mathscr{V}_p \tag{3.7}$$

$$\sum_{\substack{i \in \mathscr{C} \cup \{d_0\} \\ i \neq j}} x^p_{ijv} = \sum_{\substack{i \in \mathscr{C} \cup \{d_0\} \\ i \neq j}} x^p_{jiv} \quad \forall j \in \mathscr{C},\ \forall v \in \mathscr{V}_p \tag{3.8}$$

$$\sum_{\substack{i \in \mathscr{C} \cup \{d_0\} \\ i \neq j}} x^p_{ijv} = y^p_{jv} \quad \forall j \in \mathscr{C},\ \forall v \in \mathscr{V}_p \tag{3.9}$$

$$\sum_{i \in \mathscr{C}} d_i \cdot y^c_{iv} \leq Q_c \quad \forall v \in \mathscr{V}_c \tag{3.10}$$

$$\sum_{i \in \mathscr{C}} d_i \cdot y^p_{iv} \leq Q_p \quad \forall v \in \mathscr{V}_p \tag{3.11}$$

$$\sum_{j \in \mathscr{C}} x^c_{d_0 jv} \leq z_v \quad \forall v \in \mathscr{V}_c \tag{3.12}$$

$$z_v \leq \sum_{j \in \mathscr{C}} x^c_{d_0 jv} \quad \forall v \in \mathscr{V}_c \tag{3.13}$$

$$\sum_{i \in \mathscr{C}} \delta^c_{iv} = \sum_{j \in \mathscr{C}} x^c_{d_0 jv} \quad \forall v \in \mathscr{V}_c \tag{3.14}$$

$$\delta^c_{iv} \leq y^c_{iv} \quad \forall i \in \mathscr{C},\ \forall v \in \mathscr{V}_c \tag{3.15}$$

$$u^c_{jv} \geq u^c_{iv} + d_j - Q_c(1 - x^c_{ijv}) \quad \forall i \neq j \in \mathscr{C},\ \forall v \in \mathscr{V}_c \tag{3.16}$$

$$u^c_{iv} \geq d_i \cdot y^c_{iv} \quad \forall i \in \mathscr{C},\ \forall v \in \mathscr{V}_c \tag{3.17}$$

$$u^c_{iv} \leq Q_c \quad \forall i \in \mathscr{C},\ \forall v \in \mathscr{V}_c \tag{3.18}$$

$$u^p_{jv} \geq u^p_{iv} + d_j - Q_p(1 - x^p_{ijv}) \quad \forall i \neq j \in \mathscr{C},\ \forall v \in \mathscr{V}_p \tag{3.19}$$

$$u^p_{iv} \geq d_i \cdot y^p_{iv} \quad \forall i \in \mathscr{C},\ \forall v \in \mathscr{V}_p \tag{3.20}$$

$$u^p_{iv} \leq Q_p \quad \forall i \in \mathscr{C},\ \forall v \in \mathscr{V}_p \tag{3.21}$$

$$e_i \leq t^c_{iv} \leq l_i \quad \forall i \in \mathscr{C} \cup \{d_0\},\ \forall v \in \mathscr{V}_c \tag{3.22}$$
$$t^c_{jv} \geq t^c_{iv} + s_i + t_{ij} - M(1 - x^c_{ijv}) \quad \forall i \neq j \in \mathscr{C} \cup \{d_0\},\ \forall v \in \mathscr{V}_c \tag{3.23}$$
$$t^c_{d_0 v} = e_{d_0} \quad \forall v \in \mathscr{V}_c \tag{3.24}$$

$$e_i \leq t^p_{iv} \leq l_i \quad \forall i \in \mathscr{C} \cup \{d_0\},\ \forall v \in \mathscr{V}_p \tag{3.25}$$
$$t^p_{jv} \geq t^p_{iv} + s_i + t_{ij} - M(1 - x^p_{ijv}) \quad \forall i \neq j \in \mathscr{C} \cup \{d_0\},\ \forall v \in \mathscr{V}_p \tag{3.26}$$
$$t^p_{d_0 v} = e_{d_0} \quad \forall v \in \mathscr{V}_p \tag{3.27}$$

Variable Domains

$$x^c_{ijv}, x^p_{ijv}, y^c_{iv}, y^p_{iv}, z_v, \delta^c_{iv} \in \{0,1\} \tag{3.28}$$
$$u^c_{iv} \in [0, Q_c], \quad u^p_{iv} \in [0, Q_p] \tag{3.29}$$
$$t^c_{iv}, t^p_{iv} \geq 0 \tag{3.30}$$

3.3.5 Constraints Explanation

Customer Visit Constraint. Constraint 3.2 ensures that each customer $i \in \mathscr{C}$ is visited exactly once, either by a company-owned or a crowdsourced vehicle. This guarantees that all client demands are satisfied without duplication.

Crowdsourced Vehicle Routing Constraints. Constraint 3.3 limits each crowdsourced vehicle $v \in \mathscr{V}_c$ to a single departure from the depot, reflecting the open nature of crowdsourced routes. Constraint 3.4 enforces flow conservation by ensuring that, for each customer node j, the difference between incoming and outgoing arcs equals the binary indicator δ^c_{jv}, which identifies the last client visited. Constraint 3.5 links routing and assignment by ensuring that a customer is assigned to a vehicle if and only if that vehicle routes to the customer.

Company-Owned Vehicle Routing Constraints. Constraint 3.6 ensures that each company vehicle leaves the depot at most once. Constraint 3.7 enforces a return to the depot, thus modeling a closed route. Constraint 3.8 maintains flow conservation by ensuring that the number of arrivals to and departures from each client node is balanced. Constraint 3.9 guarantees that if a vehicle visits a client, the corresponding assignment variable reflects this.

Vehicle Capacity Constraints. Constraints 3.10 and 3.11 ensure that the total quantity delivered by each vehicle does not exceed its respective capacity, Q_c for crowdsourced vehicles and Q_p for company-owned vehicles.

Crowdsourced Vehicle Activation. Constraints 3.12 and 3.13 establish a logical equivalence between the activation variable z_v and the routing decisions of crowdsourced vehicle $v \in \mathscr{V}_c$. Specifically, constraint 3.12 ensures that if $z_v = 0$, then the vehicle cannot depart from the depot to any customer, i.e., $x^c_{d_0 jv} = 0$ for all $j \in \mathscr{C}$. Conversely, constraint 3.13

guarantees that if the vehicle departs to at least one customer, then it is marked as active, i.e., $z_v = 1$.

Together, these constraints enforce the following bidirectional condition:

$$z_v = \begin{cases} 1 & \text{if vehicle } v \text{ is used to serve at least one customer,} \\ 0 & \text{otherwise.} \end{cases}$$

This formulation ensures that the fixed activation cost associated with each crowdsourced vehicle is incurred if and only if the vehicle is effectively used in the solution.

Crowdsourced Routes. Constraint 3.14 ensures that the number of customers marked as the last visited is equal to the number of vehicle departures from the depot. Constraint 3.15 ensures that only customers assigned to a vehicle may be designated as its final client.

Load Continuity and Subtour Elimination . Constraints 3.16–3.18 and 3.19–3.21 ensure feasible load propagation using the Miller-Tucker-Zemlin formulation. They eliminate subtours and guarantee that vehicle capacities are not violated while satisfying customers demands.

Time Window Constraints for Crowdsourced Vehicles. Constraint 3.22 imposes the service time window $[e_i, l_i]$ for each customer and depot. Constraint 3.23 ensures that travel and service durations are respected between two sequentially visited nodes. Constraint 3.24 fixes the initial time at the depot to the earliest feasible time.

Time Window Constraints for Company Vehicles. Similar to the crowdsourced case, Constraints 3.25–3.27 enforce temporal feasibility and ensure that deliveries are made within customer-specified time intervals, maintaining schedule consistency for company-owned vehicles.

Variable Domains. Constraint 3.28 defines binary decision variables for routing (x^c_{ijv}, x^p_{ijv}), assignment (y^c_{iv}, y^p_{iv}), vehicle usage (z_v), and last-customer indicators (δ^c_{iv}). Constraint 3.29 restricts the cumulative load variables u^c_{iv}, u^p_{iv} to the interval $[0, Q_c]$ or $[0, Q_p]$, respectively. Constraint 3.30 ensures non-negativity for all arrival time variables.

3.4 Resolution approach

3.4.1 Exact Resolution

The mathematical model was formulated using the Pyomo 6.9.1 optimization modeling framework in Python 3.13.2 and solved with the Gurobi Optimizer version 12.0.1 on a 12th Gen Intel(R) Core(TM) i7-12700H CPU with 16 GB RAM, running Windows 11.

Figure 3.1: Resolution approach

Pyomo is a Python-based open-source optimization modeling language that provides a versatile and expressive platform for formulating, analyzing, and solving complex mathematical optimization problems. It supports a wide spectrum of problem types, including linear programming (LP), mixed-integer programming (MIP), quadratic programming (QP), nonlinear programming (NLP), stochastic programming, bilevel programming, and differential algebraic equations. One of Pyomo's core strengths lies in its ability to model structured and symbolic optimization problems while seamlessly integrating with Python's full-featured programming capabilities. This allows users to benefit from rich data handling, preprocessing, and visualization libraries, making Pyomo particularly suitable for building flexible and reproducible optimization workflows. Unlike traditional algebraic modeling languages such as AMPL, AIMMS, or GAMS, Pyomo models are embedded directly in Python, enabling dynamic and programmatic model construction. Pyomo interfaces with both commercial solvers, such as Gurobi and CPLEX, and open-source alternatives, making it widely adopted in both academic research and industrial applications [22].

The following code snippet illustrates the solver setup used in the Python environment:

```
import pyomo.environ as pyo
# ... model definition goes here
solver = pyo.SolverFactory('gurobi')
```

3.4.2 Metaheuristic

Simulated annealing

Simulated Annealing (SA) is a metaheuristic local search algorithm designed to solve complex discrete and continuous optimization problems. Inspired by the physical process of annealing in metallurgy, SA probabilistically accepts both improvements and certain degradations in the objective function to escape local optima and search for the global optimum, see figure 3.2. In each iteration, a neighboring solution is generated and accepted based on the Metropolis criterion, where inferior solutions may still be accepted with a probability that decreases over time according to a cooling schedule. As the system's "temperature" is gradually reduced, the algorithm increasingly focuses on refining the best solutions, converging toward a global or near-global optimum. Due to its flexible structure and ability to escape local minima, Simulated Annealing is particularly suitable for complex combinatorial problems such as the Vehicle Routing Problem (VRP), where the search space is vast and highly multi-modal (multiple hill climbing)[103].

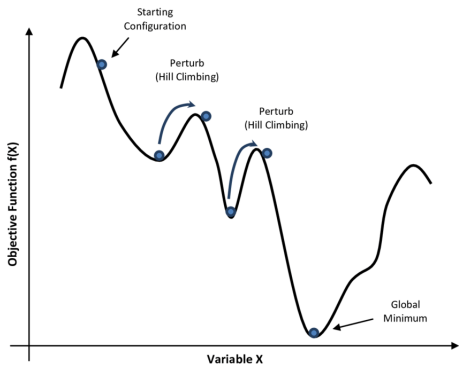


Figure 3.2: Simulated Annealing

The Simulated Annealing (SA) algorithm shown in Figure 3.3 starts by initializing the system temperature T at a high value, denoted $T = T_{\max}$. An initial solution x is randomly generated and its associated energy (objective function value) $E(x)$ is calculated. Subsequently, the algorithm enters an iterative loop where a new candidate solution x_{new} is generated, and its energy $E(x_{\text{new}})$ is evaluated. The energy variation 3.31 between the new and current solutions is then calculated according to:

$$\Delta E = E(x_{\text{new}}) - E(x) \tag{3.31}$$

If $\Delta E < 0$, the new solution x_{new} is accepted immediately, as it represents an improvement. However, if $\Delta E \geq 0$, the algorithm applies the Metropolis acceptance criterion 3.32: the new solution is accepted with a probability

$$p = \exp\left(-\frac{\Delta E}{T}\right) \tag{3.32}$$

The algorithm then generates an r which which is a randomly generated number in the range $[0,1]$. The new solution is accepted if $r < p$. Otherwise, the current solution is retained.

After each iteration, the temperature T is decreased according to a cooling schedule, typically $T \leftarrow \alpha \times T$ where $0 < \alpha < 1$. The iterative process continues until a stopping condition is met, which is either when the temperature falls below a limit $T_{\min}$ or when the solution energy $E(x)$ reaches a predefined acceptable value E_{th}. Finally, the algorithm outputs the best solution found.

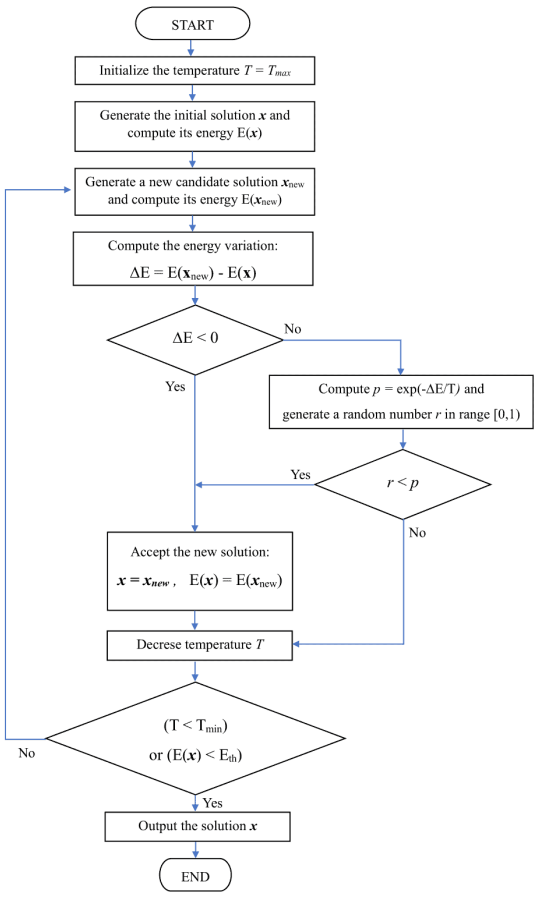


Figure 3.3: Flowchart of the simulated annealing algorithm

Solution Encoding

In our scenario, we have adopted a hybrid fleet-based vector encoding approach, combining two separate vehicle categories: company-owned and crowdsourced vehicles. The encoding structure captures the routing plan of each fleet type within a unified representation .

The first segment corresponds to the company fleet, where each vehicle route starts and ends at the depot, reflecting the closed nature of company vehicle operations. The second segment addresses the crowdsourced fleet, where vehicles start from the depot but are allowed to terminate their routes at the last served client, representing an open routing structure.

Each vehicle's route is a sequence of customer indices it must visit, with node 0 reserved exclusively for the central depot. In this structure:

- **Company vehicle routes** are enclosed between depot visits (start and end at node 0).
- **Crowd vehicle routes** begin at the depot (node 0) but do not necessarily return.

The overall encoding is organized as a dictionary with two distinct parts:

- `company`: lists the routes for company vehicles.
- `crowd`: lists the routes for crowdsourced vehicles.

In a general case with N nodes (one depot and $N-1$ clients), K_c company vehicles, and K_p crowdsourced vehicles, the indexing is defined as:

- The depot is indexed as node 0.

- Clients are indexed from 1 to $N-1$.

- The company fleet is described first, followed by the crowd fleet.

Consider a scenario with five clients (nodes 1 to 6), two company vehicles ($K_c = 2$), and two crowdsourced vehicles ($K_p = 2$). The encoding structure would be:

```
sol = {
    company: [[0, 2, 5, 0], [0, 3, 4, 0]],
    crowd: [[0, 1, 6], [  ]]
}
```

In this encoding:

- Company Vehicle 1 starts from the depot (0), serves clients 2 and 5, and returns to the depot.

- Company Vehicle 2 starts from the depot, serves clients 3 and 4, and returns to the depot.

- Crowd Vehicle 1 starts from the depot, serves client 1 , and ends his routing with client 6.

- Crowd Vehicle 2 remains unused in this example and is deactivated.

The proposed encoding is designed to facilitate the following:

Constraint management Capacity and time window constraints are verified individually for each route during the evaluation of the solution, as shown in the algorithm 1:

Algorithm 1: Constraint Management

Input: Solution *sol*, Vehicle capacities Q_c, Q_p, Time windows $\{[e_i, l_i]\}$, Travel times t_{ij}, Service time s
Output: Feasibility status of the solution
for *each route r in* `company` *fleet* **do**
    Initialize cumulative load $L \leftarrow 0$;
    Initialize current time $T \leftarrow e_0$;
    for *each consecutive pair* (u, v) *in r* **do**
        $L \leftarrow L + \text{demand}(v)$;
        if $L > Q_c$ **then**
            return ***Infeasible****: Company vehicle capacity exceeded*;
        $T \leftarrow T + s + t_{uv}$;
        if $T < e_v$ **then**
            $T \leftarrow e_v$; `// Wait if early`
        if $T > l_v$ **then**
            return ***Infeasible****: Time window violated*;
for *each route r in* `crowd` *fleet* **do**
    Initialize cumulative load $L \leftarrow 0$;
    Initialize current time $T \leftarrow e_0$;
    for *each consecutive pair* (u, v) *in r* **do**
        $L \leftarrow L + \text{demand}(v)$;
        if $L > Q_p$ **then**
            return ***Infeasible****: Crowdsourced vehicle capacity exceeded*;
        $T \leftarrow T + s + t_{uv}$;
        if $T < e_v$ **then**
            $T \leftarrow e_v$; `// Wait if early`
        if $T > l_v$ **then**
            return ***Infeasible****: Time window violated*;
return ***Feasible***;

Neighborhood generation Solutions variations can be easily implemented through operations such as client swapping, segment reversal, and insertion across or within routes, see algorithm 2.

Algorithm 2: Neighborhood Generation

Input: Current solution *sol*, Reversal probability p_{rev}, Crossover probability p_{cross}
Output: New neighbor solution
Create a deep copy of *sol* as *new*;
Randomly choose fleet $\mathscr{F} \in \{company, crowd\}$ (if available);
if *Random number* $< p_{rev}$ **then**
    Select a random route r from fleet $\mathscr{F}$;
    if *r has sufficient stops for reversal* **then**
        Randomly select two distinct indices i and j ($1 \leq i < j < |r| - 1$);
        Reverse the segment $r[i : j]$;
else
    if *Random number* $< p_{cross}$ **then**
        Select two different routes r_1 and r_2 from fleet $\mathscr{F}$;
        if *both* r_1 *and* r_2 *have sufficient clients for exchange* **then**
            Randomly select client c_1 from r_1 and c_2 from r_2;
            Swap c_1 and c_2;
    else
        Select a route r_1 from fleet $\mathscr{F}$;
        if r_1 *contains removable clients* **then**
            Randomly select a client c from r_1 (excluding depot);
            Remove c from r_1;
            Randomly choose fleet $\mathscr{F}' \in \{company, crowd\}$ (if available);
            Select a route r_2 from fleet $\mathscr{F}'$;
            Insert c into r_2 at a random position (excluding depot);
return *new*

Cost calculation The travel distance for each route is calculated based on the specific cost structure of the type of vehicle (fixed and distance-based costs for crowd-sourced vehicles, distance-based costs for company vehicles), see algorithm 3.

Algorithm 3: Cost calculation

Input: Solution sol, Distance costs c_{ij}, Fixed cost for crowd vehicles c_{fixed}, Vehicle capacities Q_c, Q_p, Time windows $\{[e_i, l_i]\}$, Travel times t_{ij}, Service time s

Output: Total cost of the solution

Initialize total cost $C \leftarrow 0$;
for *each route r in* `company` *fleet* **do**
    Initialize cumulative load $L \leftarrow 0$;
    Initialize current time $T \leftarrow e_0$;
    for *each consecutive pair* (u, v) *in r* **do**
        $C \leftarrow C + c_{uv}$;
        $L \leftarrow L +$ demand of node v;
        $T \leftarrow T + s + t_{uv}$;
        if $T < e_v$ **then**
            $T \leftarrow e_v$;
for *each route r in* `crowd` *fleet* **do**
    if *route r is not empty* **then**
        $C \leftarrow C + c_{\text{fixed}}$;
    Initialize cumulative load $L \leftarrow 0$;
    Initialize current time $T \leftarrow e_0$;
    for *each consecutive pair* (u, v) *in r* **do**
        $C \leftarrow C + c_{uv}$;
        $L \leftarrow L +$ demand of node v;
        $T \leftarrow T + s + t_{uv}$;
        if $T < e_v$ **then**
            $T \leftarrow e_v$;
return C

Simulated Annealing for the VRP-TW and Heteregeneous Fleet

The simulated annealing flowchart 3.4 represents a metaheuristic optimization technique that combines random exploration with a temperature-controlled acceptance mechanism to solve vehicle routing problems with combined company and crowd-sourced vehicles. Starting with an initial solution and high temperature, the algorithm iteratively explores neighboring solutions while gradually decreasing temperature, accepting better solutions automatically and occasionally accepting worse solutions based on a probability function that depends on both solution quality difference and current temperature. The algorithm ends when either the temperature falls below a minimum limit or a satisfactory solution quality is achieved.

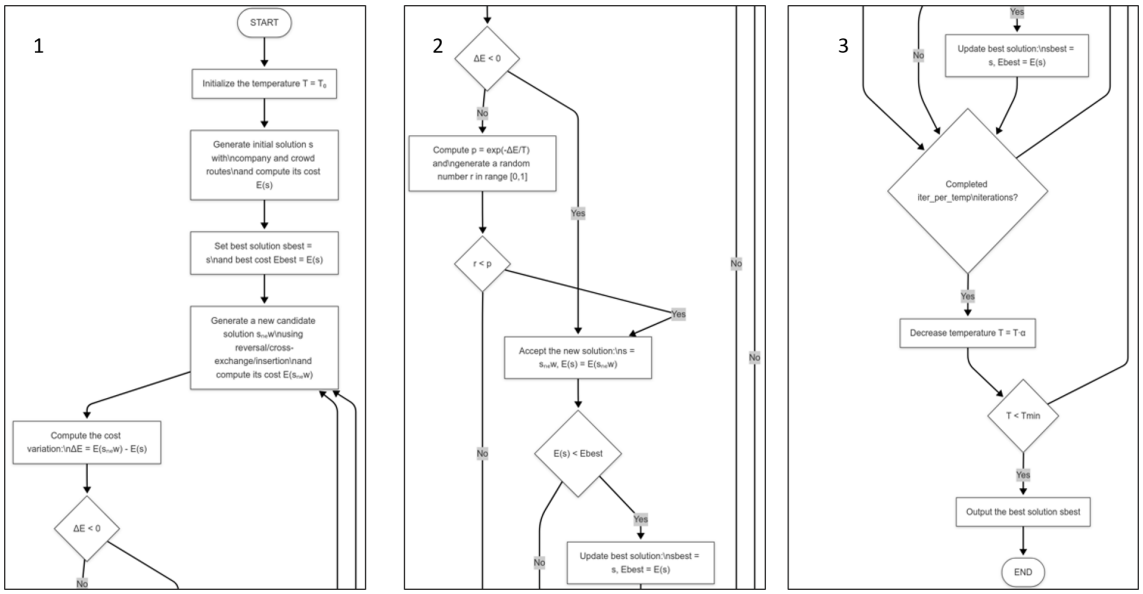


Figure 3.4: Flowchart for SA for VRP-TW-HF

The Algorithm 4 represents a snippet of the implementation of Simulated Annealing in our model. The total cost function combines distance-based transportation costs, fixed costs for crowdsourced vehicle activation, and penalties for violations of capacities and time windows.

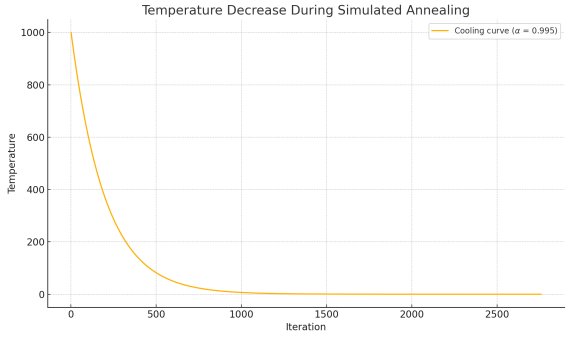


Figure 3.5: Temperature decrease with $\alpha = 0.995$ starting from $T_0 = 1000$ until reaching $T_{\min} = 10^{-3}$.

Starting from an initial solution, Simulated Annealing iteratively generates neighboring solutions through segment reversals, client exchanges between routes, and client insertions across fleets. A new solution is accepted if it improves the cost or, if worse, with a probability defined by the Metropolis criterion to encourage exploration. The temperature progressively decreases according to a cooling schedule, reducing the acceptance of inferior solutions over time; see figure 3.5. The algorithm terminates when the system cools below a threshold, returning the best solution found.

Algorithm 4: Simulated Annealing VRP-TW-HF

Input: Initial solution x, Data (demands, distances, time windows, service times, vehicle capacities), Initial temperature T_0, Cooling rate α, Minimum temperature $T_{\min}$

Output: Best solution found

Initialize $x_{\text{best}} \leftarrow x$, compute initial cost $E(x)$ considering:

- Distance-based costs for company and crowdsourced vehicles,
- Fixed cost for each activated crowdsourced vehicle,
- Penalties for vehicle capacity violations,
- Penalties for time window violations.

Set $T \leftarrow T_0$;

while $T > T_{min}$ **do**

  for *each iteration at current temperature* **do**

    Generate a neighbor solution x_{new} by randomly applying:

    - **Segment Reversal:** Reverse a subsequence within a company or crowd route,
    - **Cross-Exchange:** Swap clients between two different routes (company $\leftrightarrow$ company, crowd $\leftrightarrow$ crowd),
    - **Client Insertion:** Move a client from one route to another (across fleets).

    Evaluate cost $E(x_{\text{new}})$, considering updated routes and penalties;

    Compute cost difference $\Delta E = E(x_{\text{new}}) - E(x)$;

    if $\Delta E < 0$ **then**

      Accept x_{new}: set $x \leftarrow x_{\text{new}}$;

    else

      Accept x_{new} with probability $p = \exp\left(-\frac{\Delta E}{T}\right)$;

    if $E(x) < E(x_{best})$ **then**

      Update best solution: $x_{\text{best}} \leftarrow x$;

  Decrease temperature: $T \leftarrow \alpha \times T$;

return x_{best};

3.5 Application

3.5.1 Data

The data set used in this study was inspired by the structure of the Solomon benchmark instances [95, 127], and was adapted to represent a realistic urban last-mile delivery scenario centered in Tlemcen, Algeria. Geographic coordinates were generated using the OpenRouteService platform, which also provided the distance matrix and travel times between nodes [54].

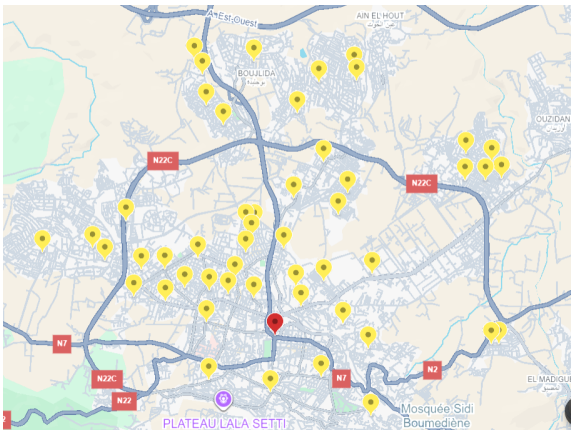


Figure 3.6: Geographic map illustrating client locations (in yellow) and the depot location (in red)

Client demands were randomly assigned within a range of 1 to 9 units, consistent with typical parcel delivery operations. Company-owned vehicles with a capacity of 65 units were modeled, while crowd-sourced vehicles with a capacity of 40 units. Each vehicle type was associated with distinct cost structures: company vehicles incurred only distance-based operational costs (0.8 currency units per kilometer), whereas crowdsourced vehicles also incurred a fixed activation cost (15 currency units) in addition to a higher variable cost (1.0 currency units per kilometer). Each customer was assigned an individual time window to represent service availability constraints, while the depot was configured to operate between 8:00 AM and 6:00 PM. Furthermore, a random service time ranging from 10 to 15 minutes was assigned to each client to reflect variations in handling and delivery.

Table 3.1: Summary of Input Data Used for the Vehicle Routing Problem

Parameter	**Value/Description**
Number of Nodes	*N* (1 depot + *N-1* clients).
Distance Matrix	Realistic distances based on coordinates (in kilometers).
Travel Time Matrix	Travel times computed realistically (in minutes).
Demand per Client	Random values between 1 and 9 units.
Crowd Vehicle Capacity	35 units.
Company Vehicle Capacity	65 units.
Traveling Cost for Company Vehicles	0.8 currency units per kilometer.
Traveling Cost for Crowd Vehicles	1.0 currency units per kilometer.
Fixed Cost for Crowd Vehicles	15 currency units per activated vehicle.
Service Time at Each Node	Randomly from 10 to 15 minutes.
Time Windows	Defined for each client between 8AM to 6PM.
Number of Company Vehicles	3 vehicles.
Number of Crowd Vehicles	4 vehicles.

This dataset was employed to validate both the mathematical programming model solved via Gurobi and the Simulated Annealing metaheuristic developed for larger or more complex instances.

3.5.2 Scenarios

The model was evaluated and tested through six different scenarios, each characterized by variations in fleet composition and time window constraints. Only one depot is used across all scenarios. Additionally, we tested each scenario with 3 different instances of nodes N (15, 20, and 30) to ensure robust validation. The details of the scenarios are summarized as follows 3.2:

Scenarios	Number of nodes	Fleet	Time window
1	15	Company only	Hard
	20		
	30		
2	15	Company only	Light
	20		
	30		
3	15	Crowd only	Hard
	20		
	30		
4	15	Crowd only	Light
	20		
	30		
5	15	Company and crowd	Hard
	20		
	30		
6	15	Company and crowd	Light
	20		
	30		

Table 3.2: Scenario configuration based on fleet type, number of nodes, and time window conditions.

Since the performance of the simulated annealing metaheuristic is highly dependent on its parameter configuration, a sensitivity analysis was conducted to identify the optimal parameter values presented in Table 3.3:

Table 3.3: Simulated Annealing Algorithm Parameters

Parameter	Value
Initial Temperature (T_0)	1000
Cooling Rate (α)	0.995
Iterations per Temperature	300
Stopping Temperature	10^{-3}

3.5.3 Results and discussion

Table 3.4 and Figure 3.7 present a comparative evaluation of the solutions obtained by the exact solver Gurobi and the Simulated Annealing (SA) across six different operational scenarios. Each scenario is characterized by varying fleet configurations (company-only,

crowd-only, mixed) and time window Hardness (Hard or Light). The table details the objective function values obtained by both methods, the corresponding optimality gap (GAP), and the computational times (CPU time) measured in seconds for both approaches. Additionally, average CPU times and average GAP across different numbers of clients (15, 20, and 30) are reported for each scenario.

Table 3.4: Performance Comparison of Gurobi and SA Algorithms

Scen.	Nodes	Gurobi	SA	GAP	CPU G	CPU SA	Avg CPU SA	Avg CPU G	Avg GAP
1	15	31.78	32.80	0.0310	2.0	7.0	24.77	21.63	0.0350
	20	42.21	43.83	0.0370	3.9	42.1			
	30	98.80	102.60	0.0371	59.0	25.2			
2	15	24.65	24.65	0.0000	12.3	15.0	33.30	792.77	0.0134
	20	28.85	28.05	0.0000	252.0	30.9			
	30	42.45	44.22	0.0400	2114.0	54.0			
3	15	89.64	97.03	0.0761	2.0	18.0	18.00	18.33	0.0892
	20	93.83	103.85	0.0965	22.0	24.0			
	30	120.22	132.82	0.0949	31.0	12.0			
4	15	93.97	94.64	0.0071	15.0	11.0	16.87	615.33	0.0166
	20	95.32	95.91	0.0062	35.0	2.6			
	30	102.10	105.98	0.0366	1796.0	37.0			
5	15	31.78	31.78	0.0000	5.0	15.2	23.27	33.33	0.0439
	20	54.38	57.02	0.0463	25.0	18.6			
	30	98.72	107.94	0.0854	70.0	36.0			
6	15	24.65	24.65	0.0000	12.3	15.0	39.97	191.43	0.0188
	20	28.05	28.05	0.0000	252.0	30.9			
	30	55.78	59.12	0.0565	310.0	74.0			

Where GAP is calculated as: $\text{GAP} = \frac{\text{SA}-\text{Gurobi}}{\text{SA}}$

Scenario 1 (Company-only, Hard Time Windows):

- The small GAP (around 3–4%) indicates that SA performs very close to Gurobi, providing near-optimal solutions.
- SA execution time is slightly higher compared to Gurobi for 20 clients but remains acceptable overall.

Scenario 2 (Company-only, Light Time Windows):

- At 15 and 20 clients, the GAP is 0%, meaning SA perfectly matches Gurobi. Only a small deviation (4%) appears at 30 clients.
- Despite the problem size increase, SA remains substantially faster than Gurobi, particularly when Gurobi's computation time grows considerably.

Scenario 3 (Crowd-only, Hard Time Windows):

- The moderate GAP (around 9%) indicates that SA solutions are less optimal compared to Gurobi but still within an acceptable range.
- SA maintains reasonable computation times even though the problem complexity increases with Hard crowd-sourced routing.

Scenario 4 (Crowd-only, Light Time Windows):

- The very low GAP (0.6–3.6%) highlights that SA performs almost as well as Gurobi under Light conditions.
- SA significantly reduces computation times compared to Gurobi, whose performance deteriorates as the instance size increases.

Scenario 5 (Mixed Fleet, Hard Time Windows):

- A moderate GAP (4.6–8.5%) is observed as the number of clients increases, reflecting the increased difficulty of mixed fleet management under Hard constraints.
- SA maintains acceptable computational efficiency, providing feasible solutions faster than Gurobi.

Scenario 6 (Mixed Fleet, Light Time Windows):

- SA maintains solution quality, with GAP% remaining below 6% even for larger instances.
- SA shows substantial computational savings compared to Gurobi, confirming its suitability for large-scale mixed fleet problems with Light constraints.

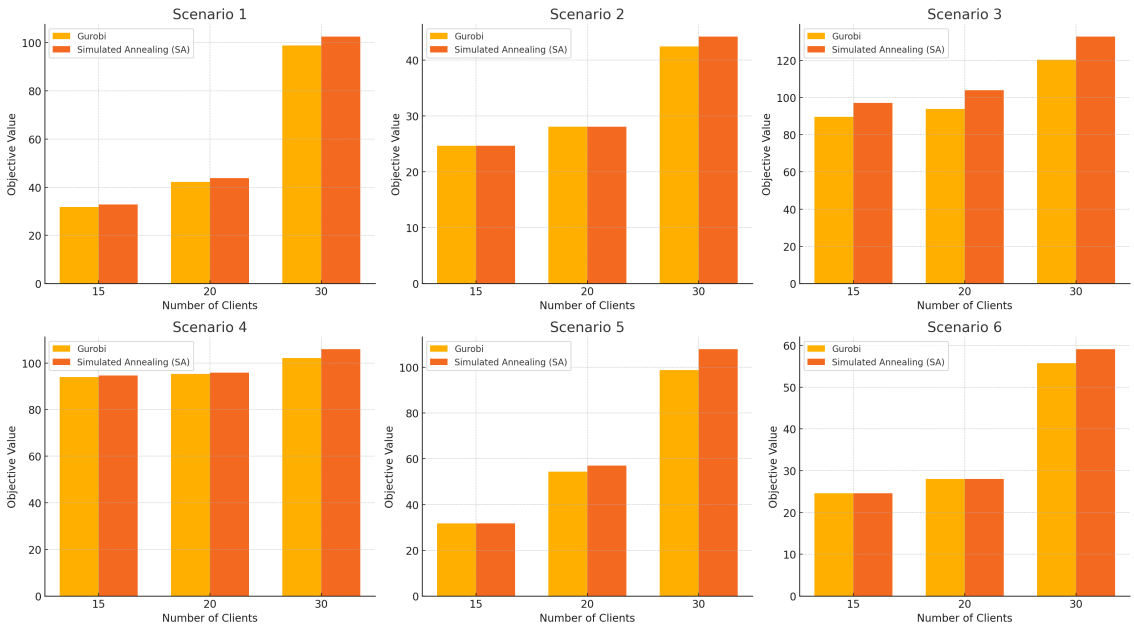


Figure 3.7: Objective function comparison between Gurobi and SA

Overall GAP Analysis:

- The GAP between Gurobi and SA increases slightly with the complexity and size of the problem, particularly in scenarios involving Hard time windows and crowd-only fleets.
- SA, does not guarantee global optimality but consistently achieves near-optimal solutions within significantly reduced computational times.
- Even in the most complex cases (Scenario 3 and Scenario 5), SA maintains GAP values within a practical range (under 10%), confirming its viability for large, real-world delivery problems where computational resources and time are constrained.

Routing results for large-scale instances

Figures 3.8 to 3.13 present a graphical representation of the routing solutions and client service schedules under different fleet configurations and time window constraints for 30 nodes. Each figure is composed of two subplots: One side shows the spatial distribution of routes across clients and depot, the other side displays the service times for each client throughout the day.

Figures 3.8 and 3.9 correspond to the mixed fleet configuration, combining company-owned and crowd-sourced vehicles under Hard and Light time windows. Figures 3.10 and 3.11 represent the routing solutions for a company-only fleet under Hard and Light time window settings. Finally, Figures 3.12 and 3.13 illustrate the results when only crowd-sourced vehicles are used, again considering both Hard and Light time constraints.

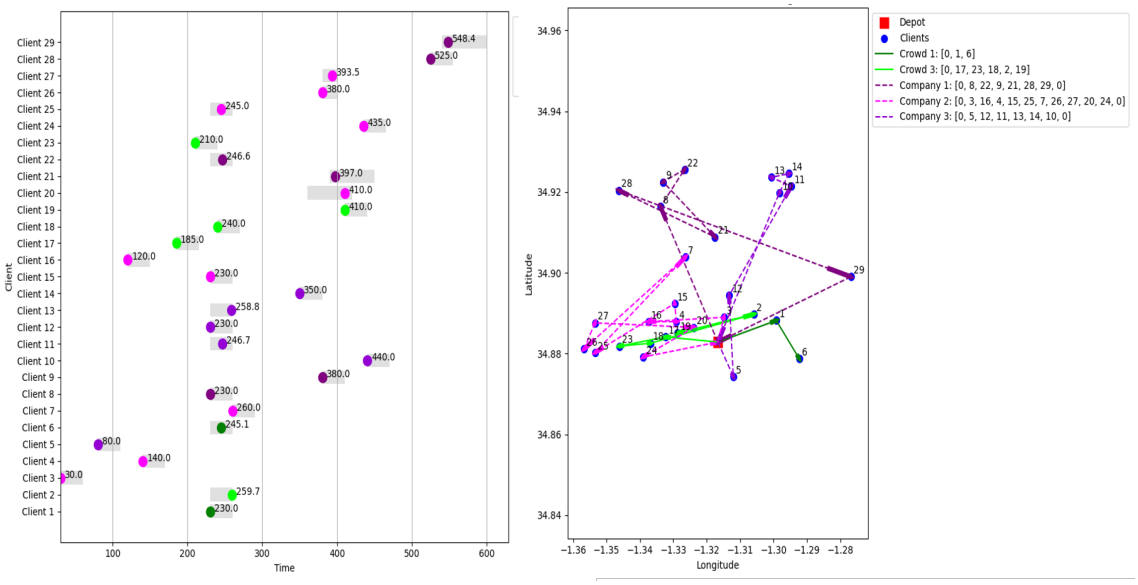


Figure 3.8: 30 Nodes with Hard time window- Mixed Fleet

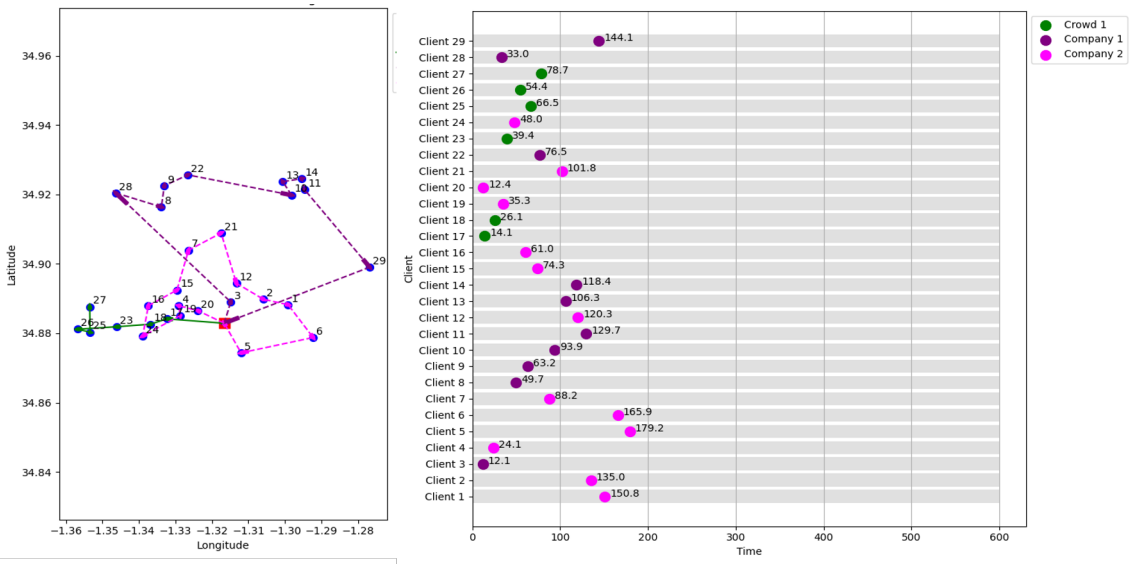


Figure 3.9: 30 Nodes with Light time window- Mixed Fleet

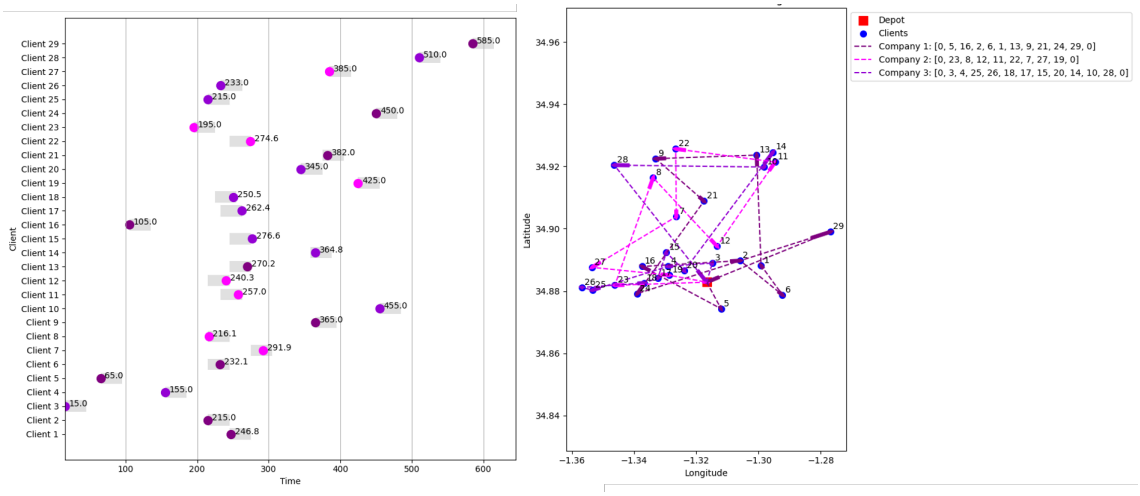


Figure 3.10: 30 Nodes with Hard time window- Company only

Figures 3.8 to 3.13 reveal how the type of fleet composition and the nature of time window constraints impact service efficiency and customer satisfaction.

Under Hard time windows (Figures 3.8, 3.10, and 3.12), crowd-sourced vehicles play a critical role in enhancing service responsiveness. In Figure 3.8 , it is observed that the crowd fleet enables operational effectiveness during peak hours by reducing delivery bottlenecks and ensuring that client servicing is more evenly distributed across the delivery horizon. This flexibility can complement company-owned vehicles by facilitating earlier deliveries, as compared to the company-only configuration shown in Figure 3.10, where deliveries tend to be more clustered and delayed.

The mixed fleet configuration under Hard constraints (Figure 3.8) offers a balanced strategy. The inclusion of crowd vehicles enables the system to advance certain deliveries earlier in the day, while company vehicles maintain scheduled service for others. This hybrid approach effectively reduces congestion during peak hours around midday, improving overall service dispersion across the timeline.

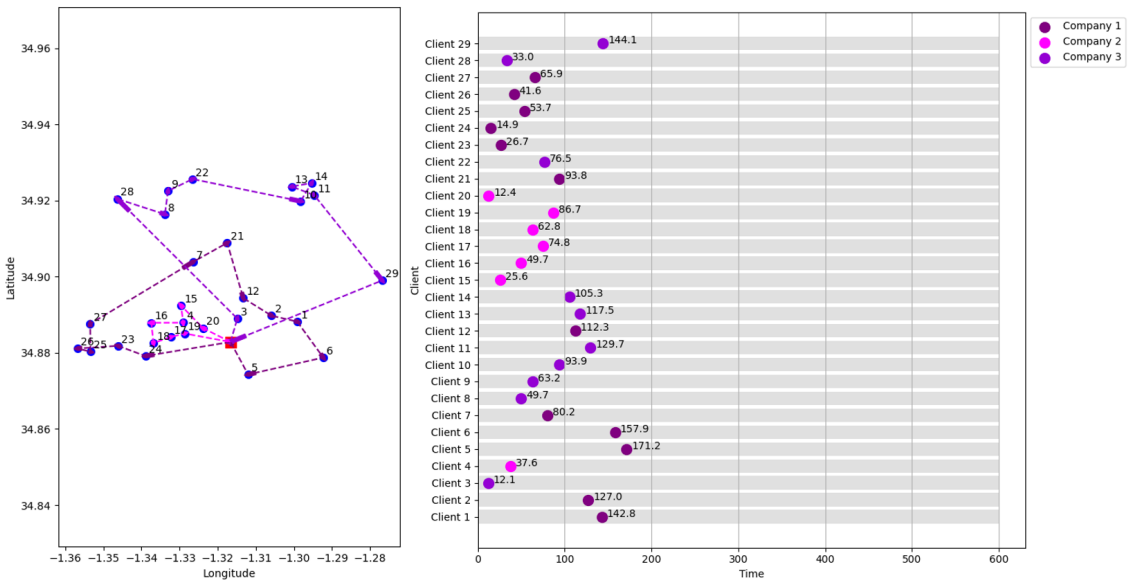


Figure 3.11: 30 Nodes with Light time window-Company only

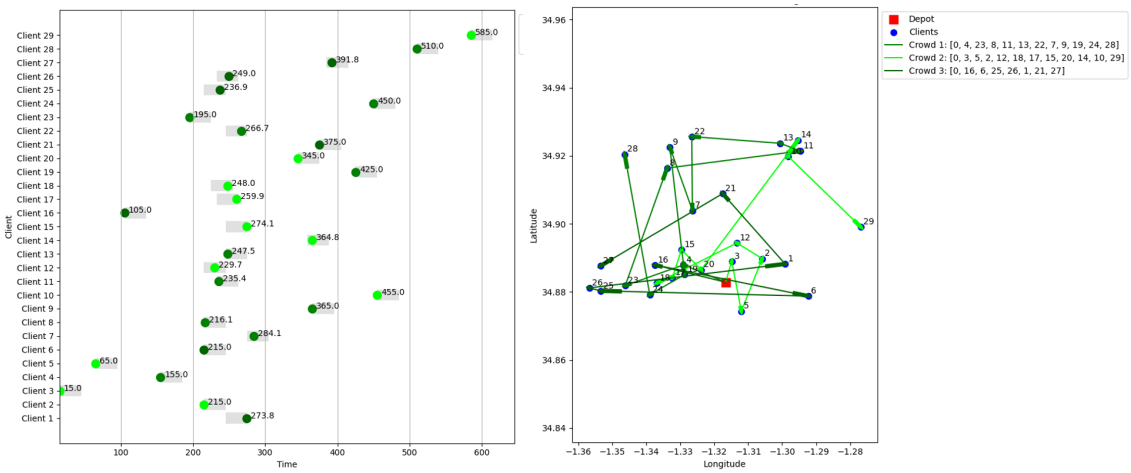


Figure 3.12: 30 Nodes with Hard time window- Crowd Only

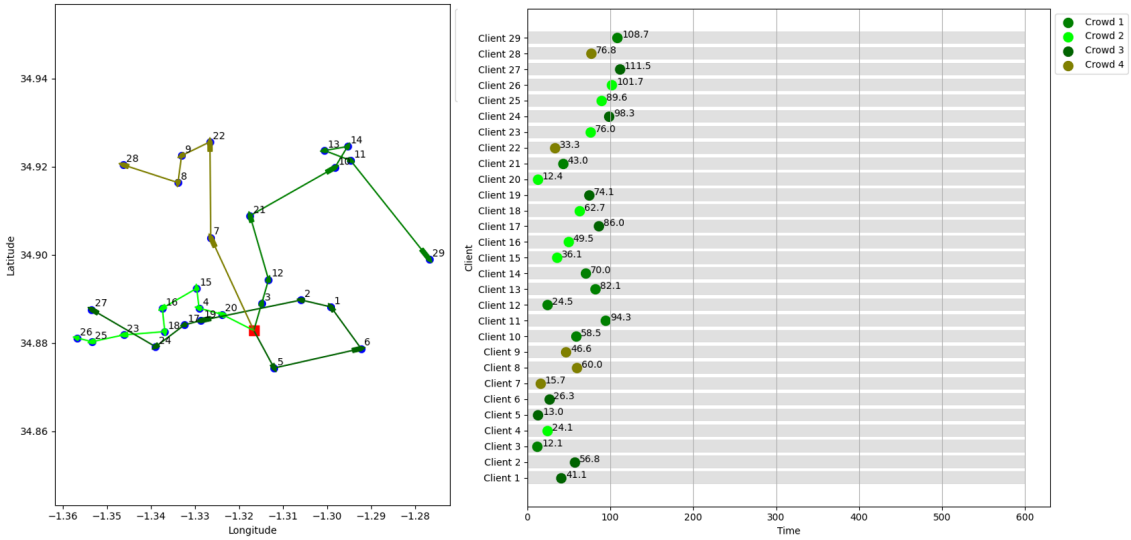


Figure 3.13: 30 Nodes with Light time window- Crowd Only

Under Light time windows (Figures 3.9, 3.11, and 3.13), service pressure decreases, and the difference between fleet strategies becomes less pronounced. Figure 3.13 (Crowd-only, Light Windows) still demonstrates earlier and more distributed deliveries compared to Figure 3.11 (Company-only, Light Windows), where several clients are served much later in the operational day .

The crowd-sourced approach proves particularly advantageous during high-demand periods, where flexible vehicle deployment ensures that peak-hour customer expectations are better met. By not requiring a return to the depot and by allowing more direct routing, crowd vehicles facilitate earlier customer service, especially under Hard operational constraints.

Conclusion

This chapter presented an approach to solve the Vehicle Routing Problem with Time Windows (VRPTW) using a heterogeneous fleet composed of company-owned and crowd-sourced vehicles. Through a detailed mathematical formulation and the application of both

exact (Gurobi) and metaheuristic (Simulated Annealing) solution methods, we aimed to minimize total delivery costs while satisfying service time windows and vehicle capacity constraints.

The hybrid fleet model enabled flexible vehicle allocation based on operational requirements and customer constraints. Company-owned vehicles were optimized for cost-effective routing under standard demand conditions, while crowdsourced vehicles provided valuable flexibility, especially during peak hours or when delivery constraints were Hard. Unlike traditional vehicles, crowd drivers were not required to return to the depot, enabling more direct delivery paths and faster service.

The simulation results confirmed that integrating crowdsourced vehicles can significantly enhance last-mile efficiency. In particular, crowdshipping proved to be advantageous in high-demand scenarios, helping to absorb overflow, reduce route lengths, and improve on-time delivery rates. This flexibility leads to better customer satisfaction and more resilient logistics operations.

Overall, the chapter demonstrates how combining optimization techniques with a mixed delivery strategy can lead to meaningful improvements in both cost and performance. The findings also set the stage for further exploration into how digital technologies, such as blockchain, can support traceability and coordination in such hybrid logistics systems.

Chapter 4

Blockchain Framework for the Last Mile delivery

Introduction

Blockchain technology, first introduced to the public through the cryptocurrency Bitcoin, has evolved into a transformative solution far beyond digital finance. Its decentralized, transparent, and tamper-proof nature makes it a powerful tool for rethinking how data, assets, and services are managed and exchanged. In recent years, blockchain has found increasing relevance in fields such as supply chain management, where it enables improved visibility, traceability, and trust across all stages of product flow from origin to final delivery.

Last-mile delivery is among the most demanding segments of modern logistics. This final leg of the delivery process is often the most inefficient, costly, and environmentally impactful. Traditional systems for last-mile operations are usually centralized, limiting transparency and introducing risks of manipulation, delay, or data loss. As delivery ecosystems evolve to include crowdsourced drivers and dynamic routing, the need for secure, decentralized coordination becomes even more critical.

This chapter presents the implementation of a blockchain-based solution using Solana blockchain, it supports a complete delivery lifecycle: from the registration of depots, vehicles, and drivers, to order creation, assignment, fulfillment, carbon tracking, and packaging return. Unlike conventional platforms, this approach ensures that all key operations are verifiable, decentralized, and resistant to tampering.

The proposed system enables secure token-based payments for the stakeholders and implements smart contracts written in Rust with Anchor along with incentive mechanisms. In addition, decentralized storage with IPFS is used to manage off-chain data such as route plans and delivery proofs. To validate the functionality of the system, the implementation includes an interactive TypeScript client (client.ts) and a test suite (anchor.test.ts), simulating real-world logistics scenarios with both company-owned and crowd-sourced fleets.

In this chapter, we will detail the development and testing of this smart contract-based platform, illustrating how blockchain technology can enhance transparency, sustainability, and operational trust in last-mile delivery services.

4.1 Overview of Solana Blockchain

Solana is a high-performance, open-source blockchain platform designed to support scalable, secure, and low-cost decentralized applications (dApps) and smart contracts. Unlike traditional blockchain systems that rely on energy-intensive mining, Solana employs a unique consensus mechanism that combines Proof of Stake (PoS) with Proof of History (PoH). This hybrid approach enables the network to achieve high throughput, processing up to 65,000 transactions per second (TPS), with exceptionally low latency and minimal transaction costs, typically averaging ~$0.00025 per transaction.

Solana's native cryptocurrency is *SOL*, which is used to pay for transaction fees and participate in staking operations. The smallest unit of *SOL* is called a *Lamport*, where 1 *SOL* = 1,000,000,000 *Lamports*, enabling small payments and fast application processes

Functioning as a decentralized ledger, Solana verifies blocks of transactions through a global network of validator nodes, ensuring transparency, immutability, and resistance to centralized control.

By supporting token creation, smart contract execution, and integration with decentralized storage systems such as IPFS, Solana provides a robust foundation for Web3 innovation. Its energy efficiency, very low transaction costs, and developer-friendly environment make it particularly well-suited for real-world applications such as last-mile delivery, logistics, decentralized finance, and digital marketplaces.

4.2 Solana Programs

In Solana, "smart contracts" are referred to as programs. These programs are deployed to specific on-chain accounts that store the compiled executable code. Users interact with programs by submitting transactions that contain instructions which define the operations the program should execute.

4.2.1 Key Characteristics

- A Solana program is stored in an account that contains executable code, and the program's functionality is divided into units of logic called instructions.
- Programs on Solana are stateless; however, they can execute instructions that modify or create other accounts to store persistent data.
- Programs can be updated by an upgrade authority. Once this authority is removed, the program becomes immutable and cannot be changed.
- On-chain program accounts can be verified to match their corresponding source code, ensuring transparency through verifiable builds.
- **Program Ownership:** In Solana's Account Model, each account is owned by a designated program. Only the owner program has the authority to:
  - Modify the account's data
  - Deduct lamports from the account's balance

4.2.2 Development of Solana Programs

Solana programs are typically written in the Rust programming language, with two common development approaches:

- **Anchor Framework:** Anchor is a development framework that simplifies writing Solana programs. It uses Rust macros to reduce repetitive code, making the development process faster and easier. This approach is recommended for beginners as it abstracts many of the complexities of working with Solana.
- **Native Rust:** This approach involves writing Solana programs directly in Rust, without the use of a framework. While it provides more flexibility and control, it also introduces greater complexity and requires a deeper understanding of the Solana runtime environment.

In this work, we chose the Anchor framework for developing the Solana program because of its simplicity and efficiency. Anchor simplifies the development process by reducing the complexity of working directly with Solana's blockchain. It provides useful features like automatic account management and a simplified program structure, making it a practical choice for quick development and testing.

Here is a basic template for an empty Solana program using the Anchor framework. It includes the necessary structure but doesn't define any functionality yet.

```
use anchor_lang::prelude::*;

// Define the program ID
declare_id!("PROGRAM_ID Here");
#[program]
pub mod Solana_Program {
    use super::*;
    // Program functions goes here
}
#[derive(Accounts)]
pub struct Account<'info> {
    // Definition of the accounts structure
}
#[account]
pub struct Struct {
    // structs
}
```

Listing 4.1: Solana Program Structure using Anchor(lib.rs)

- `use anchor_lang::prelude::*;`: This statement imports essential components from the Anchor framework, such as macros and types, enabling the use of features like `declare_id!`, `#[program]`, and `[derive(Accounts)]`.
- `declare_id!`: This macro defines the program's on-chain address (program ID), which is important for interactions with the Solana blockchain.
- `#[program]`: This attribute macro designates the module containing the program's instruction handlers, where the core logic of the program is implemented.
- `#[derive(Accounts)]`: This macro is applied to structs to define the accounts required by each function, facilitating automatic validation and deserialization of account data.
- `#[account]`: This attribute macro is used to define custom account types, specifying the structure of data stored in accounts created by the program.

4.3 Tokens on Solana

Tokens are digital assets that represent ownership over diverse categories of assets, including currencies, rights, or utilities. Tokenization facilitates the encoding of property rights into a digital form, enabling secure and transparent transactions on blockchain networks. On the Solana blockchain, tokens are implemented through the Solana Program Library (SPL) and are known as SPL Tokens. The SPL Token Program provides the logic for managing both fungible (interchangeable) and non-fungible (unique) tokens.

- Fungible tokens are divisible and interchangeable assets, such as stablecoins like USDC, where each unit holds identical value.
- Non-fungible tokens (NFTs) represent unique, indivisible digital assets—such as artwork or certificates—each with distinct characteristics and ownership.

A Mint Account defines a particular token type and maintains global metadata such as its total supply and the authority permitted to issue new tokens. A Token Account is linked to a specific Mint Account and records the balance of a given token for a specific owner. For simplicity and standardization, each wallet can have an Associated Token Account, which is a uniquely derived Token Account based on the wallet and mint addresses, ensuring one-to-one mapping for token ownership.

4.4 Offchain and Onchain storage

On-chain storage involves persisting data directly within Solana's blockchain using accounts. Each account can store arbitrary data and is managed by the network's consensus mechanism, ensuring immutability and transparency. However, this method incurs costs proportional to the data size and is best suited for critical information like token balances, smart contract states, and essential metadata.

Off-chain storage refers to keeping data outside the Solana blockchain, typically on decentralized storage networks like IPFS or Arweave. In this setup, only a reference or hash of the data is stored on-chain, allowing for verification without the need to store large files directly on the blockchain. This approach is cost-effective and scalable, making it ideal for storing sizable assets such as images, videos, results of an optimization model or extensive metadata associated with NFTs.

4.5 Framework Overview and assumption

Figure 4.1 provides an overview of the proposed blockchain-based framework for last-mile delivery. This approach uses the Anchor framework to develop the application logic, while Solana serves as the underlying blockchain infrastructure. Non-persistent data, including delivery metadata and package tracking information, is shared among stakeholders through the InterPlanetary File System (IPFS), ensuring efficient off-chain storage. The framework is organized into several key modules, including order lifecycle management, vehicle routing optimization, and sustainability features.

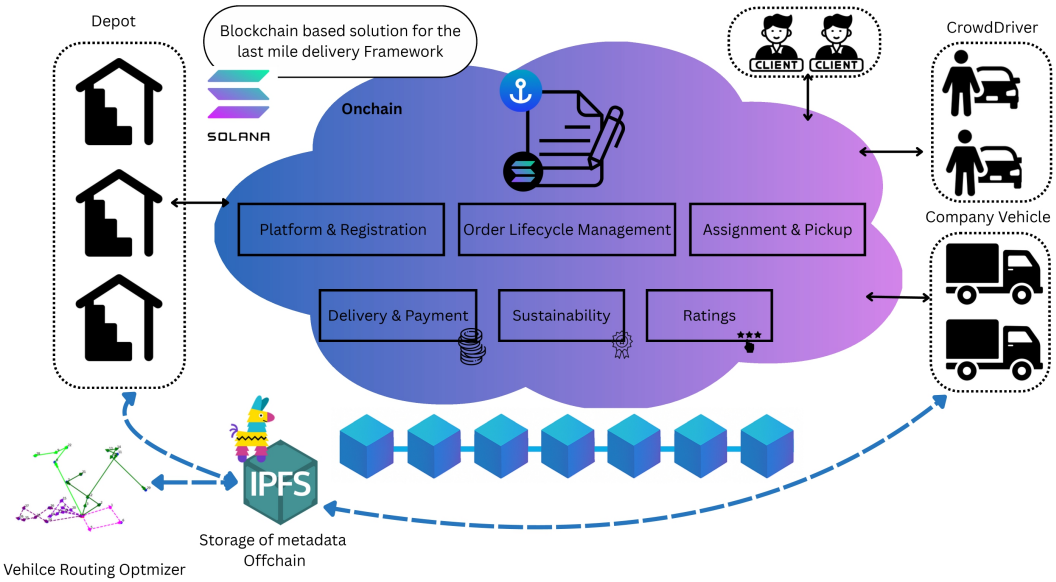


Figure 4.1: Framework of the Blockchain-based Last Mile Delivery

The roles of each participant in the system are as follows:

- **Depot:** Acts as hub where vehicles are registered. The depots manage the vehicle fleet, ensuring vehicle availability and capacity management. Each depot interacts with the platform to update the vehicles' availability and metadata.
- **Platform:** The platform is the core of the framework, managing key operations such as order registration, vehicle assignment, payment processing, sustainability features, and rating systems. The platform enables users to engage with the blockchain and ensures all orders and delivery processes are securely recorded on-chain.
- **Company Vehicles:** These are the primary vehicles used for deliveries. Company vehicles register on the platform and are assigned delivery tasks based on their availability, capacity, and environmental data. They also interact with the vehicle routing optimizer to find the most efficient delivery routes.
- **Crowd Drivers:** These are external, crowd-sourced drivers who register on the platform and take up available delivery tasks when company vehicles are unavailable or additional capacity is needed. Crowd drivers follow the same process as company vehicles to accept and complete orders, and can also interact with the vehicle routing optimizer to find the optimal route.
- **Clients:** Clients initiate orders through the platform, providing the necessary details such as destination and package size. Once the order is confirmed and completed, clients also provide ratings for the delivery service.
- **IPFS/Pinata:** Decentralized storage is used to store off-chain metadata such as vehicle profiles, optimized routes, and delivery status. IPFS (InterPlanetary File System) provides a reliable, secure, and scalable solution for sharing data among the participants.

4.6 Functions and Data Structures

This section presents the core data structures and functions that underpin the logic of the decentralized last-mile delivery platform implemented on Solana. The system is composed of account-based structures representing key entities—such as depots, vehicles, drivers, and orders—as well as enumerated types, error codes, and helper functions. Each structure is carefully designed to ensure traceability, decentralization, and alignment with the platform's functional requirements. The smart contract functions manage essential delivery processes, including order creation, assignment, pickup, delivery, returns, carbon offset calculation, and reward distribution.

Table 4.1 defines the core data types used in the smart contract, including Rust primitives like `u64`, `bool`, and `Option<T>`, as well as Solana-specific terms such as `Pubkey` and SPL tokens.

Table 4.1: Glossary of Terms Used in Smart Contract Implementation

Term	**Definition**
`Pubkey`	A public key representing an address on Solana. Used to identify users, accounts, or programs on the blockchain.
`u64`	Unsigned 64-bit integer. Used for IDs and counters such as order_id or driver_count.
`u32`	Unsigned 32-bit integer. Used for capacities, counts, and total number of ratings.
`u8`	Unsigned 8-bit integer. Used for small numbers such as rating values between 0 and 5.
`bool`	Boolean type indicating true or false. Used for flags like `is_available` or `active`.
`f64`	64-bit floating-point number. Used for decimals such as coordinates, CO_2 emissions, and delivery costs.
`String`	A text string (UTF-8 encoded). Used for names, addresses, vehicle types, and URLs.
`Option<T>`	Represents an optional value. It can be `Some(T)` if present, or `None` if absent. Used for fields like metadata URLs or return reasons.
`i64`	Signed 64-bit integer. Commonly used for timestamps in Unix format (seconds since epoch).
`SPL Token`	A fungible token standard on Solana, similar to ERC-20. Used for payments and rewards in the platform.

Table 4.2 summarizes the primary data structures used in the system. These include components like the delivery platform, depots, vehicles, drivers, orders, and packaging return mechanisms that participate in our last mile delivery system.

Table 4.2: Main Structures

Structure Name	**Description**
DeliveryPlatform	Core structure representing the delivery platform itself
Depot	Represents physical storage/distribution facilities
Vehicle	Represents company-owned delivery vehicles
Driver	Represents crowd-sourced delivery drivers
Order	Represents delivery orders in the system
PackagingReturn	Tracks returned packaging for recycling incentives

Table 4.3 presents the enumeration types used in the contract. These fixed-value sets, such as `OrderStatus` and `FuelType`, standardize the tracking of order lifecycles and emission-related attributes across the platform.

Table 4.3: Enumeration Types

Enum Name	**Values**
OrderStatus	Created, AssignedToVehicle, AssignedToDriver, InTransit, Delivered, Returned, Cancelled, ReturnScheduled, ReturnPickedUp
FuelType	Diesel, UnleadedGasoline (Used for CO2 emissions calculations)

Table 4.4 lists the custom error types defined in the contract. These errors enforce business rules, such as depot capacity limits, vehicle availability, or invalid actions based on status or actor type, ensuring logical and secure execution.

Table 4.4: Error Codes

Error Name	**Description**
Unauthorized	Access denied for the requested operation
DepotCapacityExceeded	The depot has reached its maximum capacity
InvalidOrderStatus	Order status does not permit the requested action
NotCompanyVehicle	The vehicle is not company-owned
VehicleNotAvailable	The vehicle is currently unavailable
VehicleNotAtDepot	Vehicle is not at the correct depot location
NotCrowdSourced	The driver is not registered as crowd-sourced
DriverNotAvailable	The driver is currently unavailable
InvalidRating	Rating value outside valid range (0-5)
PackagingAlreadyExists	Packaging has already been returned for this order

Table 4.5 outlines the structure responsible for managing global counters, administrator identity, and system activation. It serves as the root of the smart contract, linking all other components and tracking the platform's operational status.

Table 4.5: DeliveryPlatform Structure

Field	**Type**	**Description**
admin	Pubkey	Administrator wallet address
platform_name	String	Name of the delivery platform
order_count	u64	Total number of orders in the system
driver_count	u64	Total registered drivers
vehicle_count	u64	Total registered vehicles
depot_count	u64	Total registered depots
returned_package_count	u64	Count of returned packages
active	bool	Platform activation status

Table 4.6 details the structure representing physical storage depots, including location, capacity, and identification information. Depots are essential for organizing package distribution and enabling a good coordination of delivery routes.

Table 4.6: Depot Structure

Field	**Type**	**Description**
platform	Pubkey	Reference to parent platform
manager	Pubkey	Depot manager wallet address
name	String	Name of the depot
location_lat	f64	Latitude coordinate
location_lng	f64	Longitude coordinate
capacity	u32	Maximum package capacity
available_capacity	u32	Current available capacity
active	bool	Depot activation status
depot_id	u64	Unique identifier for the depot

Table 4.7 describes the structure of delivery vehicles, including ownership status, availability, emission factors, and optional metadata. It supports both company and sustainability-focused logistics functionalities within the platform.

Table 4.7: Vehicle Structure

Field	Type	Description
platform	Pubkey	Reference to parent platform
owner	Pubkey	Vehicle owner's wallet address
vehicle_type	String	Type/model of the vehicle
capacity	u32	Package carrying capacity
license_plate	String	Vehicle registration identifier
is_company_owned	bool	Whether vehicle belongs to the company
is_available	bool	Current availability status
current_depot	Pubkey	Current depot location
vehicle_id	u64	Unique identifier for the vehicle
rating	u8	Average customer rating (0-5) - initialized to 0
total_ratings	u32	Total number of ratings received - initialized to 0
fuel_type	FuelType	Type of fuel used by vehicle
co2_per_km	f64	CO2 emissions per kilometer
metadata_url	Option < String >	Optional URL for additional metadata - initially None

Table 4.8 presents the structure for crowd-sourced drivers, capturing personal, operational, and environmental data. It includes ratings, availability, vehicle attributes, and supports decentralized task assignment in last-mile delivery.

Table 4.8: Driver Structure

Field	Type	Description
platform	Pubkey	Reference to parent platform
driver_wallet	Pubkey	Driver's wallet address
name	String	Driver's name
vehicle_type	String	Type of vehicle used by driver
capacity	u32	Package carrying capacity
license_plate	String	Vehicle registration identifier
is_crowd_sourced	bool	Whether driver is crowd-sourced
is_available	bool	Current availability status
rating	u8	Average customer rating (0-5) - initialized to 0
total_ratings	u32	Total number of ratings received - initialized to 0
total_deliveries	u32	Total completed deliveries - initialized to 0
driver_id	u64	Unique identifier for the driver
fuel_type	FuelType	Type of fuel used by driver's vehicle
co2_per_km	f64	CO2 emissions per kilometer
metadata_url	Option<String>	Optional URL for additional metadata - initially None

Table 4.9 presents all necessary information for managing delivery orders. It includes client data, destination coordinates, package size, timestamps, emissions, and carbon offset costs, ensuring full traceability throughout the delivery lifecycle.

Table 4.9: Order Structure

Field	**Type**	**Description**
platform	Pubkey	Reference to parent platform
client	Pubkey	Client's wallet address
client_name	String	Name of the client
delivery_address	String	Delivery destination address
location_lat	f64	Destination latitude coordinate
location_lng	f64	Destination longitude coordinate
package_size	u8	Size classification of the package
urgent	bool	Whether delivery is urgent
status	OrderStatus	Current status in the delivery lifecycle
delivery_cost	f64	Cost of delivery service
created_at	i64	Timestamp of order creation
assigned_at	i64	Timestamp of driver/vehicle assignment - initialized to 0
picked_up_at	i64	Timestamp of package pickup - initialized to 0
delivered_at	i64	Timestamp of successful delivery - initialized to 0
depot	Pubkey	Source depot for the package
driver	Option<Pubkey>	Assigned driver (if applicable) - initially None
vehicle	Option<Pubkey>	Assigned vehicle (if applicable) - initially None
order_id	u64	Unique identifier for the order
return_reason	Option<String>	Reason for return (if applicable) - initially None
co2_emitted	Option<f64>	Calculated CO2 emissions for delivery (ademe formula)- initially None
offset_cost	Option<f64>	Cost to offset carbon emissions - initially None
delivery_metadata_url	String	URL for delivery route optimization data

Table 4.10 defines the structure used to log packaging returns as part of circular economy integration. It tracks return events, associates them with orders, and optionally links to proof via an external URL.

Table 4.10: PackagingReturn Structure

Field	**Type**	**Description**
order	Pubkey	Reference to related order
client	Pubkey	Client's wallet address
returned_at	i64	Timestamp of packaging return
proof_url	Option<String>	Optional URL with proof of return - may be None

Table 4.11 introduces the initialization and registration functions for creating the platform and enrolling its key entities. These operations form the foundation for all our decentralized logistics processes.

Table 4.11: Function Descriptions – Initialization and Registration

Function Name	**Description**
`initialize`	Initializes the delivery platform with an admin, name, and zero counters
`register_depot`	Registers a new depot with location and capacity information
`register_company_vehicle`	Registers a company-owned vehicle with sustainability metrics
`register_crowd_driver`	Registers a crowd-sourced driver with vehicle and sustainability information

Table 4.12 contains the order lifecycle management functions of the system. From order creation to assignment, pickup, and delivery, these functions automate and secure key steps in the decentralized delivery.

Table 4.12: Function Descriptions – Order Lifecycle Management

Function Name	**Description**
`create_order`	Creates a new delivery order with location, size, and urgency information. Delivery cost is calculated based on package size and urgency.
`accept_order_as_driver`	Allows crowd-sourced driver to accept an order. Sets `order.driver = Some(driver.key())` and updates status.
`accept_order_as_vehicle`	Assigns company vehicle to an order. Sets `order.vehicle = Some(vehicle.key())` and updates status.
`pick_up_order_driver`	Records package pickup by a driver. Updates order status to `InTransit`.
`pick_up_order_vehicle`	Records package pickup by a company vehicle. Updates order status to `InTransit`.
`deliver_order_driver`	Completes delivery by driver, processes payment using SPL tokens, and calculates carbon offset.
`deliver_order_vehicle`	Completes delivery by company vehicle, processes payment using SPL tokens, and calculates carbon offset.
`cancel_order`	Cancels a pending order by client. Only allowed for orders in `Created` status.
`see_optimized_route`	Attaches route optimization metadata URL to an order for efficient delivery routing.

Table 4.13 includes the functions that enable clients to evaluate drivers and vehicles. These ratings are stored on-chain and used to compute average scores, supporting transparency and service quality monitoring.

Table 4.13: Function Descriptions – Rating System

Function Name	**Description**
`rate_driver`	Records customer rating for a driver (0–5). Calculates new average rating.
`rate_vehicle`	Records customer rating for a vehicle (0–5). Calculates new average rating.

Table 4.14 outlines return-related functions that support reverse logistics. These functions promote packaging return practices and validate sustainability practices through reward mechanisms and pickup confirmations.

Table 4.14: Function Descriptions – Return and Rewards

Function Name	Description
`return_package`	Marks a delivered package as returned with reason for return. Updates platform returned package count.
`return_packaging`	Records return of packaging materials for recycling. Proof URL parameter is optional.
`reward_for_packaging_return`	Issues SPL tokens as reward for packaging return based on package size.
`schedule_packaging_return_pickup`	Schedules pickup for packaging materials. Proof URL parameter is optional.
`return_pickup_completed`	Marks packaging pickup as completed. Changes order status to `ReturnPickedUp`.

Table 4.15 presents the functions used to associate additional metadata with drivers and vehicles. These links (pointing to IPFS/Pinata) enrich the data model with off-chain details like documentation or optimized routing.

Table 4.15: Function Descriptions – Metadata Assignment

Function Name	Description
`set_vehicle_metadata_url`	Sets URL with additional vehicle metadata. Updates `vehicle.metadata_url` from `None` to `Some(metadata_url)`.
`set_driver_metadata_url`	Sets URL with additional driver metadata. Updates `driver.metadata_url` from `None` to `Some(metadata_url)`.

Table 4.16 details the carbon offset calculation function. It uses a fixed coefficient (0.02 currency units per kg CO_2) based on ADEME guidelines to compute the financial cost of compensating delivery emissions.

Table 4.16: Function Description – Carbon Offset Calculation

Function Name	Description
`adem_offset`	Calculates the monetary cost to offset CO_2 emissions using the formula: $\texttt{order.offset\_cost} = 0.02 \times (\texttt{distance\_km} \times \texttt{driver.co2\_per\_km})$ This applies a fixed rate of 0.02 currency unit per kilogram of CO_2 emitted, based on ADEME's estimation of the average cost of carbon offsetting.

4.7 System Interactions

To provide a good understanding of the operational workflow, the complete system interaction diagram illustrates the sequential flow of functions executed among key actors:

`Admin`, `Client`, `Platform`, `Depot`, `CompanyVehicle`, `CrowdDriver`, and `IPFS/Pinata`. This end-to-end diagram discribe the core lifecycle of last-mile delivery, from platform initialization, depot and entity registration, to order assignment, delivery tracking, carbon offset computation, packaging return, and reward incentives. Due to its extensive length, the sequence has been divided into three parts for clarity in the manuscript. For an uninterrupted view of the full process, readers can scan the QR code in Figure 4.2 or access the diagram directly via this IPFS link: `https://gateway.pinata.cloud/ipfs/bafybeigyc3l2txhi5qj3yg52pbqlugch5zjkvghz4uv2g6n77ux5rkc2n4`.

Figure 4.2: QR Code - Full System Sequence Diagram (IPFS using Pinata)

Figure 4.3 illustrates the initial setup phase of the decentralized delivery platform. The Admin initiates the system by calling `initialize()`, which creates the `DeliveryPlatform` account. Next, depots are registered via `register_depot()`, which triggers depot creation and storage on-chain. The Admin continues by registering company-owned vehicles using `register_company_vehicle()`, specifying essential operational and environmental attributes such as fuel type and CO_2 emissions. Finally, crowd-sourced drivers are onboarded via `register_crowd_driver()`, enabling decentralized workforce participation. These setup interactions ensure the foundational entities are securely registered on-chain.

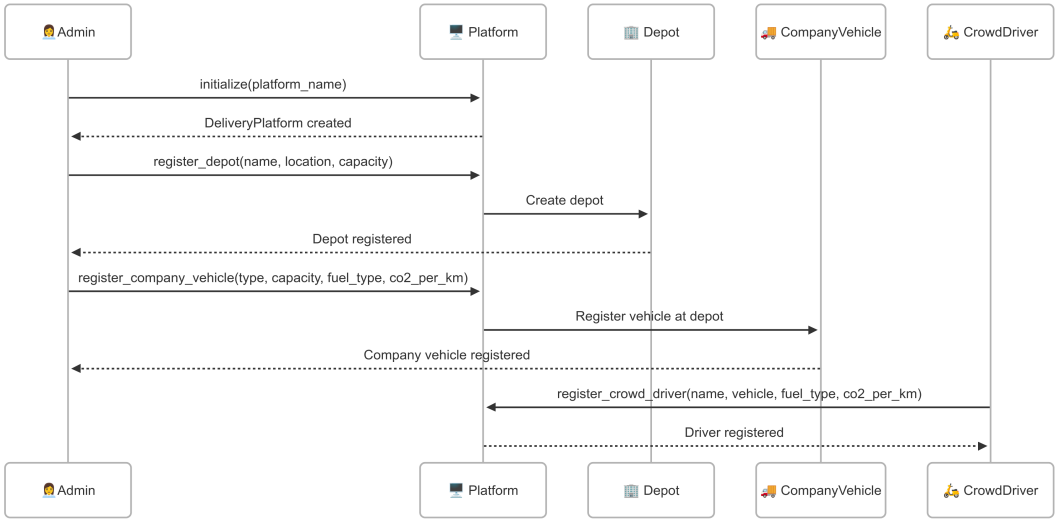


Figure 4.3: System interaction part 1

Figure 4.4 captures the order management and metadata assignment process. A client creates a delivery order by specifying location, size, and urgency, which triggers cost

estimation and depot capacity reservation. The platform then supports dynamic assignment: company vehicles and crowd-sourced drivers check availability and accept orders autonomously. Simultaneously, metadata is uploaded to IPFS via Admin or Driver and linked to on-chain accounts through `set_vehicle_metadata_url()` and `set_driver_metadata_url()`. Admins can also upload optimized delivery routes stored on IPFS and attached via `see_optimized_route()`, enhancing transparency and data extensibility.

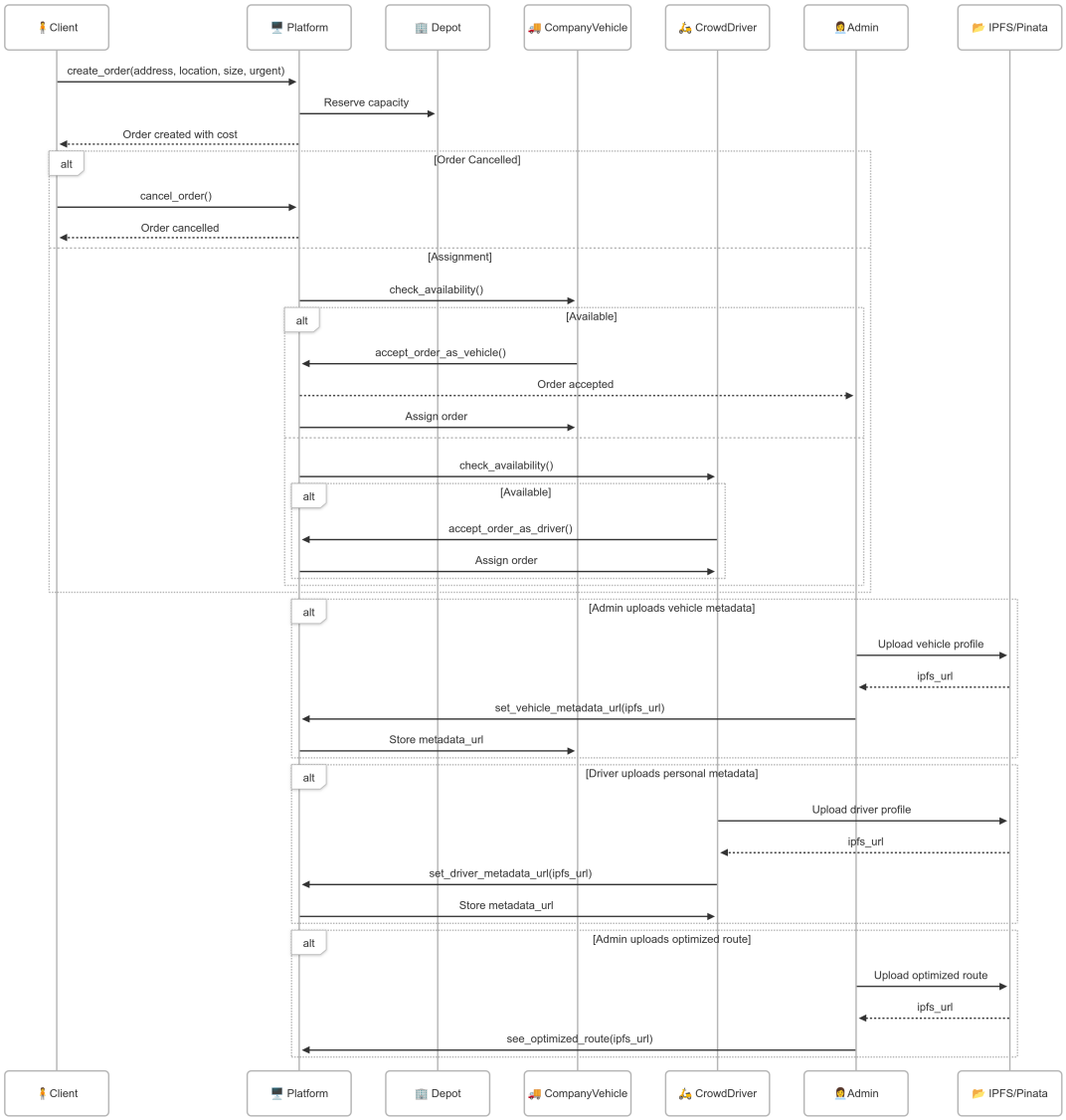


Figure 4.4: System interaction part 2

Figure 4.5 presents the core delivery execution phase, including package pickup, CO_2 tracking, payment, and return handling. Drivers or vehicles mark orders as picked up and in transit. Upon delivery, SPL token-based payments are transferred, and CO_2 emissions are calculated using ADEME's offset model. Clients can rate drivers and vehicles, with scores recorded on-chain. In the event of returns, clients submit reasons and trigger packaging return workflows. Returned packaging is verified and rewarded through minting SPL tokens. Scheduled pickups of packaging are also supported, closing the loop and enabling circular economy integration.

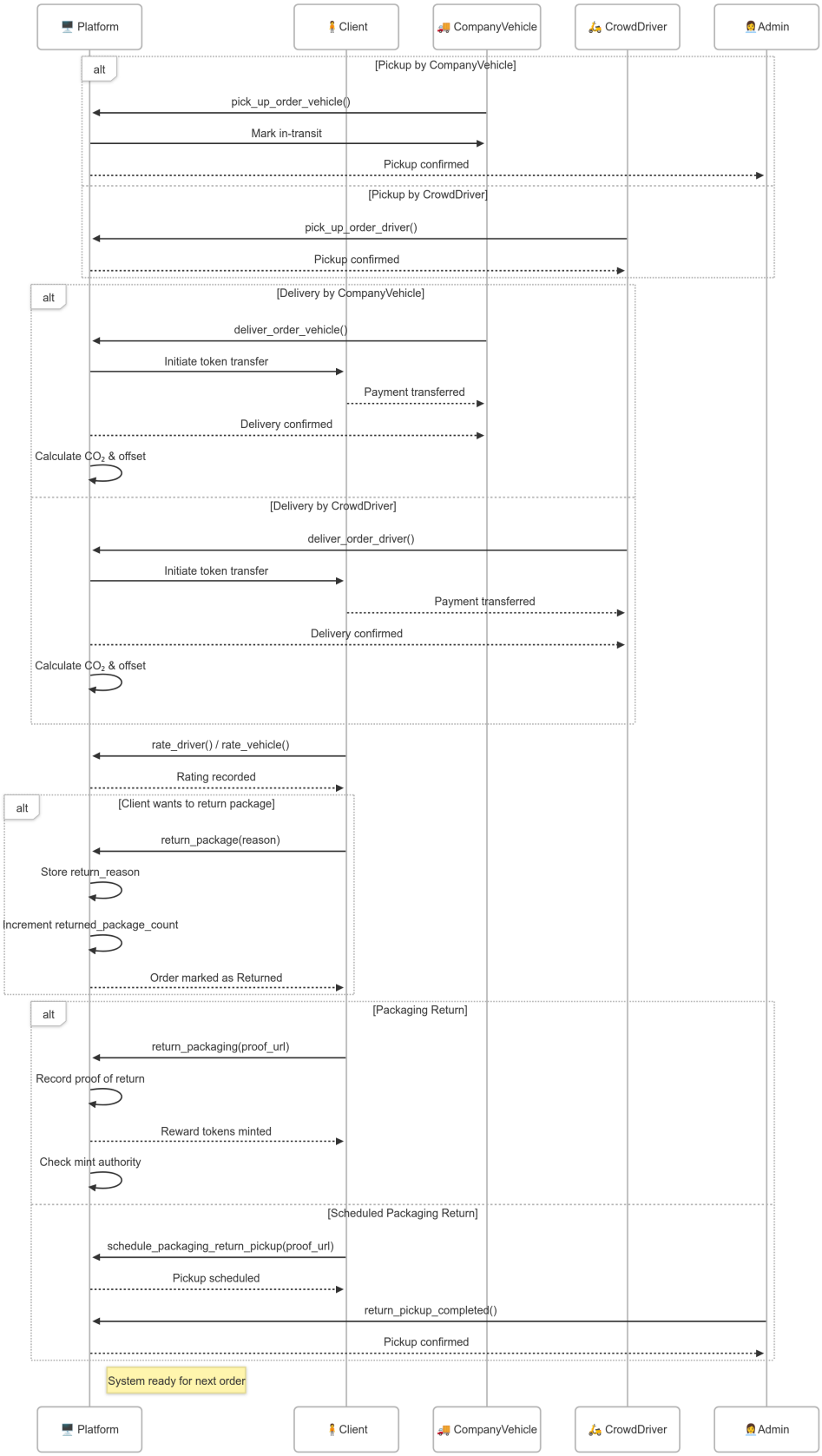


Figure 4.5: System interaction part 3

4.8 System Configuration and Operating System

The project is performed on a 2.30 GHz Intel Core i7-12700H CPU with 16 GB of RAM. We implement the project using Windows 11.

4.8.1 Solana playground

Solana Playground 4.6 is a browser-based Integrated Development Environment (IDE) for writing, testing, and deploying Solana smart contracts using the Rust-based Anchor framework. It enables developers to interact with Solana programs directly in the browser without requiring local setup. The platform also supports scripting through `client.ts`, a TypeScript-based client file that simulates real-world interactions with the smart contract, such as initializing accounts, submitting transactions, or reading on-chain data. Additionally, Solana Playground provides an environment to implement and execute test suites (`anchor.test.ts`) using TypeScript and the Mocha testing framework. These tests are essential for verifying the correctness of program logic and ensuring contract reliability before deployment. We use Solana Playground (`https://beta.solpg.io/`) to build, test, and interact with our smart contracts for last-mile delivery system on the Solana blockchain.

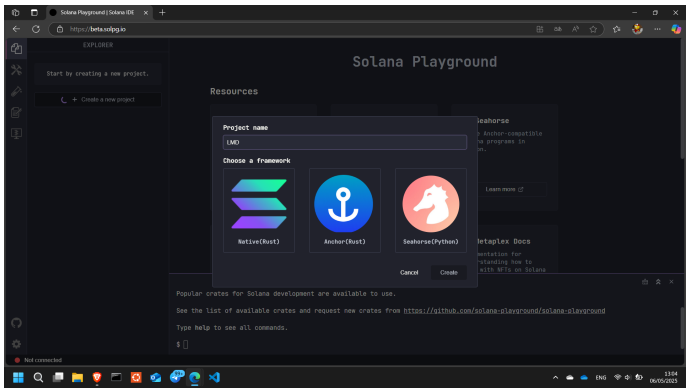


Figure 4.6: Solana Playground

4.8.2 NodeJS and Mocha

Node.js 4.7 is a runtime environment built on the V8 JavaScript engine that enables the execution of JavaScript code outside of a web browser. It is widely used for developing scalable, server-side applications and supports asynchronous, event-driven programming, making it ideal for blockchain testing and scripting with frameworks like Mocha and TypeScript.

Figure 4.7: Logo of Mocha and Node.js

Mocha is a JavaScript-based testing framework designed to run asynchronous tests on Node.js applications. It provides a structured environment for writing and executing unit tests, making it a popular choice for validating smart contract behavior in TypeScript environments such as Solana Playground.

4.8.3 Phantom wallet

Phantom 4.8 is a crypto wallet designed for managing digital assets and interacting with decentralized applications (dApps) on the Solana blockchain. Available as a browser extension and mobile application, it provides a user-friendly interface and injects the Solana API into the web context, enabling applications such as Solana Playground to request transactions and communicate with smart contracts. Phantom allows users to create and manage Solana accounts, securely sign transactions, and store or transfer SOL and SPL tokens. We chose Phantom for its seamless integration with Solana-based programs and its support for on-chain interactions such as contract deployment, execution, and testing.

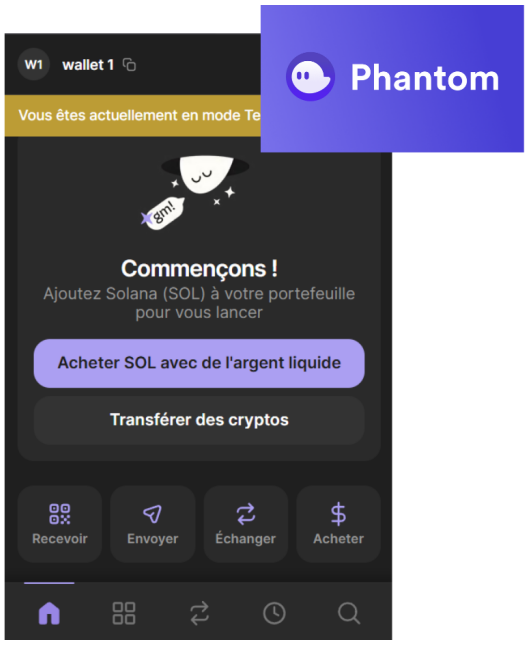


Figure 4.8: Phantom Wallet

4.8.4 Python SDK for Solana

The Solana Python SDK 4.9, known as `solana-py`, is an open-source library that enables developers to interact with the Solana blockchain using Python. It provides functionalities for building transactions, querying blockchain data, and managing SPL tokens. This SDK is particularly useful for integrating Solana smart contracts with Python-based applications, such as data analysis tools or optimization algorithms. In our project, we utilized `solana-py` to connect our Python optimization of the VRP in Jupyter Notebook with on-chain programs deployed via Solana Playground.

Figure 4.9: Solana.py

In order to enable interaction with the Solana program from a Python environment, we installed the required dependencies using Jupyter in Visual Studio code as follows:

Install Anchor and Dependencies for Python environment

```
>> !pip install solana
>> !pip install anchorpy
>> !pip install solders
```

4.8.5 Solana Explorer

Solana Explorer used to monitor on-chain transactions, inspect account states, and verify the execution of smart contracts deployed through Solana Playground. It provides a transparent interface for visualizing program interactions and validating blockchain activity during testing and deployment phases (`https://explorer.solana.com`).

4.9 Implementation

This section presents the implementation of the system and the essential steps required to set up the last-mile delivery system, enabling seamless interaction with the Solana blockchain through smart contract integration.

4.9.1 Environment configuration

This section outlines the configuration steps necessary to set up the development environment for our blockchain-based last-mile delivery system using Solana.

1. **Create a New Workspace:** By Clicking on `+ New Workspace`, selecting the `Anchor (Rust)` template. This template initializes the project structure with essential files, including `client.ts` for front-end interaction and `anchor.test.ts` for writing and executing test cases.

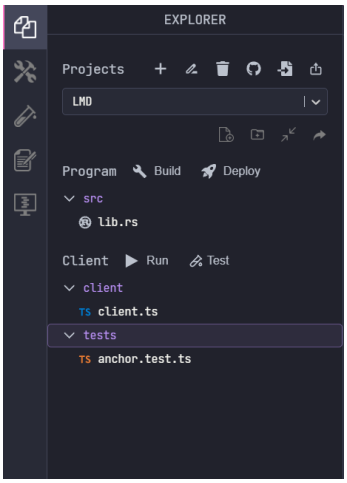


Figure 4.10: Solana workplace

2. **Program ID:** Initially, the `declare_id!()` macro in `lib.rs` is set to a default placeholder value `"11111111111111111111111111111111"`. When the program is built, Solana Playground automatically generates a unique program ID and replaces the placeholder with the actual on-chain address of the deployed program.

Figure 4.11: Program id by default

3. **Write Program Logic:** Implement smart contract logic inside `lib.rs`, defining the main structures and functions such as platform initialization, order lifecycle management, and carbon tracking.
4. **Connect Wallet:** Before proceeding with deployment or interaction, connect your Solana wallet by clicking on the wallet icon in the top-right corner of Solana Playground. This connects the workspace to a wallet on the `devnet`, allowing you to sign and authorize transactions. If no wallet is connected, deployment and testing will fail.

Figure 4.12: Connecting the wallet

5. **Build and Deploy the Program:** To compile the smart contract, we run the `$build` command in the Solana Playground terminal. This command compiles the code and prepares the program bytecode for deployment. Upon successful compilation, the `declare_id!()` macro is automatically updated with the generated on-chain address. To deploy the program to the network (typically `devnet`), run the `$deploy` command. Note that deploying a program requires allocating SOL to the executable program account. To obtain devnet SOL, use the `solana airdrop 5` command in the terminal or access the Solana Web Faucet.

Build and Deploy the Program

```
$ solana airdrop 5
$ build
$ deploy
```

6. **Explorer Access:** After deployment, copy the program ID and paste it into `https://explorer.solana.com/` to monitor transactions and program states. Be sure to select the appropriate cluster, in our case we choosed `devnet`.

4.10 Last-Mile delivery Smart contract

The core logic of our decentralized last-mile delivery platform is implemented as a Solana program using the Anchor framework. The program contains modular functions for platform initialization, order processing, sustainability integration, and metadata handling.

Only the key functions within the `#[program]` module are shown in this section. Context structs and data accounts are summarized in dedicated tables for readability previsously in this chapter.

Before defining the program logic, we have to import in our program in `lib.rs` essential modules from the Anchor framework, the SPL Token library, and Solana's core types. The macro `declare_id!()` is used to register the program ID, which is automatically updated upon building in Solana Playground 4.13.

Figure 4.13: Import in the programs and Program ID

4.10.1 Platform Initialization and Registration

Function: initialize

Initializes the delivery platform account, sets the admin, and resets all counters. This is the entry point for creating a platform instance.

```
pub fn initialize(ctx: Context<Initialize>, platform_name:
    String) -> Result<()> {
     let platform = &mut ctx.accounts.platform;
     platform.admin = ctx.accounts.admin.key();
     platform.platform_name = platform_name;
     platform.order_count = 0;
     platform.driver_count = 0;
     platform.vehicle_count = 0;
     platform.depot_count = 0;
     platform.returned_package_count = 0;
     platform.active = true;
     Ok(())
```

```
}
```

Listing 4.2: Initialize the delivery platform

Function: register_depot

Registers a new depot with geolocation and capacity under the supervison of the platform admin. Each depot is assigned a unique ID.

```
pub fn register_depot(
    ctx: Context<RegisterDepot>,
    name: String,
    location_lat: f64,
    location_lng: f64,
    capacity: u32,
) -> Result<()> {
    require!(
        ctx.accounts.platform.admin == ctx.accounts.admin.
            key(),
        CustomError::Unauthorized
    );

    let platform = &mut ctx.accounts.platform;
    let depot = &mut ctx.accounts.depot;

    depot.platform = platform.key();
    depot.manager = ctx.accounts.admin.key();
    depot.name = name;
    depot.location_lat = location_lat;
    depot.location_lng = location_lng;
    depot.capacity = capacity;
    depot.available_capacity = capacity;
    depot.active = true;
    depot.depot_id = platform.depot_count;

    platform.depot_count = platform
        .depot_count
        .checked_add(1)
        .ok_or(CustomError::OverflowError)?;
    Ok(())
}
```

Listing 4.3: Register a new depot

Function: register_company_vehicle

The `register_company_vehicle` function registers company-owned vehicles, linking them to a depot while specifying metadata such as fuel type and CO_2 emissions.

```
pub fn register_company_vehicle(
    ctx: Context<RegisterVehicle>,
    vehicle_type: String,
```

```
    capacity: u32,
    license_plate: String,
    fuel_type: FuelType,
    co2_per_km: f64,
) -> Result<()> {
    require!(
        ctx.accounts.platform.admin == ctx.accounts.admin.
           key(),
        CustomError::Unauthorized
    );

    let platform = &mut ctx.accounts.platform;
    let vehicle = &mut ctx.accounts.vehicle;

    vehicle.platform = platform.key();
    vehicle.owner = ctx.accounts.admin.key();
    vehicle.vehicle_type = vehicle_type;
    vehicle.capacity = capacity;
    vehicle.license_plate = license_plate;
    vehicle.is_company_owned = true;
    vehicle.is_available = true;
    vehicle.return_to_depot_needed = false;
    vehicle.current_depot = ctx.accounts.depot.key();
    vehicle.vehicle_id = platform.vehicle_count;
    vehicle.fuel_type = fuel_type;
    vehicle.co2_per_km = co2_per_km;

    platform.vehicle_count = platform
        .vehicle_count
        .checked_add(1)
        .ok_or(CustomError::OverflowError)?;
    Ok(())
}
```

Listing 4.4: Register a company-owned delivery vehicle

Function: register_crowd_driver

Finally, the `register_crowd_driver` function adds independent, crowd-sourced drivers to the system with their own vehicle, availability status, and emission profile.

```
pub fn register_crowd_driver(
    ctx: Context<RegisterDriver>,
    name: String,
    vehicle_type: String,
    capacity: u32,
    license_plate: String,
    fuel_type: FuelType,
    co2_per_km: f64,
) -> Result<()> {
    let platform = &mut ctx.accounts.platform;
    let driver = &mut ctx.accounts.driver;
```

```
12
13      driver.platform = platform.key();
14      driver.driver_wallet = ctx.accounts.driver_wallet.key();
15      driver.name = name;
16      driver.vehicle_type = vehicle_type;
17      driver.capacity = capacity;
18      driver.license_plate = license_plate;
19      driver.is_crowd_sourced = true;
20      driver.is_available = true;
21      driver.rating = 0;
22      driver.total_ratings = 0;
23      driver.total_deliveries = 0;
24      driver.driver_id = platform.driver_count;
25      driver.fuel_type = fuel_type;
26      driver.co2_per_km = co2_per_km;
27
28      platform.driver_count = platform
29          .driver_count
30          .checked_add(1)
31          .ok_or(CustomError::OverflowError)?;
32      Ok(())
33  }
```

Listing 4.5: Register a crowd-sourced delivery driver

4.10.2 Order Lifecycle Management

Function: create_order

The `create_order` function enables clients to submit delivery orders, specifying package size, urgency, address, and geolocation. It calculates the cost of delivery and stores the order with the appropriate depot.

```
1   pub fn create_order(
2       ctx: Context<CreateOrder>,
3       client_name: String,
4       delivery_address: String,
5       location_lat: f64,
6       location_lng: f64,
7       package_size: u8,
8       urgent: bool,
9   ) -> Result<()> {
10      let platform = &mut ctx.accounts.platform;
11      let depot = &mut ctx.accounts.depot;
12      let order = &mut ctx.accounts.order;
13
14      let base_cost = 0.1 * package_size as f64;
15      let urgency_multiplier = if urgent { 1.5 } else { 1.0 };
16      let delivery_cost = base_cost * urgency_multiplier;
17
18      require!(
19          depot.available_capacity > 0,
```

```
            CustomError::DepotCapacityExceeded
        );

        order.platform = platform.key();
        order.client = ctx.accounts.client.key();
        order.client_name = client_name;
        order.delivery_address = delivery_address;
        order.location_lat = location_lat;
        order.location_lng = location_lng;
        order.package_size = package_size;
        order.urgent = urgent;
        order.status = OrderStatus::Created;
        order.delivery_cost = delivery_cost;
        order.created_at = Clock::get()?.unix_timestamp;
        order.assigned_at = 0;
        order.picked_up_at = 0;
        order.delivered_at = 0;
        order.depot = depot.key();
        order.driver = None;
        order.vehicle = None;
        order.is_crowd_sourced = false;
        order.order_id = platform.order_count;

        platform.order_count = platform
            .order_count
            .checked_add(1)
            .ok_or(CustomError::OverflowError)?;
        depot.available_capacity = depot
            .available_capacity
            .checked_sub(1)
            .ok_or(CustomError::OverflowError)?;
        Ok(())
    }
```

Listing 4.6: Create a new delivery order

Function: cancel_order

The `cancel_order` function allows a client to cancel an order before it progresses beyond the 'Created' state.

```
pub fn cancel_order(ctx: Context<CancelOrder>) -> Result<()>
    {
    let order = &mut ctx.accounts.order;
    require!(
        ctx.accounts.signer.key() == order.client,
        CustomError::Unauthorized
    );
    require!(
        order.status == OrderStatus::Created,
        CustomError::InvalidOrderStatus
    );
```

```
11
12      order.status = OrderStatus::Cancelled;
13      msg!("Order {} cancelled by client", order.order_id);
14      Ok(())
15  }
```

Listing 4.7: Cancel an order by the client

4.10.3 Assignment and Pickup

Function: accept_order_as_driver

The `accept_order_as_driver` function allows available crowd-sourced drivers to self-assign pending orders. Once accepted, the driver is marked unavailable and the order is locked. The `accept_order_as_vehicle` function provides the same capability to company-owned vehicles if they are located at the correct depot.

```
1   pub fn accept_order_as_driver(ctx: Context<
        AcceptOrderAsDriver>) -> Result<()> {
2       let order = &mut ctx.accounts.order;
3       let driver = &mut ctx.accounts.driver;
4
5       require!(
6           order.status == OrderStatus::Created,
7           CustomError::InvalidOrderStatus
8       );
9       require!(driver.is_crowd_sourced, CustomError::
            NotCrowdSourced);
10      require!(driver.is_available, CustomError::
            DriverNotAvailable);
11
12      order.status = OrderStatus::AssignedToDriver;
13      order.driver = Some(driver.key());
14      order.is_crowd_sourced = true;
15      order.assigned_at = Clock::get()?.unix_timestamp;
16      driver.is_available = false;
17
18      msg!(
19          "Order {} accepted by driver {}",
20          order.order_id,
21          driver.driver_id
22      );
23      msg!("Assignment result: Assigned to Crowd Driver");
24      Ok(())
25  }
```

Listing 4.8: Accept an order by a crowd-sourced driver

Function: accept_order_as_vehicle

```rust
pub fn accept_order_as_vehicle(ctx: Context<
    AcceptOrderAsVehicle>) -> Result<()> {
    let order = &mut ctx.accounts.order;
    let vehicle = &mut ctx.accounts.vehicle;

    require!(
        order.status == OrderStatus::Created,
        CustomError::InvalidOrderStatus
    );
    require!(vehicle.is_company_owned, CustomError::
        NotCompanyVehicle);
    require!(vehicle.is_available, CustomError::
        VehicleNotAvailable);
    require!(
        vehicle.current_depot == order.depot,
        CustomError::VehicleNotAtDepot
    );

    order.status = OrderStatus::AssignedToVehicle;
    order.vehicle = Some(vehicle.key());
    order.is_crowd_sourced = false;
    order.assigned_at = Clock::get()?.unix_timestamp;
    vehicle.is_available = false;

    msg!(
        "Order {} accepted by company vehicle {}",
        order.order_id,
        vehicle.vehicle_id
    );
    msg!("Assignment result: Assigned to Company Vehicle");
    Ok(())
}
```

Listing 4.9: Accept an order by a company-owned vehicle

Function: pick_up_order_driver

`pick_up_order_driver` confirms physical pickup of the package by the assigned crowd driver, updating the order to 'InTransit'. Similarly, `pick_up_order_vehicle` performs this role for company-owned vehicles.

```rust
pub fn pick_up_order_driver(ctx: Context<PickUpDriver>) ->
    Result<()> {
    let order = &mut ctx.accounts.order;
    let driver = &ctx.accounts.driver;

    require!(
        order.status == OrderStatus::AssignedToDriver,
        CustomError::InvalidOrderStatus
    );
    require!(
```

```
        order.driver == Some(driver.key()),
        CustomError::Unauthorized
    );
    require!(
        driver.driver_wallet == ctx.accounts.signer.key(),
        CustomError::Unauthorized
    );

    order.status = OrderStatus::InTransit;
    order.picked_up_at = Clock::get()?.unix_timestamp;
    msg!("Order {} picked up by driver", order.order_id);
    Ok(())
}
```

Listing 4.10: Pickup of an assigned order by a crowd driver

Function: pick_up_order_vehicle

```
pub fn pick_up_order_vehicle(ctx: Context<PickUpVehicle>) ->
    Result<()> {
    let order = &mut ctx.accounts.order;
    let vehicle = &ctx.accounts.vehicle;

    require!(
        order.status == OrderStatus::AssignedToVehicle,
        CustomError::InvalidOrderStatus
    );
    require!(
        order.vehicle == Some(vehicle.key()),
        CustomError::Unauthorized
    );
    require!(
        vehicle.owner == ctx.accounts.signer.key(),
        CustomError::Unauthorized
    );

    order.status = OrderStatus::InTransit;
    order.picked_up_at = Clock::get()?.unix_timestamp;
    msg!("Order {} picked up by vehicle", order.order_id);
    Ok(())
}
```

Listing 4.11: Pickup of an assigned order by a company vehicle

4.10.4 Delivery and Payment

The `deliver_order_driver` function finalizes a delivery handled by a crowd-sourced driver. It transfers the payment via a CPI call to the SPL Token program and calculates CO_2 emissions based on distance and fuel type. The `deliver_order_vehicle` function replicates this behavior for company-owned vehicles.

Function: deliver_order_driver

```
pub fn deliver_order_driver(ctx: Context<DeliverDriver>) ->
    Result<()> {
    let order = &mut ctx.accounts.order;
    let driver = &ctx.accounts.driver;
    let signer = ctx.accounts.signer.key();
    let depot = &ctx.accounts.depot;

    let distance_km = compute_distance(order, depot);

    require!(
        order.status == OrderStatus::InTransit,
        CustomError::InvalidOrderStatus
    );
    require!(
        order.driver == Some(driver.key()),
        CustomError::Unauthorized
    );
    require!(driver.driver_wallet == signer, CustomError::
        Unauthorized);

    order.status = OrderStatus::Delivered;
    order.delivered_at = Clock::get()?.unix_timestamp;

    let cpi_accounts = Transfer {
        from: ctx.accounts.client_token_account.
            to_account_info(),
        to: ctx.accounts.recipient_token_account.
            to_account_info(),
        authority: ctx.accounts.signer.to_account_info(),
    };
    let cpi_ctx = CpiContext::new(ctx.accounts.token_program
        .to_account_info(), cpi_accounts);
    let amount = (order.delivery_cost * 1_000_000.0) as u64;
    token::transfer(cpi_ctx, amount)?;

    let co2_emitted = distance_km * driver.co2_per_km;
    order.offset_cost = Some(adem_offset(co2_emitted));

    msg!(
        "Order {} delivered by driver and payment processed"
            ,
        order.order_id
    );
    Ok(())
}
```

Listing 4.12: Delivery confirmation by a crowd driver with token payment

Function: deliver_order_vehicle

```
pub fn deliver_order_vehicle(ctx: Context<DeliverVehicle>)
    -> Result<()> {
    let order = &mut ctx.accounts.order;
    let vehicle = &ctx.accounts.vehicle;
    let signer = ctx.accounts.signer.key();
    let depot = &ctx.accounts.depot;

    let distance_km = compute_distance(order, depot);

    require!(
        order.status == OrderStatus::InTransit,
        CustomError::InvalidOrderStatus
    );
    require!(
        order.vehicle == Some(vehicle.key()),
        CustomError::Unauthorized
    );
    require!(vehicle.owner == signer, CustomError::
        Unauthorized);

    order.status = OrderStatus::Delivered;
    order.delivered_at = Clock::get()?.unix_timestamp;

    let cpi_accounts = Transfer {
        from: ctx.accounts.client_token_account.
            to_account_info(),
        to: ctx.accounts.recipient_token_account.
            to_account_info(),
        authority: ctx.accounts.signer.to_account_info(),
    };
    let cpi_ctx = CpiContext::new(ctx.accounts.token_program
        .to_account_info(), cpi_accounts);
    let amount = (order.delivery_cost * 1_000_000.0) as u64;
    token::transfer(cpi_ctx, amount)?;

    let co2_emitted = distance_km * vehicle.co2_per_km;
    order.offset_cost = Some(adem_offset(co2_emitted));

    msg!(
        "Order {} delivered by vehicle and payment processed
            ",
        order.order_id
    );
    Ok(())
}
```

Listing 4.13: Delivery confirmation by a company vehicle with token payment

4.10.5 Sustainability

To enable reverse logistics and circular economy practices, the contract provides the `return_package` function for clients to mark a package as returned, along with a reason. The `return_packaging` function logs a packaging return event with an optional proof URL. Clients can be rewarded using `reward_for_packaging_return`, which mints tokens proportional to package size. The `schedule_packaging_return_pickup` function lets clients schedule packaging collection, and `return_pickup_completed` confirms the execution of this collection.

Function: return_package

```
pub fn return_package(ctx: Context<ReturnPackage>, reason:
    String) -> Result<()> {
    let order = &mut ctx.accounts.order;
    require!(
        order.status == OrderStatus::Delivered,
        CustomError::InvalidOrderStatus
    );

    order.status = OrderStatus::Returned;
    order.return_reason = Some(reason.clone());

    let platform = &mut ctx.accounts.platform;
    platform.returned_package_count = platform
        .returned_package_count
        .checked_add(1)
        .ok_or(CustomError::OverflowError)?;

    msg!(
        "Order {} marked as returned. Reason: {}",
        order.order_id,
        reason
    );
    Ok(())
}
```

Listing 4.14: Mark a delivered order as returned

Function: return_packaging

```
pub fn return_packaging(
    ctx: Context<ReturnPackaging>,
    proof_url: Option<String>,
) -> Result<()> {
    let order = &ctx.accounts.order;
    let return_record = &mut ctx.accounts.packaging_return;

    require!(
        order.status == OrderStatus::Delivered,
```

```
        CustomError::InvalidOrderStatus
    );
    require!(
        ctx.accounts.client.key() == order.client,
        CustomError::Unauthorized
    );

    return_record.order = order.key();
    return_record.client = ctx.accounts.client.key();
    return_record.returned_at = Clock::get()?.unix_timestamp;
    return_record.proof_url = proof_url;

    msg!(
        "Packaging returned for order {} with proof {:?}",
        order.order_id,
        return_record.proof_url
    );
    Ok(())
}
```

Listing 4.15: Client submits proof of packaging return

Function: reward_for_packaging_return

```
pub fn reward_for_packaging_return(ctx: Context<RewardPackagingReturn>) -> Result<()> {
    let order = &ctx.accounts.order;

    require!(
        ctx.accounts.reward_mint.mint_authority
            == COption::Some(ctx.accounts.mint_authority.key()),
        CustomError::Unauthorized
    );

    let reward_per_unit: u64 = 100_000;
    let amount = reward_per_unit
        .checked_mul(order.package_size as u64)
        .ok_or(CustomError::OverflowError)?;

    let cpi_accounts = MintTo {
        mint: ctx.accounts.reward_mint.to_account_info(),
        to: ctx.accounts.client_token_account.to_account_info(),
        authority: ctx.accounts.mint_authority.to_account_info(),
    };
    let cpi_ctx = CpiContext::new(ctx.accounts.token_program.to_account_info(), cpi_accounts);
    token::mint_to(cpi_ctx, amount)?;
```

```
22
23      msg!(
24          " Rewarded {} tokens ({} x {}) to client {:?}",
25          amount,
26          order.package_size,
27          reward_per_unit,
28          ctx.accounts.client_token_account.owner
29      );
30
31      Ok(())
32  }
```

Listing 4.16: Mint SPL token rewards for packaging return

Function: schedule_packaging_return_pickup

```
1   pub fn schedule_packaging_return_pickup(
2       ctx: Context<ScheduleReturnPickup>,
3       proof_url: Option<String>,
4   ) -> Result<()> {
5       let order = &mut ctx.accounts.order;
6       let return_record = &mut ctx.accounts.packaging_return;
7
8       require!(
9           order.status == OrderStatus::Delivered,
10          CustomError::InvalidOrderStatus
11      );
12      require!(
13          ctx.accounts.client.key() == order.client,
14          CustomError::Unauthorized
15      );
16
17      order.status = OrderStatus::ReturnScheduled;
18
19      return_record.order = order.key();
20      return_record.client = ctx.accounts.client.key();
21      return_record.returned_at = Clock::get()?.unix_timestamp
            ;
22      return_record.proof_url = proof_url;
23
24      msg!(
25          " Return pickup scheduled for order {}",
26          order.order_id
27      );
28      Ok(())
29  }
```

Listing 4.17: Schedule pickup for returned packaging

Function: return_pickup_completed

```rust
pub fn return_pickup_completed(ctx: Context<
    CompleteReturnPickup>) -> Result<()> {
    let order = &mut ctx.accounts.order;

    require!(
        order.status == OrderStatus::ReturnScheduled,
        CustomError::InvalidOrderStatus
    );

    order.status = OrderStatus::ReturnPickedUp;

    msg!(
        " Packaging return picked up for order {}",
        order.order_id
    );
    Ok(())
}
```

Listing 4.18: Confirm that return pickup has been completed

4.10.6 Ratings

The `rate_driver` function allows clients to rate crowd drivers post-delivery, which helps maintain service quality. The `rate_vehicle` function allows for similar feedback on company-owned delivery vehicles.

Function: rate_driver

```rust
pub fn rate_driver(ctx: Context<RateDriver>, rating: u8) ->
    Result<()> {
    let driver = &mut ctx.accounts.driver;
    require!(rating <= 5, CustomError::InvalidRating);

    driver.total_ratings += 1;
    driver.rating = ((driver.rating as u32 * (driver.
        total_ratings - 1) + rating as u32)
        / driver.total_ratings) as u8;

    msg!(
        "Driver {} rated with {}. New average: {}",
        driver.driver_id,
        rating,
        driver.rating
    );
    Ok(())
}
```

Listing 4.19: Rate a crowd-sourced driver after delivery

Function: rate_vehicle

```
pub fn rate_vehicle(ctx: Context<RateVehicle>, rating: u8)
    -> Result<()> {
    let vehicle = &mut ctx.accounts.vehicle;
    require!(rating <= 5, CustomError::InvalidRating);

    vehicle.total_ratings += 1;
    vehicle.rating = ((vehicle.rating as u32 * (vehicle.
        total_ratings - 1) + rating as u32)
        / vehicle.total_ratings) as u8;

    msg!(
        "Vehicle {} rated with {}. New average: {}",
        vehicle.vehicle_id,
        rating,
        vehicle.rating
    );
    Ok(())
}
```

Listing 4.20: Rate a company-owned vehicle after delivery

Metadata and Routing

The `see_optimized_route` function enables the platform to store a URL to external route metadata (e.g., IPFS-hosted maps). Both `set_vehicle_metadata_url` and `set_driver_metadata_url` allow for associating additional off-chain metadata such as documents, certificates, or vehicle details to support transparency and observability.

Function: see_optimized_route

```
pub fn see_optimized_route(ctx: Context<SeeOptimizedRoute>,
    url: String) -> Result<()> {
    let order = &mut ctx.accounts.order;
    order.delivery_metadata_url = url;
    Ok(())}
```

Listing 4.21: Store a URL to the optimized route metadata

Function: set_vehicle_metadata_url

```
pub fn set_vehicle_metadata_url(ctx: Context<
    SetVehicleMetadataUrl>, metadata_url: String) -> Result
    <()> {
    let vehicle = &mut ctx.accounts.vehicle;
    vehicle.metadata_url = Some(metadata_url);
    Ok(())}
```

Listing 4.22: Attach metadata URL to a company vehicle

Function: set_driver_metadata_url

```rust
pub fn set_driver_metadata_url(ctx: Context<
    SetDriverMetadataUrl>, metadata_url: String) -> Result<()
    > {
     let driver = &mut ctx.accounts.driver;
     driver.metadata_url = Some(metadata_url);
     Ok(())}
```

Listing 4.23: Attach metadata URL to a driver

Python Script to Retrieve On-chain Vehicle Metadata

The following script demonstrates how to interact with the Solana blockchain using Python. It loads a local wallet and IDL, connects to the `devnet`, and retrieves the details of a registered vehicle account using `anchorpy` and `solders`.

```python
import asyncio, json
from solders.pubkey import Pubkey
from solders.keypair import Keypair
from solana.rpc.async_api import AsyncClient
from anchorpy import Program, Provider, Wallet, Idl
from pathlib import Path

async def get_vehicle_info():
    wallet_path = Path("wallet1.json") #PrivateKey of our
        wallet
    idl_path = Path("idl.json") ##downloaded from Solana
        playground
    vehicle_pubkey = Pubkey.from_string("...")

    # Load wallet
    secret = json.load(wallet_path.open())
    keypair = Keypair.from_bytes(bytes(secret))

    # Connect to Solana devnet
    client = AsyncClient("https://api.devnet.solana.com")
    wallet = Wallet(keypair)
    provider = Provider(client, wallet)

    # Load program from IDL
    idl = Idl.from_json(json.dumps(json.load(idl_path.open()
        )))
    program_id = Pubkey.from_string("
        ABY2kcwaJnyZzi5WQWhyJkZqEmf4AtjU5Gm3zpyvUAFt")
    program = Program(idl, program_id, provider)

    # Fetch on-chain vehicle account
    vehicle = await program.account["Vehicle"].fetch(
        vehicle_pubkey)
    await client.close()

```

```
    return {
        "type": vehicle.vehicle_type,
        "plate": vehicle.license_plate,
        "co2": vehicle.co2_per_km,
        "rating": vehicle.rating,
    }

vehicle_data = asyncio.run(get_vehicle_info())
print(json.dumps(vehicle_data, indent=2))
```

Listing 4.24: Fetch vehicle metadata using Python and AnchorPy

Python Script to Upload Metadata to IPFS Using Pinata

The following script illustrates how optimized delivery data (Results from VRP solved in the previous chapter can be uploaded to IPFS using Pinata's REST API. The JSON content is prepared off-screen and passed securely to the upload function.

```
import requests
import json

PINATA_API_KEY = "your_api_key"
PINATA_SECRET_API_KEY = "your_secret_key"

def upload_json_to_pinata(json_data):
    url = "https://api.pinata.cloud/pinning/pinJSONToIPFS"
    headers = {
        "Content-Type": "application/json",
        "pinata_api_key": PINATA_API_KEY,
        "pinata_secret_api_key": PINATA_SECRET_API_KEY,
    }
    response = requests.post(
        url,
        data=json.dumps({"pinataContent": json_data}),
        headers=headers
    )
    response.raise_for_status()
    return response.json()["IpfsHash"]

# Load or generate metadata (abstracted here)
cid = upload_json_to_pinata(my_metadata)
print("Uploaded to IPFS:", cid)
```

Listing 4.25: Upload JSON metadata to IPFS via Pinata

Minimal Client Script

The code below summarizes the main interaction workflow with the Solana smart contract using Anchor's JavaScript framework. The script initializes the delivery platform, registers a depot, and creates a delivery order on the Solana blockchain using Anchor. It simulates the full transaction flow: order creation, driver assignment, pickup, and delivery. Each step is executed via remote procedure calls (RPC) using the Anchor JavaScript client.

```
const platform = web3.Keypair.generate();
await pg.program.methods.initialize("LMD").accounts({
  platform: platform.publicKey,
  admin: pg.wallet.publicKey,
  systemProgram: web3.SystemProgram.programId,
}).signers([platform]).rpc();

const depot = web3.Keypair.generate();
await pg.program.methods.registerDepot("Main", 34.88, -1.28,
    1000).accounts({
  platform: platform.publicKey,
  depot: depot.publicKey,
  admin: pg.wallet.publicKey,
  systemProgram: web3.SystemProgram.programId,
}).signers([depot]).rpc();

const order = web3.Keypair.generate();
await pg.program.methods.createOrder("Client A", "Mansourah"
   , 34.870748, -1.338889, 2, true).accounts({
  platform: platform.publicKey,
  depot: depot.publicKey,
  order: order.publicKey,
  client: pg.wallet.publicKey,
  systemProgram: web3.SystemProgram.programId,
}).signers([order]).rpc();

await pg.program.methods.acceptOrderAsDriver().accounts({
   order: order.publicKey, driver: driver.publicKey }).rpc()
   ;
await pg.program.methods.pickUpOrderDriver().accounts({
   order: order.publicKey, driver: driver.publicKey, signer:
    pg.wallet.publicKey }).rpc();
await pg.program.methods.deliverOrderDriver().accounts({
  order: order.publicKey,
  driver: driver.publicKey,
  signer: pg.wallet.publicKey,
  depot: depot.publicKey,
  clientTokenAccount,
  recipientTokenAccount,
  tokenProgram: TOKEN_PROGRAM_ID,
}).rpc();
```

Listing 4.26: Minimal client-side interaction using Anchor

Minimal Smart Contract Testing via Anchor

A test file was developed using `Mocha` and `Anchor`'s TypeScript framework to validate the full lifecycle of the decentralized delivery platform. The script initializes the platform, registers entities (depots, vehicles, drivers), creates and processes orders, and simulates sustainability actions such as packaging returns and reward minting.

```
it("initializes the platform", async () => {
  const txHash = await pg.program.methods
    .initialize("LMD")
    .accounts({
      platform: platformKeypair.publicKey,
      admin: pg.wallet.publicKey,
      systemProgram: web3.SystemProgram.programId,
    })
    .signers([platformKeypair])
    .rpc();

  const platformAccount = await pg.program.account.
     deliveryPlatform.fetch(
    platformKeypair.publicKey
  );

  assert.equal(platformAccount.platformName, "LMD");
  assert.equal(platformAccount.active, true);
});
```

Listing 4.27: Platform initialization test case

4.11 Building and deploying the smart contract

Figure 4.14 shows the successful build and deployment of the smart contract on the Solana blockchain using Anchor. The build phase completed in 15.2 seconds, and the deployment phase—despite some transaction confirmation retries—completed successfully, indicating proper compilation and network propagation.

```
Building...
Build successful. Completed in 15.20s.

Deploying... This could take a while depending on the program size and network conditions.
Warning: 32 transactions not confirmed, retrying...
Warning: 7 transactions not confirmed, retrying...
Deployment successful. Completed in 53s.
```

Figure 4.14: Building and deploying the program

To verify the deployment on-chain, we used the Solana Explorer. Figure 4.15 displays the program account details, confirming that the contract is executable, upgradeable, and successfully linked to its deployed address.

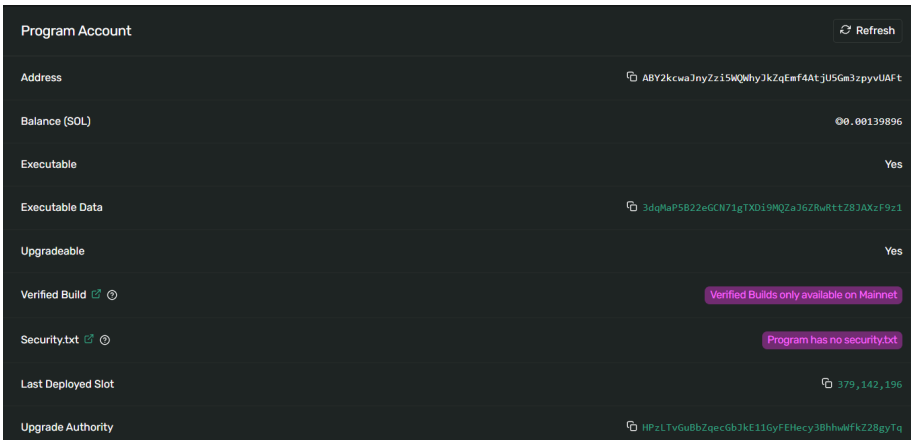


Figure 4.15: Deployement verification

Figure 4.16 presents the transaction history with a status of `Success`.

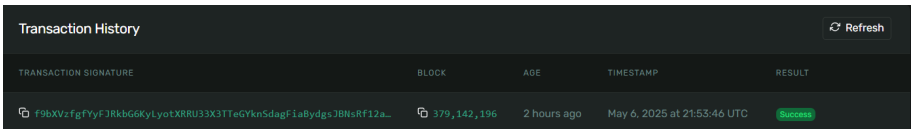


Figure 4.16: Successed Transaction

The successful deployment of the last mile delivery smart contract marks the final step in implementing our decentralized last-mile logistics solution. By integrating core functionalities such as order lifecycle management, sustainability incentives, and token-based payment mechanisms, the system is now operational and verifiable on-chain. The results demonstrate the feasibility of using Solana smart contracts for managing real-world logistics workflows in a transparent, scalable, and sustainable manner.

4.12 Smart contract Testing

To ensure the correctness and robustness of the implemented smart contract, we designed and executed three distinct test scenarios using the Anchor test suite in Solana Playground. Each function was evaluated across multiple scenarios to ensure correctness, robustness, and expected state transitions, each of the scenario simulates a delivery workflow. Three test files (`anchor.test.ts`, `test1.test.ts`, `test2.test.ts`) were executed, covering the full lifecycle: platform initialization, entity registration (depots, vehicles, drivers), order creation and assignment, SPL token-based delivery payment, packaging return, reverse logistics, reward minting, and rating mechanisms.

4.12.1 Scenario 1

In this scenario, we test the lifecycle of a delivery order that ends in a return due to customer unavailability. The goal is to validate the platform's ability to handle unsuccessful deliveries and track returned packages securely.

The process begins with the successful initialization of the delivery platform and the registration of a depot and a company vehicle. A crowd-sourced driver is also registered. An order is then created and automatically assigned to the available crowd driver.

As shown in Figure 4.17, the driver successfully picks up the order. However, during the delivery attempt, the client is unavailable. As a result, the system invokes the `returnPackage` function, recording the return status and reason within the order's on-chain data.

This test scenario verifies:

- Order lifecycle transitions (`Created` → `AssignedToDriver` → `InTransit` → `Returned`)
- Status tracking and storage of return reasons
- Correct linkage of the order to the driver and depot

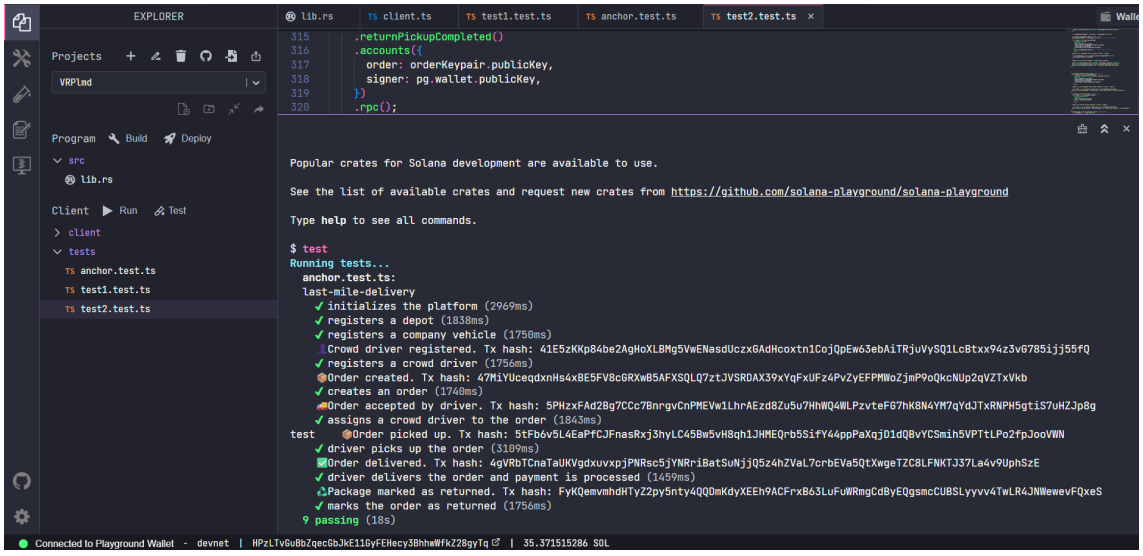


Figure 4.17: Package Return

4.12.2 Scenario 2

In the second scenario, we evaluate a successful delivery handled by the driver, followed by a circular economy process including packaging return, pickup scheduling, and tokenized reward issuance.

The sequence begins with initializing the platform and registering a depot, a company vehicle, and a crowd driver. Once an order is created, the system assigns it to the available crowd driver. The driver picks up and successfully delivers the package, triggering the SPL token-based payment mechanism.

After delivery, the client returns the packaging. The system securely stores proof of return (referenced by an IPFS hash), schedules the pickup, and marks the pickup as complete once collected. Finally, the client receives a reward in the form of a newly minted SPL token, and the driver is rated through an on-chain feedback mechanism.

Figure 4.18 illustrates the full lifecycle of this delivery flow.

Key features validated in this test:

- End-to-end lifecycle with successful delivery and post-delivery logistics
- Integration of IPFS-based proof of packaging return
- Scheduling and confirmation of reverse logistics
- Minting of a packaging reward token and on-chain rating

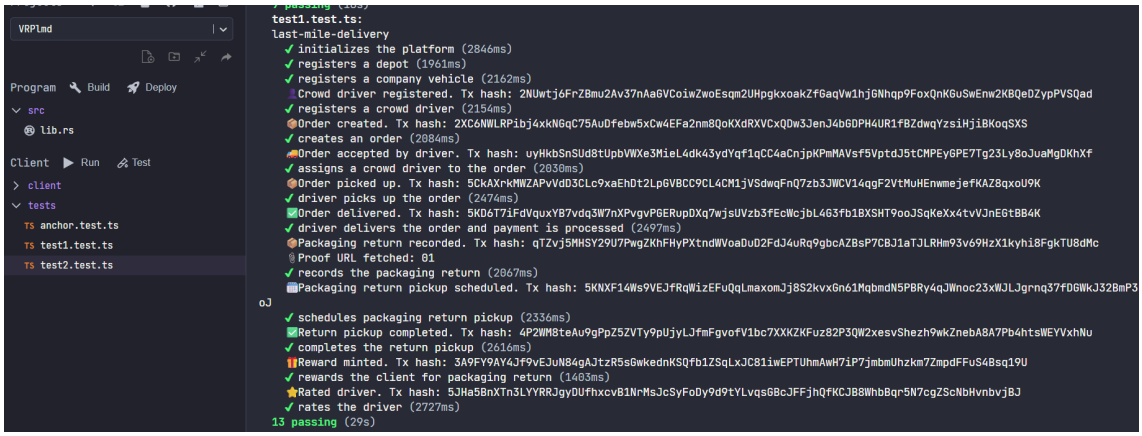


Figure 4.18: Packaging return for circular economy

4.12.3 Scenario 3

In this scenario, the last-mile delivery task is executed using a company-owned vehicle instead of a crowd-sourced driver. This test demonstrates the system's flexibility in handling hybrid fleet structures and verifies that circular economy principles are maintained regardless of the vehicle source.

After the platform and depot are initialized, both company and crowd drivers are registered. The system proceeds by creating a delivery order and assigning it to a company vehicle. The vehicle picks up and successfully delivers the order, triggering on-chain token payment. The client subsequently returns the packaging, and the system logs the proof (a URL or hash), schedules a return pickup, and confirms the pickup process.

As a final step, the client receives a reward token (minted via SPL token instructions), and the vehicle is rated for performance. This scenario confirms that both forward and reverse logistics, along with incentivization, are fully supported for enterprise-owned delivery assets.

- Validates hybrid fleet operations with company-owned vehicles
- Confirms carbon and packaging return flows for closed-loop logistics
- Demonstrates token-based reward logic across both fleet types

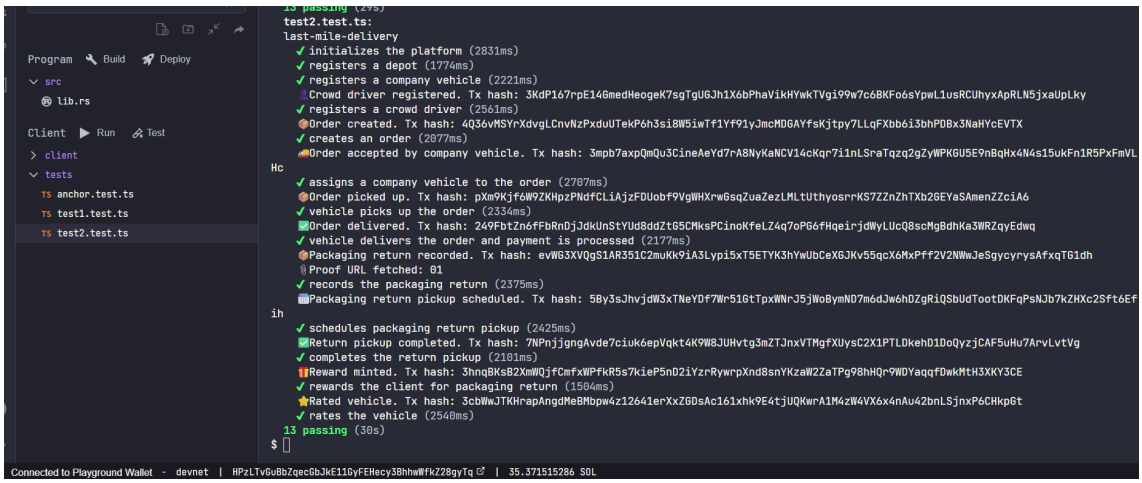


Figure 4.19: Delivery by Company Vehicle with Return and Reward

To validate the successful execution of delivery and payment transactions on the Solana blockchain, we include transaction confirmation logs retrieved from the Solana explorer. These logs provide proof of the contract's correct execution state, resource consumption, and token transfers.

Figure 4.20 presents the confirmed transaction for the vehicle-based order delivery on the Solana blockchain. The operation was finalized successfully with a minimal network fee of 0.000005 SOL, consuming 30,072 compute units.

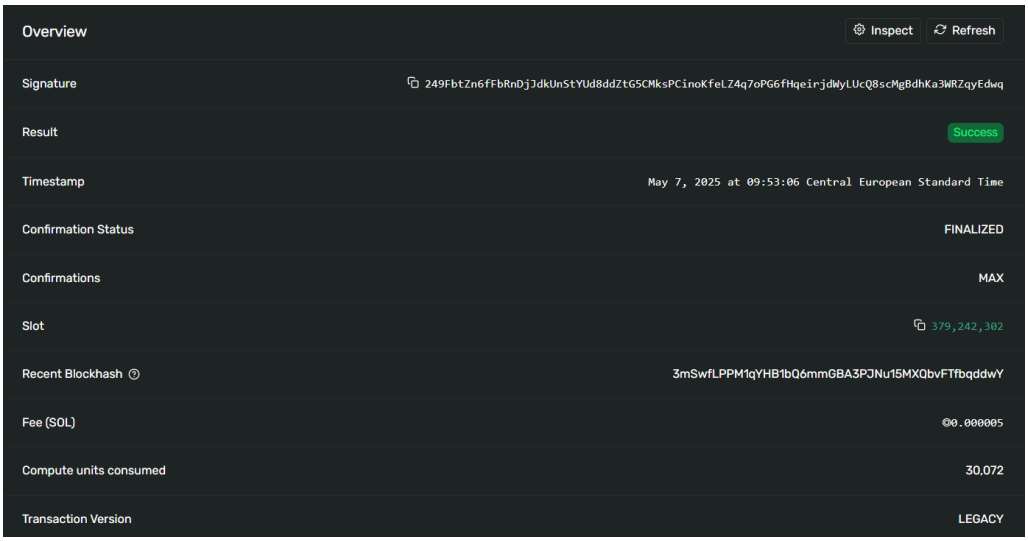


Figure 4.20: Transaction Overview

Figure 4.21shows the program logs for the successful execution of the DeliverOrderVehicle instruction, confirming token transfer and delivery finalization and returned success for the invoked operations.

Figure 4.21: Program Instruction Logs

4.12.4 Client Interaction and Workflow Validation

To validate the complete last-mile delivery lifecycle from an off-chain perspective, the script `client.ts` was executed in the Solana Playground environment. This TypeScript client simulates real-world interactions with the deployed Anchor smart contract by sequentially invoking all major functions.

The script begins by initializing the `DeliveryPlatform` and registering both a company-owned vehicle and a crowd-sourced driver. An order is created and self-assigned by the crowd driver, transitioning the order status to `inTransit` once the pickup is confirmed. Upon successful delivery, the program handles the token-based payment transfer securely through the Solana Program Library (SPL).

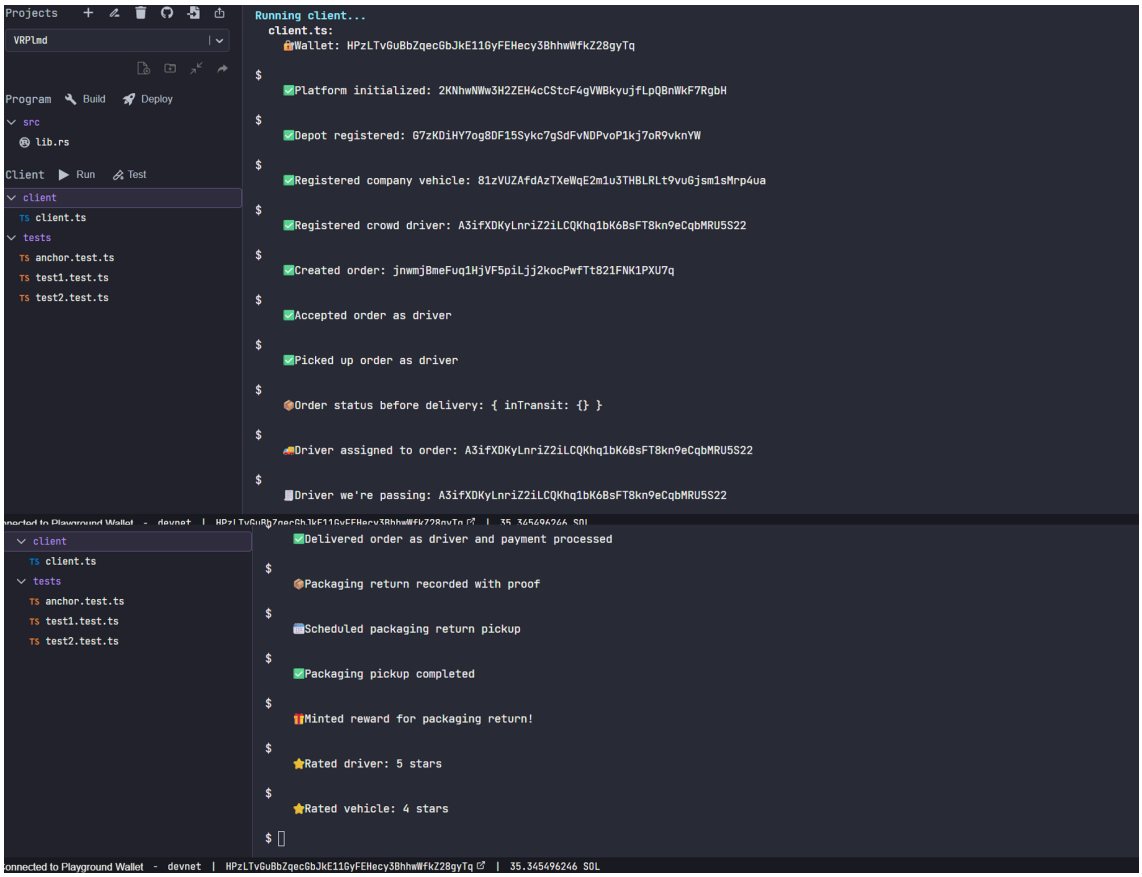


Figure 4.22: Client interaction using client.ts

Following the delivery, circular economy features are triggered. The client records a packaging return with proof, schedules a pickup, and confirms its completion. Upon successful validation, a reward token is minted and transferred to the client's wallet as an incentive. The client is then able to rate the driver and vehicle, completing the feedback process. In the demonstrated case, the driver was rated 5 stars and the vehicle 4 stars.

4.13 Result of the Offchain integration

To validate the off-chain integration with the Solana blockchain, we implemented a Python script using the `anchorpy` library. This script connects to the Devnet network, loads a wallet and program IDL, and queries the `Vehicle` account using its public key. The following Figure 4.23 shows that the data from the Onchain part was successfully retrieved:

Figure 4.23: Off-chain integration: querying a vehicle account from Solana using AnchorPy

This confirms the correct interaction between the off-chain Python client and the on-chain smart contract account data, enabling seamless metadata access from external systems.

To ensure route data integrity and traceability, optimized vehicle routes that we obtained in our mathematical model, are serialized into JSON format and pinned to IPFS using the Pinata API, 4.24.

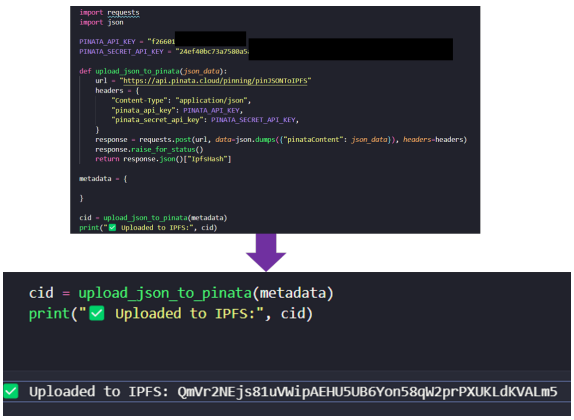


Figure 4.24: Generation of CID in pinata IPFS of the VRPTW result

Upon execution, the payload is successfully pinned to IPFS, resulting a unique content identifier (CID). This CID can then be stored on-chain for later retrieval 4.25.

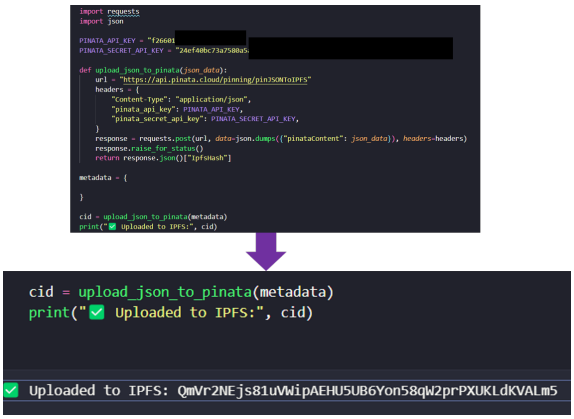


Figure 4.25: Generation of CID in Pinata which contains the result of the VRPTW

The CID generated in the Jypyter environment can be found directly in Pinata environment containing the result of the VRPTW as follow 4.26:

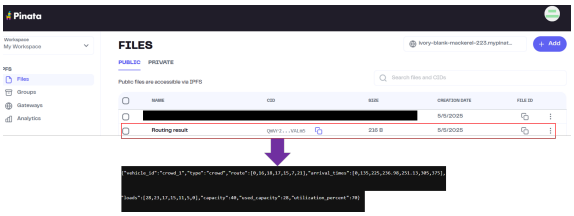


Figure 4.26: Result of the routing on IPFS pinata

Conclusion

This chapter presented the design, development, and deployment of a decentralized last-mile delivery framework based on the Solana blockchain. By using smart contracts, tokenized payments, and off-chain storage via IPFS, the proposed system ensures transparency, automation, and traceability throughout the delivery lifecycle. The implementation covered the full range of operations—from platform initialization and order management to sustainability features such as carbon tracking and packaging return rewards. Through a modular architecture and interaction with Phantom wallets, client scripts, and Solana Playground, the framework was successfully tested with different delivery scenarios. The integration of off-chain metadata and decentralized storage further strengthens the scalability and efficiency of the system. This implementation demonstrates the feasibility and advantages of using blockchain in logistics, setting a robust foundation for future research on sustainable and trustless supply chain ecosystems.

General conclusion

This thesis explored the design and implementation of a hybrid framework for last-mile delivery, combining optimization models and blockchain-based decentralized systems. Traditional last-mile logistics suffer from centralized architectures that limit transparency, introduce single points of failure, and restrict fair participation. Moreover, the increasing need for eco-conscious, cost-effective, and resilient delivery systems calls for novel technological interventions.

To address these challenges, the first part of this thesis proposed a mathematical formulation for the Hybrid Vehicle Routing Problem with Time Windows (HVRPTW), incorporating both company-owned and crowdsourced delivery agents. Using Gurobi and Simulated Annealing, the model demonstrated the potential to minimize operational costs while respecting real-world constraints such as time windows and vehicle capacities.

The second part of the thesis introduced a decentralized logistics architecture built on the Solana blockchain. Through the development of smart contract, key delivery functions were securely automated from depot registration to payment and sustainability rewards. The use of IPFS enabled scalable off-chain metadata handling, while smart contracts ensured trust, transparency, and verifiability of delivery operations.

The results show that integrating optimization with decentralized execution provides a robust, fair, and transparent solution to last-mile delivery challenges. The proposed system not only reduces delivery costs and emissions but also increases stakeholder trust through blockchain traceability and automation.

Future Work

This work lays the foundation for several promising research and development directions:

- **Frontend DApp Development:** As a next step, the system will be extended with a user-friendly decentralized application (DApp) interface to enable real-time interaction with the smart contract.
- **Reputation and Incentive Mechanisms:** A future enhancement involves integrating a dynamic reputation system that rewards reliable drivers and vehicles while penalizing service violations.
- **Multi-chain Interoperability:** Exploring deployment on alternative blockchain platforms (like Ethereum) and comparing performance, cost, and scalability.
- **IoT Integration:** Incorporating IoT devices (e.g., GPS trackers, CO_2 sensors) to provide real-time data that enriches routing and sustainability features.

These directions aim to strengthen the practical applicability, scalability, and security of decentralized last-mile delivery systems, making them viable for large-scale urban deployment.

Bibliography

[1] Amazon Web Services (AWS). *What is Blockchain?* https://aws.amazon.com/what-is/blockchain/. Accessed: April 3, 2025. 2024.

[2] Ameen Alam. *How Blockchain Works Step by Step*. Accessed: April 5,2025. 2023. URL: https://medium.com/@ameenalam202/how-blockchain-works-step-by-step-1a9d19d5e787.

[3] Ala' Alqaisi, Sherif Saad, and Mohammad Mamun. "Trustworthy Decentralized Last-Mile Delivery Framework Using Blockchain". In: *Proceedings of the 20th International Conference on Smart Business Technologies*. SCITEPRESS - Science and Technology Publications, Lda, 2023, pp. 54–65. DOI: 10.5220/0012090300003552.

[4] Riham AlTawy et al. "Lelantos: A Blockchain-based Anonymous Physical Delivery System". In: *2017 15th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 2017, pp. 205–212. DOI: 10.1109/PST.2017.00013.

[5] Rodrigo Rezende Amaral and El-Houssaine Aghezzaf. "City logistics and traffic management: Modelling the inner and outer urban transport flows in a two-tiered system". In: *Transportation Research Procedia* 6 (2015), pp. 297–312.

[6] Amazon Freight. *Middle Mile Delivery*. Accessed: March 22, 2025. URL: https://freight.amazon.co.uk/newsroom/middle-mile-delivery.

[7] Amazon Web Services. *AWS Business Support*. https://aws.amazon.com/fr/premiumsupport/plans/business/. Accessed on April 7, 2025. 2025.

[8] Claudia Archetti, Martin Savelsbergh, and Grazia Speranza. "The Vehicle Routing Problem with Occasional Drivers". In: *European Journal of Operational Research* (2016). DOI: 10.1016/j.ejor.2016.03.049.

[9] ASCM. *SCOR Digital Standard*. SCOR Version 14.0. All rights reserved. Available at https://scor.ascm.org. 2022.

[10] Mateen Ashraf and Cathal Heavey. "A prototype of supply chain traceability using solana as blockchain and IoT". In: *Procedia Computer Science* 217 (2023), pp. 948–959.

[11] Mohsen Attaran and Angappa Gunasekaran. *Applications of Blockchain Technology in Business: Challenges and Opportunities*. Chapter 3. Springer Cham, 2019, pp. 13–20. DOI: 10.1007/978-3-030-27798-7.

[12] Amrou Awaysheh and Robert D Klassen. "The impact of supply chain structure on the use of supplier socially responsible practices". In: *International journal of operations & production management* 30.12 (2010), pp. 1246–1268.

[13] Leo Maxim Bach, Branko Mihaljevic, and Mario Zagar. "Comparative analysis of blockchain consensus algorithms". In: *2018 41st international convention on information and communication technology, electronics and microelectronics (MIPRO)*. Ieee. 2018, pp. 1545–1550.

[14] Arshdeep Bahga and Vijay K Madisetti. "Blockchain platform for industrial internet of things". In: *Journal of Software Engineering and Applications* 9.10 (2016), pp. 533–546.

[15] Chunguang Bai and Joseph Sarkis. "A supply chain transparency and sustainability technology appraisal model for blockchain technology". In: *International journal of production research* 58.7 (2020), pp. 2142–2162.

[16] T. Bányai. "Real-Time Decision Making in First Mile and Last Mile Logistics: How Smart Scheduling Affects Energy Efficiency of Hyperconnected Supply Chain Solutions". In: *Energies* 11 (2018), p. 1833.

[17] Tamás Bányai, Béla Illés, and Ágota Bányai. "Smart Scheduling: An Integrated First Mile and Last Mile Supply Approach". In: *Complexity* (2018). DOI: 10.1155/2018/5180156.

[18] Abdelghani Bekrar et al. "Digitalizing the closing-of-the-loop for supply chains: A transportation and blockchain perspective". In: *Sustainability* 13.5 (2021), p. 2895.

[19] Abderrahim Benjelloun and Teodor Gabriel Crainic. "Trends, challenges and perspectives in city logistics". In: *Buletin AGIR* 4 (Jan. 2009).

[20] Muhammad Nasir Mumtaz Bhutta et al. "A survey on blockchain technology: Evolution, architecture and security". In: *Ieee Access* 9 (2021), pp. 61048–61073.

[21] Nils Boysen, Stefan Fedtke, and Stefan Schwerdfeger. "Last-mile delivery concepts: a survey from an operational research perspective". In: *OR Spectrum* 43 (2021), pp. 1–58. DOI: 10.1007/s00291-020-00607-8.

[22] Michael L. Bynum et al. *Pyomo–optimization modeling in python*. Third. Vol. 67. Springer Science & Business Media, 2021.

[23] Vittorio Capocasale. "Blockchain applications to Supply Chain: an application to last-mile delivery". MA thesis. Politecnico di Torino, 2019.

[24] Miguel Pincheira Caro et al. "Blockchain-based traceability in Agri-Food supply chain management: A practical implementation". In: *2018 IoT Vertical and Topical Summit on Agriculture - Tuscany (IOT Tuscany)*. 2018, pp. 1–4. DOI: 10.1109/IOT-TUSCANY.2018.8373021.

[25] Vincent E. Castillo et al. "Crowdsourcing Last Mile Delivery: Strategic Implications and Future Research Directions". In: *Journal of Business Logistics* (2017).

[26] Zoran Cekerevac, Milanka Bogavac, and Lyudmila Prigoda. "A Review of Blockchain Application in Logistics and Last-Mile Delivery". In: *MEST Journal* 12.1 (2024), pp. 4–12. DOI: 10.12709/mest.12.12.01.02. URL: https://www.meste.org/mest/MEST_Najava/XXII_Cekerevac.pdf.

[27] CFTE Education. *Distributed Ledger in Blockchain*. Accessed: April 3, 2025. 2023. URL: https://blog.cfte.education/distributed-ledger-in-blockchain/.

[28] Arun Chandramouli. “Optimizing Last-Mile Delivery Operations: Leveraging Predictive Analytics, Technology Integration, and Sustainable Practices”. In: *Journal of Mathematical & Computer Applications* 2.3 (2023), pp. 1–7. DOI: 10.47363/JMCA/2023(2)145.

[29] Nguyen Quoc Chinh et al. “Collaborative vehicle routing problem for urban last-mile logistics”. In: *2016 IEEE international conference on systems, man, and cybernetics (SMC)*. IEEE. 2016, pp. 001531–001536.

[30] Sunil Chopra and Peter Meindl. *Supply Chain Management: Strategy, Planning, and Operation*. 5th. Upper Saddle River, NJ: Pearson, 2013.

[31] Mohammad Jabed Morshed Chowdhury et al. “Blockchain versus database: A critical analysis”. In: *2018 17th IEEE International conference on trust, security and privacy in computing and communications/12th IEEE international conference on big data science and engineering (TrustCom/BigDataSE)*. IEEE. 2018, pp. 1348–1353.

[32] Konstantinos Christidis and Michael Devetsikiotis. “Blockchains and Smart Contracts for the Internet of Things”. In: *IEEE Access* 4 (2016), pp. 2292–2303. DOI: 10.1109/ACCESS.2016.2566339.

[33] Konstantinos Christidis and Michael Devetsikiotis. “Blockchains and smart contracts for the internet of things”. In: *IEEE access* 4 (2016), pp. 2292–2303.

[34] Martin Christopher. *Logistics & Supply Chain Management*. 4th. Harlow, UK: Pearson Education Limited, 2011.

[35] H. Chu et al. “Data-Driven Optimization for Last-Mile Delivery”. In: *Complex & Intelligent Systems* 9 (2023), pp. 2271–2284. DOI: 10.1007/s40747-021-00293-1.

[36] Corporate Finance Institute. *Supply Chain*. https://corporatefinanceinstitute.com/resources/management/supply-chain/. Accessed: March 22, 2025.

[37] Online Hash Crack. *Hashing in Blockchain explained*. Accessed: April 3,2025. 2025. URL: https://www.onlinehashcrack.com/how-to-hashing-in-blockchain-explained.php.

[38] Richard E. Crandall, William R. Crandall, and Charlie C. Chen. *Principles of Supply Chain Management*. 2nd. Boca Raton, FL: CRC Press, Taylor & Francis Group, 2015. ISBN: 978-1-4822-1202-0.

[39] George B Dantzig and John H Ramser. “The truck dispatching problem”. In: *Management science* 6.1 (1959), pp. 80–91.

[40] DHL. *What is Logistics?* https://www.dhl.com/discover/en-global/logistics-advice/import-export-advice/what-is-logistics. Accessed: March 21, 2025.

[41] Parisa Dolati Neghabadi, Karine Evrard Samuel, and Marie-Laure Espinouse. “Systematic literature review on city logistics: overview, classification and analysis”. In: *International Journal of Production Research* (2018). DOI: 10.1080/00207543.2018.1489153.

[42] Alexandre Dolgui et al. “Blockchain-oriented dynamic modelling of smart contract design and execution in the supply chain”. In: *International Journal of Production Research* (2019). DOI: 10.1080/00207543.2019.1627439.

[43] Ali Dorri, Salil S Kanhere, and Raja Jurdak. “Blockchain in internet of things: challenges and solutions”. In: *arXiv preprint arXiv:1608.05187* (2016).

[44] Soufiane El Moudaa et al. “PackChain: Toward a Blockchain-based Management Platform for Last-mile Delivery”. In: *Proceedings of the Relevant Conference*. 2020.

[45] Feruz Elmay et al. “Digital Twins and Dynamic NFTs for Blockchain-Based Crowdsourced Last-Mile Delivery”. In: *Information Processing and Management* 61 (2024), p. 103756. DOI: 10.1016/j.ipm.2024.103756. URL: https://doi.org/10.1016/j.ipm.2024.103756.

[46] M. Faccio and M. Gamberi. “New City Logistics Paradigm: From the “Last Mile” to the “Last 50 Miles” Sustainable Distribution”. In: *Sustainability* 7 (2015), pp. 14873–14894.

[47] X. Fan et al. “CrowdNavi: Demystifying Last Mile Navigation With Crowdsourced Driving Information”. In: *IEEE Transactions on Industrial Informatics* 13 (2017), pp. 771–781.

[48] Kurt Fanning and David P Centers. “Blockchain and its coming impact on financial services”. In: *Journal of Corporate Accounting & Finance* 27.5 (2016), pp. 53–57.

[49] M.A. Figliozzi. “Lifecycle modeling and assessment of unmanned aerial vehicles (Drones) CO2e emissions”. In: *Transportation Research Part D: Transport and Environment* 57 (2017), pp. 251–261.

[50] Solana Foundation. *Proof of History: How Solana brings time to crypto*. Accessed December 5, 2024. 2021. URL: https://solana.com/fr/news/proof-of-history.

[51] Akshay S Gaikwad. “Overview of blockchain”. In: *International Journal for Research in Applied Science and Engineering Technology* 8.6 (2020), pp. 2268–2270.

[52] Roel Gevaers, Eddy Van de Voorde, Thierry Vanelslander, et al. “Characteristics of innovations in last-mile logistics-using best practices, case studies and making the link with green and sustainable logistics”. In: *Association for European Transport and contributors* 1 (2009), p. 21.

[53] Roel Gevaers, Eddy Van de Voorde, and Thierry Vanelslander. “Cost Modelling and Simulation of Last-mile Characteristics in an Innovative B2C Supply Chain Environment with Implications on Urban Areas and Cities”. In: *Procedia - Social and Behavioral Sciences* 125 (2014). Presented at the 8th International Conference on City Logistics, pp. 398–411. DOI: 10.1016/j.sbspro.2014.01.1483.

[54] GIScience Research Group, Heidelberg University. *OpenRouteService: Open Source Route Planning*. Accessed: January 15,2025. 2024. URL: https://openrouteservice.org/.

[55] Johannes Göbel et al. “Bitcoin blockchain dynamics: The selfish-mine strategy in the presence of propagation delay”. In: *Performance evaluation* 104 (2016), pp. 23–41.

[56] David Goldschlag, Michael Reed, and Paul Syverson. “Onion routing”. In: *Communications of the ACM* 42.2 (1999), pp. 39–41.

[57] R. Golini et al. “An Assessment Framework to Support Collective Decision Making on Urban Freight Transport”. In: *Transport* 33 (2018), pp. 890–901.

[58] Julija Golosova and Andrejs Romanovs. “The advantages and disadvantages of the blockchain technology”. In: *2018 IEEE 6th workshop on advances in information, electronic and electrical engineering (AIEEE)*. IEEE. 2018, pp. 1–6.

[59] Kent N. Gourdin. *Global Logistics Management: A Competitive Advantage for the 21st Century*. 2nd. Malden, MA, Massachusetts: Wiley-Blackwell, 2006. ISBN: 9781405127134.

[60] Carrefour Group. *Carrefour Bio Blockchain*. https://www.carrefour.com/en/news/2022/carrefourbioblockchain. Accessed on April 7, 2025. 2022.

[61] Francesca Guerriero et al. “A Blockchain-Based System for the Last-Mile Delivery”. In: *Proceedings of the 12th International Conference on Operations Research and Enterprise Systems (ICORES 2023)*. SCITEPRESS–Science and Technology Publications, Lda. 2023, pp. 237–244. DOI: 10.5220/0011886200003396.

[62] Huaqun Guo and Xingjie Yu. “A survey on blockchain technology and its security”. In: *Blockchain: research and applications* 3.2 (2022), p. 100067.

[63] Niels Hackius and Moritz Petersen. “Blockchain in logistics and supply chain: trick or treat?” In: *Digitalization in Supply Chain Management and Logistics: Smart and Digital Solutions for an Industry 4.0 Environment. Proceedings of the Hamburg International Conference of Logistics (HICL), Vol. 23*. Berlin: epubli GmbH. 2017, pp. 3–18.

[64] S. Halzack. *Amazon Flex, the retailer’s Uber-like effort to bring you packages*. Washington Post. 2015. URL: https://www.washingtonpost.com/news/business/wp/2015/09/29/amazon-flex-the-retailers-uber-like-effort-to-bring-you-packages/?noredirect=on (visited on 02/26/2025).

[65] Haya R. Hasan and Khaled Salah. “Blockchain-Based Proof of Delivery of Physical Assets With Single and Multiple Transporters”. In: *IEEE Access* 6 (2018), pp. 46781–46792. DOI: 10.1109/ACCESS.2018.2866512.

[66] Ryan Henry, Amir Herzberg, and Aniket Kate. “Blockchain access privacy: Challenges and directions”. In: *IEEE Security & Privacy* 16.4 (2018), pp. 38–45.

[67] Frank Hofmann et al. “The immutability concept of blockchains and benefits of early standardization”. In: *2017 ITU Kaleidoscope: Challenges for a Data-Driven Society (ITU K)*. 2017, pp. 1–8. DOI: 10.23919/ITU-WT.2017.8247004.

[68] Marko Hribernik et al. “City logistics: Towards a blockchain decision framework for collaborative parcel deliveries in micro-hubs”. In: *Transportation Research Interdisciplinary Perspectives* 8 (2020), p. 100274. DOI: https://doi.org/10.1016/j.trip.2020.100274.

[69] Hyperledger Fabric Documentation Team. *Hyperledger Fabric: Key Concepts and Architecture*. https://hyperledger-fabric.readthedocs.io/en/latest/blockchain.html. Accessed on October 31, 2025. 2025.

[70] IBM. *What are Smart Contracts on Blockchain?* https://www.ibm.com/topics/smart-contracts. Accessed on January 5, 2025. 2023.

[71] IBM and Maersk. *Maersk and IBM Unveil First Industry-Wide Cross-Border Supply Chain Solution on Blockchain*. https://www.prnewswire.com/news-releases/maersk-and-ibm-unveil-first-industry-wide-cross-border-supply-chain-solution-on-blockchain-300418039.html. Accessed on April 7, 2025. 2017.

[72] Mubashar Iqbal and Raimundas Matulevičius. "Exploring sybil and double-spending risks in blockchain systems". In: *IEEE Access* 9 (2021), pp. 76153–76177.

[73] Gwyneth Iredale. *Learn how blockchain truly works, master key definitions, and uncover what makes smart contracts so "smart."* Accessed= March 31,2025. 2021. URL: https://101blockchains.com/smart-contract-programming-languages/.

[74] M. Jahangiriesmaili, S. Bahrami, and M. J. Roorda. "Solution of Two-Echelon Facility Location Problems by Approximation Methods". In: *Transportation Research Record* 2610 (2017), pp. 1–9.

[75] Reza Jazemi et al. "A Review of Literature on Vehicle Routing Problems of Last-Mile Delivery in Urban Areas". In: *Applied Sciences* 13.24 (2023), p. 13015. DOI: 10.3390/app132413015. URL: https://doi.org/10.3390/app132413015.

[76] Marija Jović et al. "A review of blockchain technology implementation in shipping industry". In: *Pomorstvo* 33.2 (2019), pp. 140–148.

[77] S. N. Khan, F. Loukil, C. Ghedira-Guegan, et al. "Blockchain smart contracts: Applications, challenges, and future trends". In: *Peer-to-Peer Networking and Applications* 14 (2021), pp. 2901–2925. DOI: 10.1007/s12083-021-01127-0.

[78] Aggelos Kiayias et al. "Ouroboros: A provably secure proof-of-stake blockchain protocol". In: *Annual international cryptology conference*. Springer. 2017, pp. 357–388.

[79] Henry M Kim and Marek Laskowski. "Toward an ontology-driven blockchain design for supply-chain provenance". In: *Intelligent Systems in Accounting, Finance and Management* 25.1 (2018), pp. 18–27.

[80] Grigorios D. Konstantakopoulos, Sotiris P. Gayialis, and Evripidis P. Kechagias. "Vehicle Routing Problem and Related Algorithms for Logistics Distribution: A Literature Review and Classification". In: *Operational Research* (2020). DOI: 10.1007/s12351-020-00600-7.

[81] P.M. Kornatowski et al. "Last-Centimeter Personal Drone Delivery: Field Deployment and User Interaction". In: *IEEE Robotics and Automation Letters* 3 (2018), pp. 3813–3820.

[82] Ahmed Kosba et al. "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts". In: *2016 IEEE symposium on security and privacy (SP)*. IEEE. 2016, pp. 839–858.

[83] Nacima Labadie, Christian Prins, and Caroline Prodhon. *Metaheuristics for Vehicle Routing Problems*. Volume 3. London, UK and Hoboken, NJ, USA: ISTE Ltd and John Wiley & Sons, Inc., 2016. ISBN: 978-1-84821-811-6.

[84] Lorne Lantz and Daniel Cawrey. *Mastering Blockchain: Unlocking the Power of Cryptocurrencies, Smart Contracts, and Decentralized Applications*. Sebastopol, CA: O'Reilly Media, 2021. ISBN: 978-1-492-05470-2.

[85] Ledger Academy. *Ledger Glossary*. Accessed: April 3, 2025. 2025. URL: https://www.ledger.com/academy/glossary/ledger.

[86] T. Letnik et al. "Dynamic management of loading bays for energy efficient urban freight deliveries". In: *Energy* 159 (2018), pp. 916–928.

[87] K.H. Leung et al. "A B2C e-commerce intelligent system for re-engineering the e-order fulfilment process". In: *Expert Systems with Applications* 91 (2018), pp. 386–401.

[88] Ming Li et al. "CrowdBC: A Blockchain-Based Decentralized Framework for Crowdsourcing". In: *IEEE Transactions on Parallel and Distributed Systems* 30.6 (2019), pp. 1251–1261. DOI: 10.1109/TPDS.2018.2881735.

[89] Ying-Chang Liang and Ying-Chang Liang. "Blockchain for dynamic spectrum management". In: *Dynamic Spectrum Management: From Cognitive Radio to Blockchain and Artificial Intelligence* (2020), pp. 121–146.

[90] Linux Foundation. *Understanding Ledgers – Blockchain: Understanding Its Uses and Implications (LFS170)*. https://trainingportal.linuxfoundation.org/learn/course/blockchain-understanding-its-uses-and-implications-lfs170/blockchain-mechanics/understanding-ledgers?page=1. Accessed: November 15th, 2024. 2024.

[91] Y. Idel Mahjoub, M. Hassoun, and D. Trentesaux. "Blockchain adoption for SMEs: opportunities and challenges". In: *IFAC PapersOnLine* 55.10 (2022), pp. 1834–1839. DOI: 10.1016/j.ifacol.2022.09.665.

[92] Yassine Idel Mahjoub et al. "Supply chain application of blockchain-based solutions for cyber-physical systems: Review and prospects". In: *International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing*. Springer. 2021, pp. 545–558.

[93] L.G. Marujo et al. "Assessing the sustainability of mobile depots: The case of urban freight distribution in Rio de Janeiro". In: *Transportation Research Part D: Transport and Environment* 62 (2018), pp. 256–267.

[94] R. Masa'deh et al. "The Blockchain Effect on Courier Supply Chains Digitalization and Its Contribution to Industry 4.0 within the Circular Economy". In: *Sustainability* 16 (2024), p. 7218. DOI: 10.3390/su16167218.

[95] Masud7866. *Solomon VRPTW Benchmark*. Accessed: January 16,2025. 2022. URL: https://www.kaggle.com/datasets/masud7866/solomon-vrptw-benchmark.

[96] Ahmed Afif Monrat, Olov Schelén, and Karl Andersson. "A Survey of Blockchain From the Perspectives of Applications, Challenges, and Opportunities". In: *IEEE Access* 7 (2019), pp. 117134–117151.

[97] E. Morganti and M. Browne. "Technical and operational obstacles to the adoption of electric vans in France and the UK: An operator perspective". In: *Transport Policy* 63 (2018), pp. 90–97.

[98] Andrés Muñoz-Villamizar et al. "Measuring Disruptions in Last-Mile Delivery Operations". In: *Logistics* 5.1 (2021), p. 17. DOI: 10.3390/logistics5010017.

[99] Kadim Lahcen Nadime et al. “Automating Attended Home Deliveries with Smart Contracts: A Blockchain-based Solution for E-commerce Logistics”. In: *E3S Web of Conferences*. Vol. 469. Presented at ICEGC’2023. EDP Sciences, 2023, p. 00026. DOI: 10.1051/e3sconf/202346900026. URL: https://doi.org/10.1051/e3sconf/202346900026.

[100] Satoshi Nakamoto. “Bitcoin whitepaper”. In: *URL: https://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)* 9 (2008), p. 15.

[101] Jakub Nalepa, ed. *Smart Delivery Systems: Solving Complex Vehicle Routing Problems*. Amsterdam, The Netherlands: Elsevier, 2020. ISBN: 978-0-12-815715-2.

[102] Dung H. Nguyen et al. “What Is the Right Delivery Option for You? Consumer Preferences for Delivery Attributes in Online Retailing”. In: *Journal of Business Logistics* 40.4 (2019), pp. 299–321. DOI: 10.1111/jbl.12210.

[103] Alexander G. Nikolaev and Sheldon H. Jacobson. “Simulated Annealing”. In: *Handbook of Metaheuristics*. Ed. by Michel Gendreau and Jean-Yves Potvin. Vol. 146. International Series in Operations Research & Management Science. New York, NY: Springer, 2010, pp. 1–39. DOI: 10.1007/978-1-4419-1665-5_1.

[104] M Niranjanamurthy, BN Nithya, and SJCC Jagannatha. “Analysis of Blockchain technology: pros, cons and SWOT”. In: *Cluster Computing* 22 (2019), pp. 14743–14757. DOI: 10.1504/IJCSE.2021.115651.

[105] Michael Nofer et al. “Blockchain”. In: *Business & Information Systems Engineering* 59.3 (2017), pp. 183–187. DOI: 10.1007/s12599-017-0467-3.

[106] Karl J O’Dwyer and David Malone. “Bitcoin mining and its energy footprint”. In: *25th IET Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014)*. IET. 2014, pp. 280–285.

[107] John Olsson, Daniel Hellstr"om, and Henrik Pålsson. “Framework of Last Mile Logistics Research: A Systematic Review of the Literature”. In: *Sustainability* 11.24 (2019), p. 7131. DOI: 10.3390/su11247131.

[108] Eric Olszewski. *Why Blockchain Matters To Enterprise (Hint: It’s Not Because Of Decentralization)*. Accessed on April 3, 2025. 2025. URL: https://medium.com/@eolszewski/why-blockchain-matters-to-enterprise-hint-its-not-because-of-decentralization-8c38674f43c6.

[109] Aabha Amey Patil et al. “Blockchain Optimized Routes for Efficient Short Distance Donation Services in Intelligent Urban Mobility”. In: *Proceedings of the 10th International Conference on Advanced Computing and Communication Systems (ICACCS)*. IEEE, 2024, pp. 1164–1169. DOI: 10.1109/ICACCS60874.2024.10716976.

[110] G. Perboli et al. “Simulation–optimisation framework for City Logistics: An application on multimodal last-mile delivery”. In: *IET Intelligent Transport Systems*. Vol. 12. London, UK: Institution of Engineering and Technology, 2018, pp. 262–269.

[111] Guido Perboli and R. Gentile. *GUEST-OR: Linking Lean Business and OR*. https://staff.polito.it/guido.perboli/GUEST-site/docs/2016-euro-guest-or.pdf. Accessed on October 30, 2025. 2016.

[112] Guido Perboli, Stefano Musso, and Mariangela Rosano. "Blockchain in Logistics and Supply Chain: A Lean Approach for Designing Real-World Use Cases". In: *IEEE Access* 6 (2018), pp. 62018–62024. DOI: 10.1109/ACCESS.2018.2875782. URL: http://ieeexplore.ieee.org/document/8629879/.

[113] Daniel Phillips. *How Does Proof-of-History (PoH) Work?* Accessed: December 3, 2024. 2023. URL: https://coinmarketcap.com/academy/article/how-does-proof-of-history-poh-work.

[114] Deepak Puthal et al. "Everything You Wanted to Know About the Blockchain: Its Promise, Components, Processes, and Problems". In: *IEEE Consumer Electronics Magazine* 7.4 (2018), pp. 6–14. DOI: 10.1109/MCE.2018.2816299.

[115] Pethuru Raj, Kavita Saini, and Chellammal Surianarayanan. *Blockchain Technology and Applications*. Boca Raton, FL: CRC Press, 2021. ISBN: 978-0-367-53340-3.

[116] A. Raza, H. Wicaksono, and O. F. Valilai. "Blockchain Technologies for Sustainable Last Mile Delivery: Investigating Customer Awareness and Tendency Using NFT Reward Mechanisms". In: *2023 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)* (2023). DOI: 10.1109/IEEM58616.2023.10406357.

[117] W. J. Rose, J. E. Bell, and S. E. Griffis. "Inductive Research in Last-Mile Delivery Routing: Introducing the Re-Gifting Heuristic". In: *Journal of Business Logistics* 44 (2023), pp. 109–140. DOI: 10.1111/jbl.12318.

[118] J.F. Rougès and B. Montreuil. "Crowdsourcing Delivery: New Interconnected Business Models to Reinvent Delivery". In: *Proceedings of the 1st International Physical Internet Conference*. Québec City, QC, Canada, May 28–30, 2014.

[119] Sana Sabah Sabry, Nada Mahdi Kaittan, and Israa Majeed. "The road to the blockchain technology: Concept and types". In: *Periodicals of Engineering and Natural Sciences (PEN)* 7.4 (2019), pp. 1821–1832.

[120] Fahad Saleh. "Blockchain without waste: Proof-of-stake". In: *The Review of financial studies* 34.3 (2021), pp. 1156–1190.

[121] AR Sathya and Barnali Gupta Banik. "A comprehensive study of blockchain services: future of cryptography". In: *International Journal of Advanced Computer Science and Applications* 11.10 (2020).

[122] Martin Savelsbergh and Marlin W. Ulmer. "Challenges and opportunities in crowdsourced delivery planning and operations—an update". In: *Annals of Operations Research* 343 (2024), pp. 639–661. DOI: 10.1007/s10479-024-06249-1.

[123] Martin Savelsbergh and Tom Van Woensel. "City Logistics: Challenges and Opportunities". In: *Transportation Science* (2016). 50th Anniversary Invited Article—Articles in Advance, pp. 1–12. DOI: 10.1287/trsc.2016.0675. URL: http://dx.doi.org/10.1287/trsc.2016.0675.

[124] *SCOR Diagnostic Tool*. https://www.ascm.org/corporate-solutions/standards-tools/scor-ds. Accessed: March 23,2025.

[125] Stefan Seebacher and Ronny Schüritz. "Blockchain technology as an enabler of service systems: A structured literature review". In: *Exploring Services Science: 8th International Conference, IESS 2017, Rome, Italy, May 24-26, 2017, Proceedings 8*. Springer. 2017, pp. 12–23.

[126] Ship Mercury. *First Mile Logistics*. Accessed: March 22, 2025. URL: https://www.shipmercury.com/glossary-faq/first-mile-logistics.

[127] Marius M. Solomon. "Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints". In: *Operations Research* 35.2 (1987), pp. 254–265. DOI: 10.1287/opre.35.2.254. URL: https://www.jstor.org/stable/170697.

[128] Nick Szabo. "Formalizing and securing relationships on public networks". In: *First monday* (1997).

[129] Eiichi Taniguchi, Russell G Thompson, and Tadashi Yamada. "Emerging techniques for enhancing the practical application of city logistics models". In: *Procedia-Social and Behavioral Sciences* 39 (2012), pp. 3–18.

[130] Eiichi Taniguchi et al. "Modelling city logistics". In: *City logistics*. Emerald Group Publishing Limited, 2001, pp. 17–47.

[131] Don Tapscott and Alex Tapscott. *Blockchain revolution: how the technology behind bitcoin is changing money, business, and the world*. Penguin, 2016.

[132] Vikas Thakur et al. "Land records on Blockchain for implementation of land titling in India". In: *International Journal of Information Management* 52 (2020), p. 101940. DOI: 10.1016/j.ijinfomgt.2020.101940.

[133] Christian Tilk, Katharina Olkis, and Stefan Irnich. "The Last-Mile Vehicle Routing Problem with Delivery Options". In: *OR Spectrum* 43 (2021), pp. 877–904. DOI: 10.1007/s00291-021-00633-0.

[134] F. Torres, M. Gendreau, and W. Rei. "Crowdshipping: An Open VRP Variant with Stochastic Destinations". In: *Transportation Research Part C: Emerging Technologies* 140 (2022), p. 103677. DOI: 10.1016/j.trc.2022.103677.

[135] Paolo Toth and Daniele Vigo, eds. *Vehicle Routing: Problems, Methods, and Applications*. 2nd. Philadelphia: Society for Industrial, Applied Mathematics, and the Mathematical Optimization Society, 2014. ISBN: 9781611973587.

[136] Nikhil Vadgama, Jiahua Xu, and Paolo Tasca, eds. *Enabling the Internet of Value: How Blockchain Connects Global Businesses*. Future of Business and Finance. Cham, Switzerland: Springer Nature Switzerland AG, 2022. ISBN: 978-3-030-78183-5. DOI: 10.1007/978-3-030-78184-2.

[137] Brecht Verheye. "Land Registration in the Twenty-First Century: Blockchain Land Registers from a Civil Law Perspective". In: *Disruptive Technology, Legal Innovation, and the Future of Real Estate*. Ed. by Amnon Lehavi and Reut Levine-Schnur. Springer, Cham, 2020. DOI: 10.1007/978-3-030-52387-9_7.

[138] A. Verma. "Electric vehicle routing problem with time windows, recharging stations and battery swapping stations". In: *EURO Journal of Transportation and Logistics* 7 (2018), pp. 415–451.

[139] Dejan Vujičić, Dijana Jagodić, and Siniša Ranđić. "Blockchain Technology, Bitcoin, and Ethereum: a Brief Overview". In: *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*. East Sarajevo, Bosnia and Herzegovina: IEEE, 2018. DOI: 10.1109/INFOTEH.2018.8345547. URL: https://ieeexplore.ieee.org/document/8345547.

[140] Marko Vukolić. "The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication". In: *Open Problems in Network Security*. Ed. by Jan Camenisch and Doğan Kesdoğan. Springer International Publishing, 2016, pp. 112–125. ISBN: 978-3-319-39028-4.

[141] S. Vyas et al., eds. *Blockchain Technology: Exploring Opportunities, Challenges, and Applications*. 1st. CRC Press, 2022. DOI: 10.1201/9781003138082.

[142] Xuping Wang et al. "How to Choose "Last Mile" Delivery Modes for E-Fulfillment". In: *Mathematical Problems in Engineering* 2014 (2014), Article ID 417129, 11 pages. DOI: 10.1155/2014/417129. URL: http://dx.doi.org/10.1155/2014/417129.

[143] C. Donald J. Waters. *Logistics: An Introduction to Supply Chain Management*. Houndmills, Basingstoke, Hampshire; New York, N.Y.: Palgrave Macmillan, 2003. ISBN: 0-333-96369-5.

[144] D. Weatherspoon and A. Ross. "Designing the Last Mile of the Supply Chain in Africa: Firm Expansion and Managerial Inferences from a Grocer Model of Location Decisions". In: *International Food and Agribusiness Management Review* 11 (2008), pp. 1–16.

[145] Mark Weiser, Rich Gold, and John Seely Brown. "The origins of ubiquitous computing research at PARC in the late 1980s". In: *IBM systems journal* 38.4 (1999), pp. 693–696.

[146] Matthias Winkenbach and Milena Janjevic. "Classification of last-mile delivery models for e-commerce distribution: A global perspective". In: *City logistics 1: New opportunities and challenges* (2018), pp. 209–229.

[147] Karl Wüst and Arthur Gervais. "Do You Need a Blockchain?" In: *Proceedings of IEEE European Symposium on Security and Privacy (EuroS&P)* (2018), pp. 133–157. DOI: 10.1109/EuroSP.2018.00014.

[148] Anatoly Yakovenko. *Solana: A New Architecture for a High Performance Blockchain v0.8.14 White paper*. Tech. rep. Available online at https://solana.io/. Solana, 2018.

[149] Xiaomei Yu et al. "Solving Low-Carbon Last Mile Delivery Problem Using Discrete Marine Predators Algorithm". In: *Applied Soft Computing* 165 (2024), p. 112112. DOI: 10.1016/j.asoc.2024.112112.

[150] Sheping Zhai et al. "Research on the Application of Cryptography on the Blockchain". In: *Journal of Physics: Conference Series*. Vol. 1168. IOP Publishing. 2019, p. 032077.

[151] Haifei Zhang et al. "Review of vehicle routing problems: Models, classification and solving algorithms". In: *Archives of Computational Methods in Engineering* (2022), pp. 1–27.

[152] Peilin Zheng et al. “A detailed and real-time performance monitoring framework for blockchain systems”. In: *Proceedings of the 40th international conference on software engineering: software engineering in practice*. 2018, pp. 134–143.

[153] Zibin Zheng et al. “An overview of blockchain technology: Architecture, consensus, and future trends”. In: *2017 IEEE international congress on big data (BigData congress)*. Ieee. 2017, pp. 557–564.

[154] Zibin Zheng et al. “Blockchain Challenges and Opportunities: A Survey”. In: *International Journal of Web and Grid Services* 14.4 (2018), pp. 352–375. DOI: 10.1504/IJWGS.2018.095647.

[155] L. Zhou et al. “Location-routing Problem with Simultaneous Home Delivery and Customer’s Pickup for City Distribution of Online Shopping Purchases”. In: *Sustainability* 8 (2016), p. 828.

[156] L. Zhou et al. “Model and Algorithm for Bilevel Multisized Terminal Location-routing Problem for the Last Mile Delivery”. In: *International Transactions in Operational Research* 26 (2019), pp. 131–156.

[157] Liehuang Zhu, Keke Gai, and Meng Li. *Blockchain Technology in Internet of Things*. Springer, 2019. DOI: 10.1007/978-3-030-21766-2.