UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Universitat Politècnica de València

Department of Applied Statistics and Operational Research, and Quality

Doctoral Thesis

Doctoral Programme in Statistics and Optimization

Transport Analytics in Sustainable Logistics: from Agile Optimization to Sim-Learnheuristic

---

A thesis submitted in fulfillment of the requirements for the degrees of

Doctor of Philosophy in Statistics and Optimization

Author:
Mohammad Peyman

Thesis supervisors:
Prof. Dr. Javier Panadero
Prof. Dr. Angel A. Juan

July, 2025

"Try not to resist the changes that come your way. Instead, let life live through you. And do not worry that your life is turning upside down. How do you know that the side you are used to is better than the one to come?"

— Rumi

Acknowledgments

Although my name appears as the author of this thesis, I would not have made it this far without the support of many people who have been by my side, both personally and academically. First and foremost, my deepest gratitude goes to my parents, Mrs. Maryam and Mr. Hamid, and my sister, Dra. Mona. From the very beginning, you have been my unwavering support system. Your encouragement and belief in me have been the driving force behind all my achievements. Your love, sacrifices, and dedication throughout the years have sustained me through every challenge and moment of doubt. I can never fully express how much you mean to me, and this achievement belongs as much to you as it does to me. Thank you for being my constant source of strength.

To my supervisors, Prof. Dr. Àngel Juan and Prof. Dr. Javier Panadero: I am deeply thankful for the opportunity you gave me to be part of such an inspiring and innovative group. Under your guidance, I have gained not only profound knowledge in operations research and machine learning but also invaluable insights into the academic world—how to publish, collaborate, and push the boundaries of research. Beyond the technical and intellectual growth, you have taught me the significance of teamwork, where each member's expertise enriches the collective effort. Your mentorship has shaped me both as a researcher and as a collaborator, and I will always carry the lessons I have learned from you.

To my friends and colleagues, especially Dra. Sara, whose unwavering support and advice carried me through the highs and lows of this journey—thank you for your constant encouragement, wisdom, and the countless conversations that helped me navigate both academic and personal challenges. To Xabi, Veronika, Rachel, and Patricia: your friendship and the joy of the moments we have shared have been invaluable. To my dearest Mahsa, who brings peace into my life and has stood by me throughout this process with constant support and deep understanding—thank you. Your presence has brought me balance, comfort, and strength during moments when I needed it most. You have been my anchor on this journey, and for that, I am profoundly grateful. Finally, I want to thank the unseen forces that guide us. It is the quiet strength that pushes you to grow, not just as an academic but as a person. I am grateful for that connection, which has helped me persevere and grow throughout this journey.

Abstract

Transportation and logistics (T&L) are essential to smart city development, as urbanization accelerates and demand for sustainable solutions grows. T&L systems mitigate transportation's negative impacts (air pollution, noise, greenhouse gas emissions, and energy consumption) while enhancing efficiency and cost-effectiveness. As urban populations and environmental awareness increase, promoting low-emission vehicles like electric vehicles and expanding carsharing and ridesharing are crucial. However, these trends also introduce challenges, such as supply constraints and traffic congestion, straining infrastructure. Addressing these challenges requires integrating advanced technologies like artificial intelligence, internet of things (IoT), big data analysis, and optimization techniques. These challenges often present as NP-hard combinatorial optimization problems, where exact methods are impractical, especially for large-scale, dynamic scenarios demanding quick, high-quality solutions and real-time decision-making. Modeling uncertainty in T&L systems adds complexity, typically addressed by stochastic approaches using probability distributions. Yet, the dynamic nature of real-world problems requires even more advanced solutions. The main objective of this thesis is to develop a model integrating simulation, agile optimization, and machine learning to tackle T&L challenges. The convergence of these technologies is shaping the future of smart, sustainable cities by enabling more accurate and responsive decision-making. In this thesis a layered approach is used to address T&L complexities progressively. The first layer takes into account the analytical approach to the IoT data. The second layer introduces heuristics, biased-randomized heuristics, metaheuristics, and agile optimization to solve deterministic T&L problems. The third layer incorporates Monte Carlo simulations to account for stochastic variables, enhancing robustness. The forth layer integrates discrete event simulation with Python, providing a realistic representation of uncertainties. Finally, sim-learnheuristic combine simheuristics with machine learning to address both stochastic and dynamic uncertainties. Numerical experiments with real-world and benchmark instances validate the proposed algorithms, demonstrating their cost and time-efficiency and reliability in dynamic, uncertain, and constrained T&L scenarios.

Resumen

El transporte y la logística (T&L) son esenciales para el desarrollo de ciudades inteligentes, a medida que la urbanización se acelera y la demanda de soluciones sostenibles crece. Los sistemas de T&L mitigan los impactos negativos del transporte (contaminación del aire, ruido, emisiones de gases de efecto invernadero y consumo de energía) al tiempo que mejoran la eficiencia y la rentabilidad. A medida que las poblaciones urbanas y la conciencia ambiental aumentan, promover vehículos de bajas emisiones, como los vehículos eléctricos, y expandir el uso compartido de automóviles y servicios de transporte compartido se vuelve crucial. Sin embargo, estas tendencias también introducen desafíos, como las limitaciones de suministro y la congestión del tráfico, que sobrecargan la infraestructura. Abordar estos desafíos requiere la integración de tecnologías avanzadas, como inteligencia artificial, internet de las cosas (IoT), análisis de big data y técnicas de optimización. Estos desafíos a menudo se presentan como problemas de optimización combinatoria NP-difíciles, donde los métodos exactos son poco prácticos, especialmente para escenarios a gran escala y dinámicos que exigen soluciones rápidas, de alta calidad y toma de decisiones en tiempo real. Modelar la incertidumbre en los sistemas de T&L añade complejidad, generalmente abordada mediante enfoques estocásticos utilizando distribuciones de probabilidad. Sin embargo, la naturaleza dinámica de los problemas del mundo real requiere soluciones aún más avanzadas. El objetivo principal de esta tesis es desarrollar un modelo que integre simulación, optimización ágil y aprendizaje automático para abordar los desafíos de T&L. La convergencia de estas tecnologías está moldeando el futuro de las ciudades inteligentes y sostenibles al permitir una toma de decisiones más precisa y receptiva. En esta tesis se utiliza un enfoque en capas para abordar las complejidades de T&L de manera progresiva. La primera capa tiene en cuenta el enfoque analítico de los datos del IoT. La segunda capa introduce heurísticas, heurísticas sesgadas aleatorizadas, metaheurísticas y optimización ágil para resolver problemas deterministas de T&L. La tercera capa incorpora simulaciones de Monte Carlo para tener en cuenta variables estocásticas, mejorando la robustez. La cuarta capa integra simulación de eventos discretos con Python, proporcionando una representación realista de las incertidumbres. Finalmente, la sim-learnheuristic combina simheurísticas con aprendizaje automático para abordar tanto las incertidumbres estocásticas como dinámicas. Los

experimentos numéricos con instancias del mundo real y de referencia validan los algoritmos propuestos, demostrando su eficiencia en costos, tiempo y confiabilidad en escenarios de T&L dinámicos, inciertos y con restricciones.

Resum

El transport i la logística (T&L) són essencials per al desenvolupament de ciutats intel · ligents, a mesura que l'urbanització s'accelera i la demanda de solucions sostenibles creix. Els sistemes de T&L mitiguen els impactes negatius del transport (contaminació de l'aire, soroll, emissions de gasos d'efecte hivernacle i consum d'energia) al mateix temps que milloren l'eficiència i la rendibilitat. A mesura que les poblacions urbanes i la consciència mediambiental augmenten, promoure vehicles de baixes emissions, com els vehicles elèctrics, i expandir l'ús compartit de vehicles i els serveis de transport compartit esdevé crucial. No obstant això, aquestes tendències també introdueixen desafiaments, com ara les limitacions de subministrament i la congestió del trànsit, que sobrecarreguen la infraestructura. Enfrontar aquests desafiaments requereix la integració de tecnologies avançades, com ara la intel · ligència artificial, l'internet de les coses (IoT), l'anàlisi de grans volums de dades i les tècniques d'optimització. Aquests desafiaments sovint es presenten com a problemes d'optimització combinatòria NP-difícils, on els mètodes exactes són poc pràctics, especialment per a escenaris de gran escala i dinàmics que exigeixen solucions ràpides, de gran qualitat i presa de decisions en temps real. Modelar la incertesa en els sistemes de T&L afegeix complexitat, que generalment s'aborda mitjançant enfocaments estocàstics que utilitzen distribucions de probabilitat. Tot i això, la naturalesa dinàmica dels problemes del món real requereix solucions encara més avançades. L'objectiu principal d'aquesta tesi és desenvolupar un model que integre simulació, optimització àgil i aprenentatge automàtic per abordar els desafiaments de T&L. La convergència d'aquestes tecnologies està modelant el futur de les ciutats intel · ligents i sostenibles, permetent una presa de decisions més precisa i receptiva. En aquesta tesi s'utilitza un enfocament en capes per abordar les complexitats de T&L de manera progressiva. La primera capa té en compte l'enfocament analític de les dades de l'IoT. La segona capa introdueix heurístiques, heurístiques esbiaixades aleatòries, metaheurístiques i optimització àgil per resoldre problemes deterministes de T&L. La tercera capa incorpora simulacions de Montecarlo per tenir en compte variables estocàstiques, millorant la robustesa. La quarta capa integra la simulació d'esdeveniments discrets amb Python, proporcionant una representació realista de les incerteses. Finalment, la sim-learnheuristic combina simheurístiques amb aprenentatge automàtic per abordar tant les incerteses estocàstiques com dinàmiques. Els

experiments numèrics amb instàncies del món real i de referència validen els algoritmes proposats, demostrant la seua eficiència en costos, temps i fiabilitat en escenaris de T&L dinàmics, incerts i amb restriccions.

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

AGVs	Automated Guided Vehicles.
AI	Artificial Intelligence.
AIC	Akaike Information Criterion.
AL	AnyLogic.
ANOVA	Analysis of Variance.
ARIMA	AutoRegressive Integrated Moving Average.
AUC	Area Under the Curve.
BFS	Best Found Solution.
BIC	Bayesian Information Criterion.
BR	Biased-Randomized.
COVID-19	Coronavirus Disease 2019.
DES	Discrete-Event Simulation.
DRSP	Dynamic Ridesharing Problem.
DTOP-MV	Dynamic Team Orienteering Problem with Mandatory Visit.
EVs	Electric Vehicles.
FIFO	First-in, First-out.

HVPT	High Variability in Processing Times.
HVTA	High Variability in Task Assignment.
I2I	Infrastructure-to-Infrastructure.
ILS	Iterated Local Search.
IoT	Internet of Things.
ITS	Intelligent Transportation System.
LLRF	Less Loaded Resource First.
LPTF	Longest Processing Time First.
LRT	Longest-Route Time.
LVPT	Low Variability in Processing Times.
LVTA	Low Variability in Task Assignment.
MCS	Monte Carlo simulation.
ME	Mean Error.
ML	Machine Learning.
MLRF	Most Loaded Resource First.
MRTA	Multi-Robot Task Allocation.
MS	Multi-Start.
OBD	Best Deterministic Solution.
OBF	Average FIFO Solutions.
OBN	Average Non-Reactive Solutions.
OBR	Average Reactive Solutions.
OBS	Best Stochastic Solution.
OBUs	On-board Units.
OrV	Over to Rest.
OSRM	Open Source Routing Machine.
OVRP	Open Vehicle Routing Problem.
PFS	Picking Frequency Based Storage Strategy.
PPE	Personal Protective Equipment.
RCSP	Resource-Constrained Scheduling Problem.
RMSE	Root-Mean-Square Error.

ROC	Receiver Operating Characteristic.
RSP	Ridesharing Problem.
RSUs	Roadside Units.
RTSP	Robotic Task Sequencing Problem.
SHAP	Shapley Additive Explanations.
SLAP	Storage Allocation Problem.
SPTF	Shortest Processing Time First.
STC	Space Time Cube.
T&L	Transportation and Logistics.
T&M	Transport and Mobility.
TOP	Team Orienteering Problem.
TOP-MV	Team Orienteering Problem with Mandatory Visit.
TOPTW-MV	Team Orienteering Problem Time Window with Mandatory Visit.
UAVs	Unmanned Aerial Vehicles.
UN	United Nations.
V2I	Vehicle-to-Infrastructure.
V2V	Vehicle-to-Vehicle.
V2X	Vehicle-to-Everything.
VANETs	Vehicular Ad Hoc Networks.
VH	Very-High.
VL	Very-Low.
VRP	Vehicle Routing Problem.
XGBoost	eXtreme Gradient Boosting.

Chapter 1

Introduction

1.1 Motivation

Transportation and Logistics (T&L) are crucial for business operations across various industries, acting as the backbone of the global economy. T&L ensures the smooth flow of goods, services, and information, facilitating trade and commerce. While transportation focuses on the movement of humans, animals, and goods from one place to another, logistics involves the broader process of managing the delivery of products to carriers on time, in accurate quantities, and in good condition. This process encompasses all stages of a product's life cycle, from manufacturing to final delivery to the consumer (Speranza, 2018). In today's interconnected and globalized market, the importance of T&L cannot be overstated and the effective logistics management is essential for optimizing supply chains, reducing costs, and improving customer satisfaction.

Over the last fifty years, urbanization has become recognized as a major global trend, marked by rapid city growth and a significant increase in the number of people living in cities. The United Nations (UN) estimates that around three million people move from rural areas to cities each week, suggesting that urban populations could nearly double by 2050. As a result, it is projected that by then, 70% of the world's population will reside in cities (Freudenberg et al., 2017). In 2022, an average of 57% of the global population lived in cities (Demographia, 2022). In the european union, the urban population has reached approximately 75%, while in North America, 82% of the population resides in cities (Urbanet, 2023). Urban areas consume between 60% and 80% of the world's energy, primarily for electricity and transportation, and they have a significant environmental impact due to greenhouse gas emissions. This rapid urban growth has led to several major challenges, such as traffic congestion, pollution, increased waste generation, and rising living costs. These problems highlight the urgent need for sustainable urban development that harmonizes economic growth with environmental protection while improving the quality of life for urban residents. (Camero et al., 2019).

The concept of a smart city has emerged as a promising solution to tackle these urban challenges. By using advanced technology, smart cities aim to improve the quality of life in urban environments by enhancing environmental sustainability, delivering superior services to residents, and promoting sustainable transportation (Yigitcanlar et al., 2022). It responds to the evolving demands driven by economic, technological, and cultural developments, influencing various sectors such as the economy, education, management, and particularly T&L (Russo et al., 2016). Sustainable logistics, a crucial element of smart cities, aims to implement supply chain and transportation practices that are environmentally friendly, socially responsible, and economically sustainable (Matusiewicz et al., 2019). This approach addresses customer demands for high-quality goods and convenient delivery while minimizing environmental impact. Sustainable logistics is achieved through the adoption of green transport technologies, low-emission solutions, and circular economy strategies for packaging and storage. These practices are further enhanced by information and communication technologies and the integration of smart logistics concepts (Rashidi et al., 2019). Existing urban freight systems predominantly rely on fossil fuels, accounting for 10% to 15% of the equivalent vehicle miles traveled in urban areas (He and Bhatti, 2024). Urban freight is responsible for over 25% of CO2 emissions in cities (Crippa et al., 2021), making the reduction of its negative impact essential for the development of smart cities. Issues like congestion, air pollution, and traffic accidents significantly affect sustainability and the post-pandemic recovery of cities (Hörcher et al., 2022). The development of sustainable and smart urban transportation solutions is closely aligned with the UN sustainable development goals (Halkos et al., 2021). As cities evolve towards becoming smarter and more sustainable, there is a growing recognition on the importance of increasing the utilization of low-emission vehicles, such as Electric Vehicles (EVs), and expanding new mobility options like carsharing and ridesharing, which have become essential components of T&L systems (Wesseling et al., 2014; Prieto et al., 2017). The smart city concept empowers policymakers by bridging gaps in urban data availability, facilitating improved decision-making through the collection and analysis of real-world data via sensors, actuators, meters, and personal mobile devices (Osmólski et al., 2019). As a result, smart cities are typically defined as instrumented, interconnected, and intelligent (Harrison et al., 2010). In recent decades, the rapid development and widespread use of sensing, computing, and communication devices have revolutionized the field of T&L. The Internet of Things (IoT) has been instrumental in this transformation, tackling persistent issues like traffic congestion, accident detection, fare collection, and limited parking availability. (Dobrovnik et al., 2018). IoT consists of a vast network of interconnected devices, both digital and mechanical, that possess unique identifiers and the capability to transfer data over the internet. These devices gather and transmit data, facilitating smarter and more efficient T&L operations by connecting the various elements and helping to address the challenges within the

T&L sector.

Many of the challenges mentioned earlier, especially those encountered at the tactical, strategic, and operational levels within the supply chain, can be modeled as combinatorial optimization problems. Scientific literature has demonstrated that most realistic T&L challenges are NP-hard, meaning they are computationally complex, with the number of potential solutions increasing exponentially as the problem size grows linearly (Nagy et al., 2007). This complexity makes it exceptionally difficult to find optimal solutions (whether for cost reduction or profit maximization) within limited computing times, particularly when dealing with medium to large problem instances, even with the most advanced computers available today. In practice, decision-makers often prefer a good, non-optimal solution that can be reached quickly. As a result, approximate methods like heuristics, metaheuristics, and simheuristics (which combine optimization algorithms with simulation methods (e.g., Monte Carlo simulation (MCS), Discrete-Event Simulation (DES))) are widely used. Additionally, the integration of metaheuristics and/or heuristics with Machine Learning (ML) methods, known as learnheuristics, has proven effective in dealing with non-static inputs and generating high-quality solutions for a wide range of T&L problems (Juan, Pascual, et al., 2015). Despite these advancements, there remains a gap in fully addressing the challenges posed by the dynamic T&L environment, indicating a need for continued innovation and development in this field. In the context of real-life optimization problems, where conditions are often dynamic and uncertain, smart and agile real-time decision-making is essential. These dynamics and uncertainties introduce a range of constraints and challenges that significantly increase the complexity of T&L operations. Key examples of these constraints include demand variability (Chen, Hammad, et al., 2024), weather conditions (Pińskwar et al., 2024), and traffic congestion (Olaniyi et al., 2024). Additionally, specific issues related to EVs further complicate matters, such as battery performance, the limitations of energy hubs (Carlton et al., 2022), grid dependency, energy availability (Jochem et al., 2015), and charging station congestion (Hall et al., 2017). To effectively address these challenges, it is evident that some real-life problems demand hybrid approaches capable of handling both stochastic and dynamic conditions simultaneously. This thesis proposes multiple methods tailored to solve various challenges mentioned earlier. The proposed methods are approximate and use data aggregated from IoT devices to apply IoT analytics for identifying and predicting mobility patterns. The predictive models are based on ML techniques, enabling more accurate and responsive solutions to these complex problems.

Uncertainty and dynamic conditions are inherent in all real-world T&L activities. However, incorporating these factors into models often significantly increases their complexity (Simangunsong et al., 2012). As a result, dynamic conditions are frequently simplified or ignored to make mathematical models and solution approaches more manageable. For instance, basic heuristic

and metaheuristic algorithms typically do not account for uncertainty. A common approach to modeling uncertainty or dynamism is through probability distributions, known as a stochastic approach. This method is effective when sufficient data is available to accurately estimate these distributions. However, this requires comprehensive input analysis, which depends on sufficient and high-quality data. When such data is lacking or unreliable (Corlu et al., 2020), the stochastic approach may fall short. In these cases, DES software offers a powerful alternative. DES software can closely mimic real-world scenarios and, when combined with optimization algorithms, provides a flexible and realistic way to manage uncertainties (Rabe et al., 2020). Instead of relying on probability calculations for events like customer demand, traffic levels, or weather conditions, DES software models these dynamics directly, offering more practical insights. Moreover, since real-world conditions are constantly changing, ML algorithms are employed to predict and adapt to these changes. When integrated into the simheuristic framework, these predictive capabilities enhance the ability to tackle complex, dynamic problems effectively. This combination allows for more accurate, adaptable decision-making, better reflecting the challenges of T&L operations. Figure 1.1 provides an overview of the key components considered in this thesis. The process begins with data collection through IoT devices, including sensors and cameras, which is then integrated with open data sources. This integration enables the creation of innovative services, such as real-time traffic monitoring, mobility tracking, and road condition assessments. IoT analytics plays a crucial role in the T&L system, managing the vast amounts of data, converting it into actionable insights, developing predictive models, and facilitating intelligent decision-making. To further enhance these capabilities, different optimization algorithms are employed. These optimization algorithms address various aspects of the problem, starting with deterministic methods that provide a structured approach to decision-making. However, to introduce flexibility and explore a broader range of potential solutions, Biased-Randomized (BR) heuristics are incorporated (Grasas et al., 2017). These techniques add an element of controlled randomness to the deterministic algorithms, allowing the system to escape local optima and identify better solutions without losing the fundamental logic of the original methods. Building on this foundation, agile optimization is used as a powerful approach that combines BR heuristics with parallel computing. This method transforms traditional deterministic algorithms into probabilistic ones, enabling the system to adapt and respond effectively in real-time scenarios. Agile optimization not only enhances the decision-making process by increasing the diversity of explored solutions but also uses the computational power of parallel processing to handle the complexity and scale of the data involved.

Real-world scenarios are often shaped by uncertainty, making it essential to incorporate stochastic elements in their analysis. To address this, simulation techniques use statistical methods to model the likelihood of various outcomes in uncertain processes. Simulation is

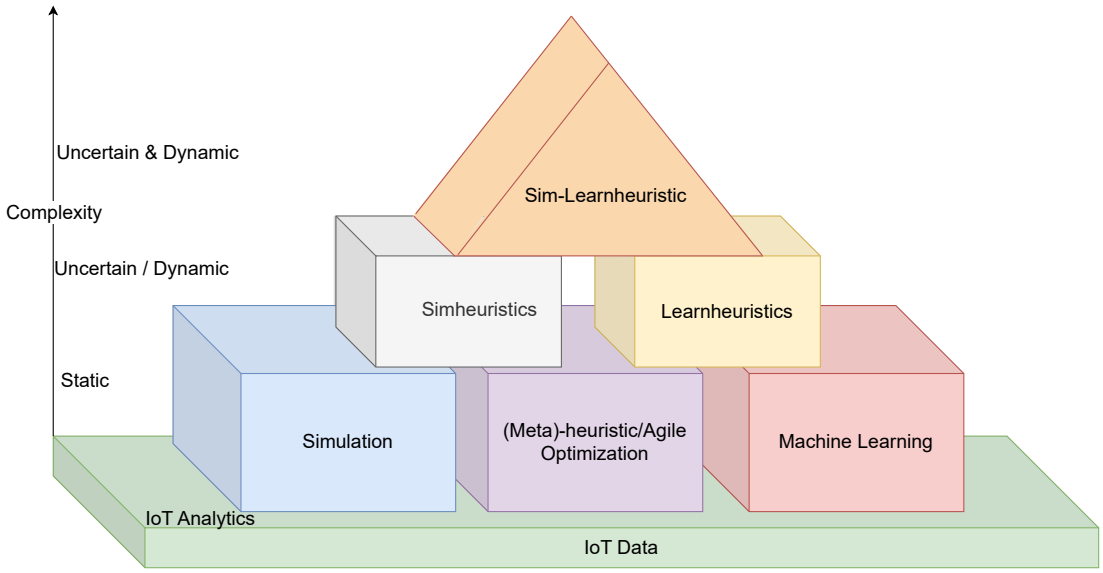


Figure 1.1: Main components addressed in this thesis.

particularly useful for studying complex, real-world systems affected by randomness. One widely used approach is DES, which models time-dependent systems. There are numerous software packages available for DES. While these tools can accurately model real-life systems, they typically lack advanced capabilities for conducting sophisticated analytics, such as ML or interactive visualizations, and are often limited when it comes to implementing complex optimization techniques. In this context, MCS is employed to effectively capture and manage these uncertainties, offering a robust framework for anticipating a wide range of potential scenarios. This approach creates the foundation for more advanced procedures, leading to the concept of simheuristics (a powerful algorithm that combines metaheuristics with MCS) to efficiently tackle NP-hard problems involving stochastic parameters. This approach is particularly effective when there is sufficient and reliable data to accurately estimate the underlying probability distributions. However, this method relies heavily on thorough input analysis which in turn depends on the availability of high-quality data. When data is limited or unreliable, the stochastic approach may not perform as expected. In such situations, DES software presents a powerful alternative. DES software can closely replicate real-world processes and, when integrated with optimization algorithms, provides a flexible and realistic method for managing uncertainties. So that this thesis provides a comprehensive guide on integrating various simulation software with programming languages like Python, and on validating these models to ensure accuracy and reliability. Also, another key component in solving the problem is ML techniques. However, to enhance its effectiveness, we combine it with optimization algorithms. While ML specializes at identifying patterns, learning from data, and making predictions, it

does not inherently provide optimal solutions. By integrating ML and optimization, we improve decision-making by not only predicting outcomes but also identifying the best solutions within given system constraints. This combination, known as the "learnheuristic" approach, combines the predictive capacity of ML with the precision and efficiency of optimization to effectively address complex problems. This approach harnesses the predictive capabilities of ML to inform and guide the optimization process, allowing it to explore the solution space more efficiently and effectively, thereby addressing complex, uncertain, and dynamic problems. Additionally, this thesis introduces a novel "sim-learnheuristic" concept specifically designed to address the complexities of dynamic problem-solving. This innovative framework combines ML algorithms, which predict and adapt to real-world dynamic conditions, with a simulation component based on MCS to handle uncertainty. By integrating these elements, the sim-learnheuristic offers a more robust and effective approach to tackling dynamic challenges. Given the increasing demand for addressing real-world challenges from both social and economic perspectives, this thesis seeks to develop an innovative methodology by integrating advanced technological components.

Figure 1.2 presents the high-level structure of this thesis, outlining the key methodologies and solution approaches, including transport analytics, various optimization techniques such as heuristics, metaheuristics, agile optimization, simheuristics, and sim-learnheuristic. The structure highlights how different methodologies are applied to specific problems. Transportation analysis is coupled with IoT analytics to enhance system monitoring and decision-making. Dynamic task scheduling is addressed using a heuristic algorithm. The Dynamic Ridesharing Problem (DRSP) problem is solved using a metaheuristic, while an agile BR heuristic combined with a planning horizon tackles the Dynamic Team Orienteering Problem with Mandatory Visit (DTOP-MV). Moreover, by integrating statistical methods such as MCS with heuristics, the Open Vehicle Routing Problem (OVRP) is solved more effectively. The integration between Python and various DES software is also explored and evaluated, facilitating the resolution of complex problems like the Team Orienteering Problem (TOP), Storage Allocation Problem (SLAP), and task sequencing under uncertainty. Finally, the thesis introduces the novel sim-learnheuristic methodology, which merges a savings-based heuristic algorithm with MCS and multiple regression models. This integration allows for the consideration of stochastic and dynamic elements, enabling the approach to capture the effects of randomness and variability. The sim-learnheuristic methodology proves effective in producing high-quality solutions in scenarios involving uncertainty, offering a robust framework for simultaneously handling both stochastic and dynamic components.

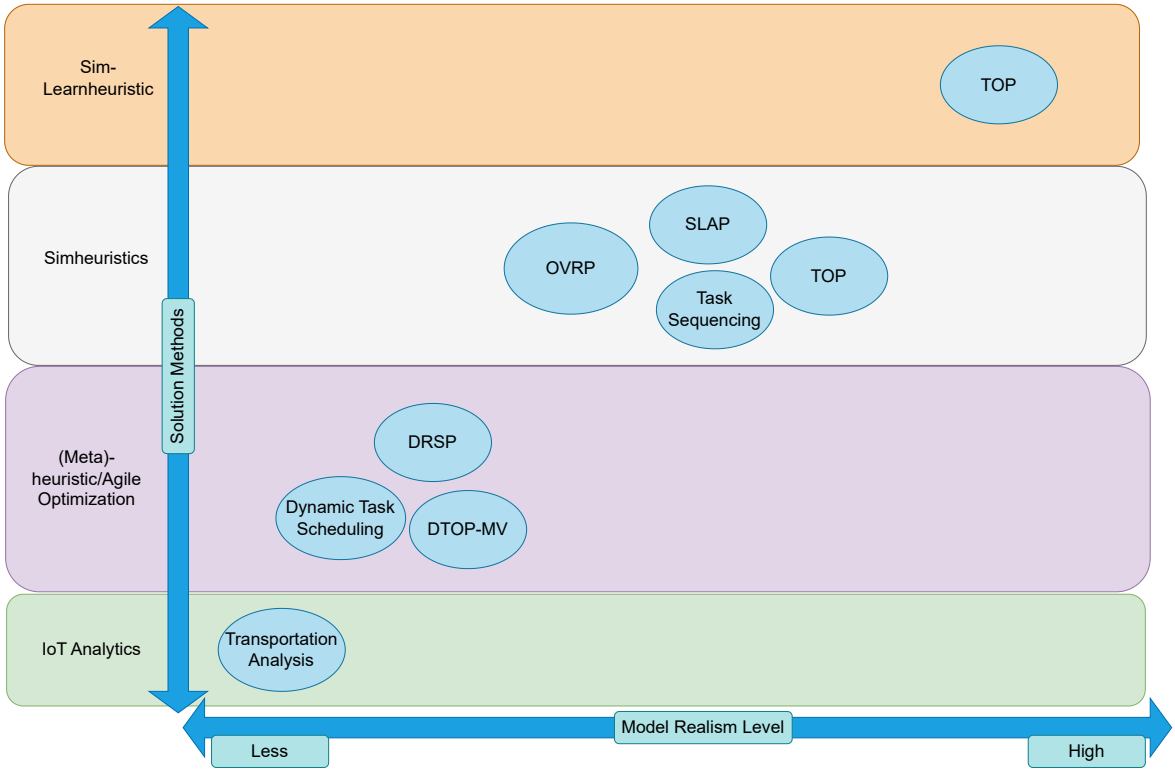


Figure 1.2: Layered structure of problem and developed solution methodologies.

1.2 Objectives and Original Contributions

The primary aim of this thesis is to propose and develop innovative methodologies that integrate transport analytics, the IoT, agile optimization algorithms, ML, and simulation to effectively address T&L challenges in smart and sustainable cities. Building on this core objective, the following original contributions and research findings have been achieved:

- Analyzed urban traffic patterns by integrating IoT data with open city datasets, utilizing exploratory visualization techniques, time series analysis, and ML methods to develop and evaluate predictive models.
- Effective solution algorithms, using agile optimization that enables the system to adapt and respond effectively in real-time scenarios through BR heuristics, and metaheuristics, are proposed and developed to address the identified optimization challenges. Additionally, simheuristics are employed to handle uncertain or dynamic scenarios, ensuring robust solutions for complex optimization problems.
- A robust integration between Python and various simulation software platforms is successfully implemented, enabling the smooth combining of optimization methods with simulation tools. for more accurate and efficient problem-solving.
- For the first time, a novel approach, termed "sim-learnheuristic", is introduced to address complex decision-making challenges in T&L systems. It combines simulation to model dynamic and uncertain environments, ML to predict outcomes and extract patterns, and (meta)-heuristics to optimize decision spaces. This innovative framework provides flexible data-driven, and robust solutions, making it suitable for real-life applications.
- A series of computational and numerical experiments are carried out to assess the performance of the established integration and developed algorithms when applied to realistic, large-scale problem instances.
- Finally, managerial insights and conclusions are drawn regarding the potential benefits of using the proposed methodologies and developed algorithms in complex, real-world decision-making processes. As a result, new areas of future research are identified and proposed.

1.3 Outline of the Thesis

This thesis is divided into six parts.

Part I which focuses on key concepts and the current state of the art, is divided into two chapters as follows:

- Chapter 2 reviews concepts and published works on subproblems related to vehicular networks, edge, fog, and cloud computing, as well as IoT analytics combined with ML in transport analytics. It explores data processing, communication protocols, and the use of ML and optimization in Intelligent Transportation System (ITS) and smart cities. The review highlights recent advancements and the potential of these technologies to improve urban mobility, optimize resources, and enhance transportation efficiency.

- Chapter 3 reviews the concepts and published literature on various subproblems related to the broader Vehicle Routing Problem (VRP), including the OVRP, Ridesharing Problem (RSP), and TOP. Additionally, considering the importance of efficient logistics and operations, it explores related works addressing challenges such as the SLAP and task sequencing.

Part II which focuses on agile optimization algorithms and their implementations, is divided into four chapters as outlined below

- Chapter 4 explores the concepts and related work on the agile optimization framework, focusing on BR heuristics, metaheuristics, and agile optimization algorithms.

- Chapter 5 explores the application of transport analytics, with a focus on urban mobility in smart cities like Barcelona, Spain. The chapter utilizes advanced visualization techniques, including heatmaps, to analyze traffic density and patterns across different city sections. It employs time series analysis, specifically AutoRegressive Integrated Moving Average (ARIMA) and eXtreme Gradient Boosting (XGBoost) models, for traffic flow prediction. The chapter demonstrates how real-time data and predictive models can be used to improve urban planning and decision-making, particularly in optimizing infrastructure like electric vehicle charging stations and transportation routes.

- Chapter 6 presents a heuristic-based approach for optimizing task scheduling for Automated Guided Vehicles (AGVs). It introduces a novel reactive methodology involving delay matrices and dynamic scheduling to improve AGVs operations.

- Chapter 7 explores the integration of IoT analytics into agile optimization, applying these techniques to a DTOP-MV. The chapter highlights the benefits of using real-time IoT data to dynamically adapt vehicle routing decisions in urban environments.

Part III which focuses on the simheuristic framework and its implementations, is divided into two chapters as follows:

- Chapter 8 discusses how uncertainty can be represented and managed in optimization problems and introduces simheuristics, an approach that integrates simulation with heuristic optimization techniques.

- Chapter 9 introduces simheuristics as a hybrid approach that combines heuristic methods with MCS to tackle the complexities of real-world T&L problems. It demonstrates the application of simheuristics through a case study on the Coronavirus Disease 2019 (COVID-19) pandemic, solving an OVRP. The chapter highlights how simheuristics can dynamically manage uncertainty and provide reliable solutions in environments characterized by stochastic variables.

Part IV which focuses on DES heuristics and their applications, consists of seven chapters as follows:

- Chapter 10 explaines the concepts and related work on the discrete-event heuristics framework.

- Chapter 11 further enhances BR heuristics by refining search procedures to deliver high-quality solutions for T&L problems. The objective is to design optimal routes that maximize total rewards by picking up passengers while dynamically adapting to real-time traffic conditions and vehicle states.

- Chapter 12 explores the integration of DES platforms with Python to enhance simulation capabilities. The chapter provides a step-by-step tutorial on how to integrate Python with various DES software (both commercial and non-commercial), demonstrating the benefits of this integration.

- Chapter 13 presents a detailed step-by-step guide for integrating the FlexSim commercial simulator with the Python programming language using socket connections. This integration enables a simheuristic algorithm, developed in Python, to optimize product allocation within a warehouse, streamlining operations and improving efficiency.

- Chapter 14 examines the integration of Simul8 simulation software with heuristic algorithms in a real-world task sequencing problem for AGVs. The problem, characterized by dynamic queue formation and resource sharing, necessitates the use of advanced techniques like heuristics and simulation. Candidate task sequences are generated by a heuristic method and then evaluated through Simul8.

- Chapter 15 by exploring more complex applications of DES. It combines FlexSim simulation software with metaheuristics to address challenges in SLAP and TOP. The chapter highlights the advantages of using DES in solving complex logistics and urban mobility problems, providing robust and scalable solutions applicable to real-world contexts.

- Chapter 16 presents a novel approach to urban mobility in smart cities, addressing a complex TOP variant. Using a BR, Iterated Local Search (ILS) heuristic and simulation software, it delivers high-quality solutions while accounting for dynamic urban traffic. This simheuristic framework successfully bridges theory and real-world application.

Part V highlights the novel sim-learnheuristic framework, presented across two chapters as follows:

- Chapter 17 explains the concept of learnheuristics and reviews the related work in the field. It also introduces the novel concept of sim-learnheuristic.

- Chapter 18 applies designed sim-learnheuristic approach to the TOP with Unmanned Aerial Vehicles (UAVs), demonstrating its effectiveness in adapting to dynamic changes and uncertainties in urban environments.

Finally, Part VI, from Chapter 19 to 21 provides conclusions about the research conducted and highlights future challenging research directions based on the solution methods and applications discussed.

Part I

Concepts and State of the Art

Chapter 2

Transport Analytics

With the emergence of fog and edge computing, new possibilities have arisen for data-driven management of citizens' mobility in smart cities. This chapter reviews the concepts and published articles on various subproblems related to vehicular networks, edge, fog, and cloud computing, and IoT analytics combined with ML in transport analytics. These subproblems are characterized by features such as real-time data processing, communication protocols, and the application of ML and optimization techniques within ITS and smart city applications. The review delves into recent advancements and research findings, highlighting the potential of these technologies to improve urban mobility, optimize resource use, and enhance the efficiency of transportation networks. All our published articles contribute to this chapter.

2.1 Vehicular Networks

The concept of networks characterized by dynamic structures and limited transmission speed and quality is not new. In their 1999 paper, Corson et al. (1999) introduced the term "mobile ad hoc network". These networks consist of mobile routers that establish routes for information transmission as needed (Chin et al., 2002). The increasing number of vehicles has driven efforts to enhance road safety and inter-vehicle entertainment through vehicular systems (Abdelgadir et al., 2017). With advancements in wireless technologies and the rising popularity of the IoT, researchers have developed communication systems where vehicles directly participate in the network. As a result, networks such as Vehicular Ad Hoc Networks (VANETs) have been proposed to facilitate communication between vehicles and other entities, including Roadside Units (RSUs) and individuals (Ferrari et al., 2011). In ITS, vehicles use communication technologies to mitigate and eliminate transportation inefficiencies (Dimitrakopoulos et al., 2010). VANETs are an extension of this concept, wherein vehicles and RSUs act as network nodes that send, transmit, and receive data, enabled by a combination of wireless access and

network routing technologies (Anwer et al., 2014). Connectivity within VANETs is inherently demanding due to the dynamic behavior of network nodes, as vehicles enter, move through, and exit specific regions of the network (Yousefi et al., 2006). However, one key advantage of VANETs is that the movement of network nodes is somewhat predictable, given that vehicles follow certain paths on mobility grids (Harri et al., 2009). Ultimately, the interactions among all VANETs participants require fast and reliable communication to meet the goals of dynamic mobility systems (Jiang et al., 2008).

VANETs hold significant potential for improving vehicle mobility and implementing an efficient communication system for managing warning messages in smart cities. For instance, timely traffic alerts and real-time incident updates can reduce traffic congestion, enhance road safety, prevent accidents, and improve urban driving experiences. Moreover, real-time traffic alert systems can shorten travel distances, lower fuel consumption, and consequently reduce CO_2 emissions (Barba et al., 2012). Figure 2.1 illustrates VANETs communication in smart cities, where interaction occurs between Infrastructure-to-Infrastructure (I2I), Vehicle-to-Vehicle (V2V), Vehicle-to-Infrastructure (V2I), and Vehicle-to-Everything (V2X) entities, such as pedestrians, mobile devices, and traffic lights.

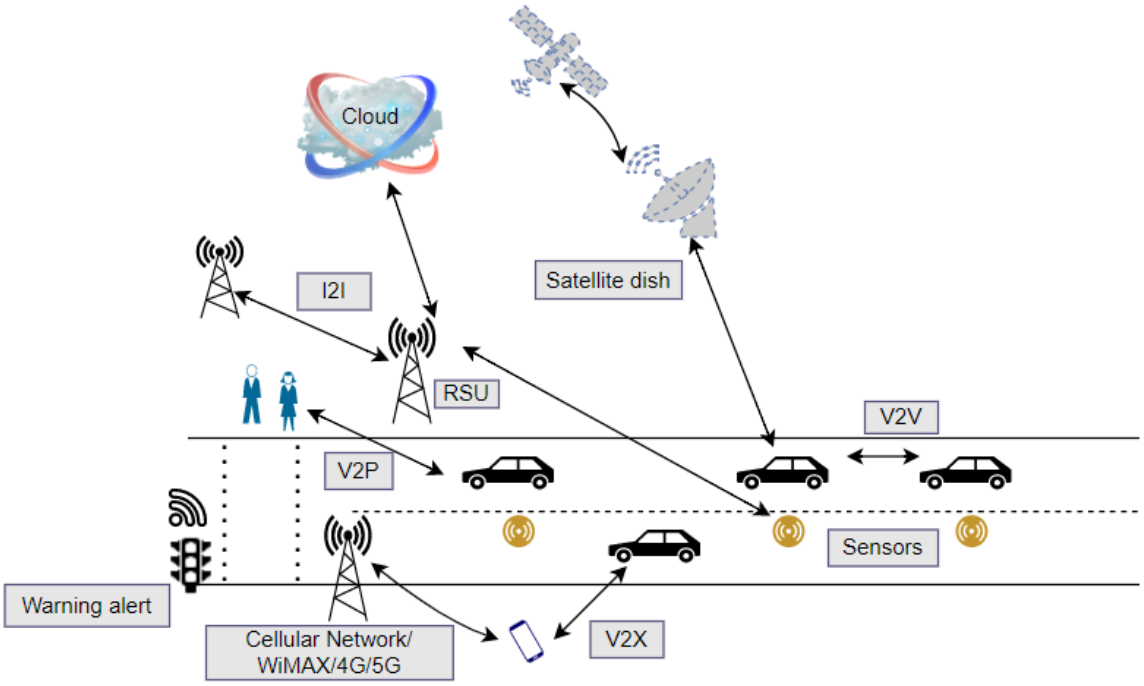


Figure 2.1: VANET in smart cities.

Source:Peyman, Fluechter, et al. (2023)

In the context of vehicular mobility, a vast array of use cases have been investigated and

implemented in research: IoT devices can be used to reserve and guide vehicles to parking spots (Liu et al., 2020; Thangam et al., 2018) or to avoid vehicle collisions by providing an exchange of information between network nodes (Daniel et al., 2016). The challenge of transmitting large amounts of data over a vehicular network has led to research into how external processing and storage could ameliorate the exchange of time-critical information. Data-center facilitated cloud computing can dynamically integrate into a VANETs application, allowing network nodes to off-load data-intensive applications (Bitam et al., 2015). Hussain et al. (2012) was the first to propose a cloud-based VANETs architecture to connect vehicles and support application loads. These vehicular cloud computing networks allow for scalable network architectures that support the ever-increasing stream of data and help alleviate connectivity limitations (Aliyu et al., 2018). Vehicle network technology supports ITS services, smart city applications, and urban computing. Qi et al. (2019) developed a knowledge-driven service offloading decision framework for internet of vehicle using deep reinforcement learning and the asynchronous advantage actor critic algorithm, achieving low service delay and optimal performance. Qiao et al. (2019) proposed a deep deterministic policy gradient based edge caching scheme optimizing content placement and delivery in vehicular edge computing networks, significantly reducing system cost and content access latency. Chen, Chen, et al. (2020) introduced a V2V task offloading scheme using max-min fairness and particle swarm optimization algorithms, which minimized task execution time based on the number of service vehicles. Wang, Ning, et al. (2020) presented an imitation learning-enabled online task scheduling algorithm for vehicular edge computing networks, demonstrating lower energy consumption and higher task-processing ratios compared to other algorithms.

2.2 Edge, Fog, and Cloud Computing

Edge, fog, and cloud computing facilitate the immediate transmission and processing of vast amounts of data produced by smart sensors linked to IoT devices. This information is transmitted to cloud-based decision-making systems, which frequently results in overloads of servers and systems. To address and mitigate this issue, edge and fog computing technologies were created. Specifically, fog computing improves the collaboration between cloud systems and IoT devices, aiding in their integration, data extraction, and analysis to maximize resource efficiency (Kumar, Kumar, et al., 2020). Fog computing possesses several defining characteristics, including minimal latency, widespread distribution, mobility, and a diverse array of nodes. These features were emphasized by Bonomi et al. (2012), facilitating the development of innovative applications and services within the IoT sector, particularly in connected vehicles, smart grids, and smart cities. In a related discussion, Peter (2015) recognized fog computing as a robust platform

for IoT, adept at handling data surges and resolving challenges associated with congestion and latency. Later, Bierzynski et al. (2017) investigated the combination of fog and cloud computing to tackle challenges associated with IoT. In this framework, Chen, Wen, et al. (2019) developed an edge computing system specifically for IoT applications within smart grids, successfully fulfilling high bandwidth requirements and mitigating low latency concerns. Their work included methods for ensuring privacy protection, predicting data trends, and implementing hierarchical decision-making via task grading and pre-processing facilitated by edge computing. Fog computing facilitates a variety of new applications, particularly in the realms of the IoT and advanced wireless technologies such as 5G and 6G, which are crucial for future technological advancements. Chiang et al. (2016) discussed both the prospects and obstacles associated with fog computing in IoT networking. Moreover, considering that cloud-based infrastructure for 5G is not ideal for high-demand scenarios such as transportation, tactile internet, and augmented reality—owing to issues like increased latency and higher computational and storage costs—Tufail et al. (2021) suggested that multi-access edge computing could serve as an effective alternative for smart city contexts. Beyond effective data management, cloud computing and the IoT provide solutions that improve security, privacy, scalability, reliability, and service quality in both cloud and vehicular networks. Specifically, Yan et al. (2012) highlighted significant challenges related to security and privacy, including the validation of high-mobility vehicles, the need for scalability, the handling of diverse identities and locations, and the creation of trust among various stakeholders during brief intermittent communications. To tackle these challenges, they put forward a directional security framework specifically designed to handle the complexities present in vehicular cloud environments. Likewise, Wang, Yin, et al. (2018) proposed a collaborative vehicular edge computing framework designed for capable vehicular networks. This framework provides the flexibility to facilitate collaboration in both horizontal and vertical dimensions, thereby enhancing the efficiency of distribution at the edges of the network. Leveraging advancements in cloud computing and IoT technologies, He, Yan, et al. (2014) introduced a vehicle data cloud platform characterized by a multilayered software architecture. The objective of this platform is to unify diverse devices found in vehicles and road infrastructure within the IoT ecosystem. In this context, Dobre et al. (2014) put forward the concept of context-aware and data-intensive applications aimed at tackling the challenges associated with big data management and enhancing the comprehension of traffic problems in major urban areas. Furthermore, Xiao et al. (2017) introduced a theory of vehicular fog computing, which converts connected vehicles such as buses and taxis into mobile fog nodes to provide economical computing resources as required.

2.3 Internet of Things Analytics and Machine Learning

IoT analytics plays a fundamental role in any ITS, as it involves the collection of extensive data, converting this data into valuable insights, creating predictive models, and facilitating informed decision-making (Muthuramalingam et al., 2019). IoT improves digital services and operations for different user demographics, aiding the creation of more intelligent urban and rural areas. It enables an understanding of the interactions among sensors, energy sources, and data storage by examining the movement patterns of digital devices such as smartphones and vehicles. Utilizing the widespread application of IoT and Big Data in ITS, Dai et al. (2021) enhanced transportation management through traffic optimization derived from an analysis of diverse road traffic challenges. The operation of vehicles and interactions between parties in traffic situations produces substantial quantities of raw data in ITS. To interpret this sensor data effectively, Swarnamugi et al. (2021) investigated the potential for utilizing modeling techniques and reasoning methods within ITS. They also recognized ML and deep learning methods utilized in the reasoning phase. Mohandu et al. (2021) examined insights from data analytics relevant to the transportation and mobility sector, covering implementations of ITS, threshold scenarios, and applications like routing, planning, and platform architecture. Likewise, Calabrese et al. (2013) investigated methods for obtaining effective mobility statistics relevant to transportation studies. Through the combination of statistical analysis and mobile phone tracking data, they established a dependable indicator of individual movement patterns, offering valuable perspectives on intra-urban mobility differences as well as the non-vehicular aspects of total mobility. At two analytical tiers(GPS data traces and street segments) Jiménez-Meza et al. (2013) employed GPS information, including date-time, latitude, and longitude, to determine travel time, distance, and speed. They evaluated street segments by computing service levels derived from average speeds. Following this, Zanella et al. (2014) examined various technologies, protocols, and architectures pertinent to urban IoT with the objective of establishing a comprehensive urban-scale ICT platform, illustrated through case studies in Padova, Italy. Mahdavinejad et al. (2018) examined multiple ML techniques aimed at tackling the challenges posed by IoT data in smart cities. They categorized various algorithms and explained their application for enhancing the precision of information extraction. Simultaneously, Graser (2019) presented a Python library named MovingPandas, which is based on Pandas and GeoPandas and is specifically designed for analyzing movement data. They assessed current frameworks to identify essential features for this library and showcased its usage in independent Python scripts. Pappalardo et al. (2019) tackled the issues of data cleaning related to unprocessed spatio-temporal trajectories and created an algorithm designed to synthetically produce trajectories that mimic authentic human movement behavior. In a similar vein, Bao et al. (2019) performed a mathematical

evaluation of transportation systems and introduced a multi-agent deep deterministic policy gradient framework aimed at enhancing real-time signal control strategies within extensive ITS. Additionally, Teng et al. (2019) proposed an economical code distribution strategy that employs vehicles as carriers in a smart city environment. Code stations obtain the latest code from the cloud data center and relay it to the vehicles. They refined the selection of code and deployment methods to enhance distribution reach throughout the city, all while reducing expenses and time required. Cao, Wachowicz, et al. (2017) introduced an edge computing platform that employs descriptive analysis to extract significant patterns from real-time data streams related to bus transit. They emphasized the platform's potential uses in areas like autonomous vehicles, intelligent intersections, and smart traffic light systems. Similarly, Yongjun et al. (2012) developed a system utilizing IoT and GPS technology to effectively gather precise data on bus operations. This information is transmitted to a centralized dispatch center and database. The system provides essential details such as current traffic conditions, bus locations, estimated arrival times, and bus routes, enabling the monitoring center to modify the bus schedule according to real-time traffic updates. Panadero, Freixes, et al. (2018) explored the stochastic variant of the TOP related to UAVs, introducing a simheuristic algorithm (Chica, Juan, Christopher, et al., 2020) that integrates a BR heuristic (Quintero-Araujo, Caballero-Villalobos, et al., 2017) with simulation methods for an agile optimization framework (Martins, Tarchi, et al., 2021). Furthermore, Adi et al. (2020) examined the ML analysis process in the generation of data within the IoT and created a framework that facilitates applications to learn from various other IoT systems. Lastly, Wei et al. (2020) concentrated on optimizing bus routes by taking into account the relationship and competition between metro and bus services, formulating a quantitative approach to evaluate their methodology.

Chapter 3

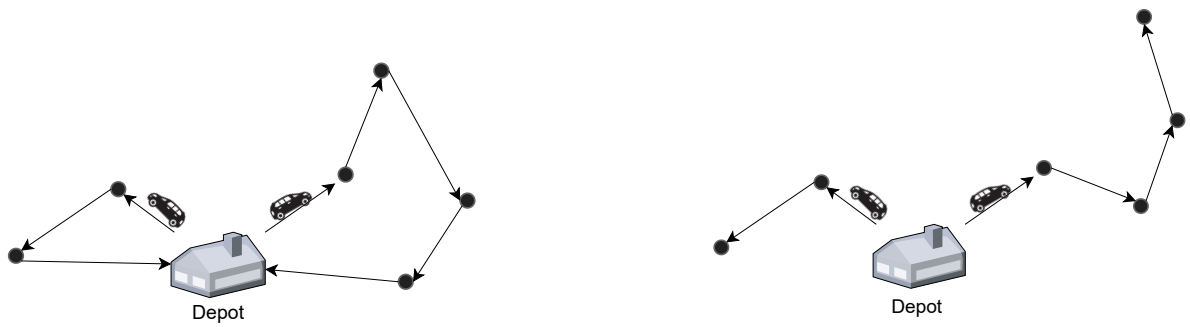
Logistics and Smart Cities Mobility

This section reviews the concepts and published literature on various subproblems related to the broader VRP, including the OVRP, the RSP, and the TOP. Additionally, given the significance of efficient logistics and operations, it examines related work on challenges such as the SLAP, and task sequencing. All our published articles contribute to this chapter.

3.1 The Open Vehicle Routing Problem

The OVRP is an extended form of the traditional VRP (Yousefikhoshbakht et al., 2014). As depicted in Figure 3.1, part 3.1a shows that VRP is focused on finding the optimal routes for a fleet of vehicles to serve a specific set of customers, represented by black circles. This is one of the most crucial and extensively researched combinatorial optimization problems. The VRP is a well-known NP-hard problem (Lenstra and Kan, 1981) has various applications within the transportation sector (Braekers et al., 2016). In its simplest form, the problem involves a set of customers, each with a known demand to be fulfilled by a single depot with virtually unlimited capacity. A fleet of homogeneous, capacitated vehicles is used to deliver a single product to the customers. Each customer must be visited once. Due to the vehicles' limited capacity, multiple routes must be created to meet all customer demands. After deliveries, vehicles return to the depot. The main goal is to minimize transportation costs, typically measured by the total distance traveled or time taken (Pisinger et al., 2007). In contrast, as shown in 3.1b, the OVRP differs from the VRP in that the routes are not "closed" meaning the origin and destination depots are not the same. Each route begins at the depot and ends at one of the customers. Each customer is visited exactly once by a single vehicle, and their demand is fully satisfied. The total demand of customers assigned to each route does not exceed the vehicle's capacity. The objective in the OVRP is to minimize the total travel and vehicle operating costs. In this context, Li, Golden, et al., 2007 conducted a review and comparison of various

algorithms aimed at addressing the OVRP, introducing a record-to-record travel algorithm designed for standard VRP to effectively manage open routes. Sariklis et al. (2000) offered a heuristic approach that utilizes a minimum spanning tree along with penalties to tackle the OVRP. Meanwhile, Cao and Lai (2010) focused on an OVRP characterized by fuzzy demands, suggesting a chance-constrained model. In a similar vein, Gruler, Quintero-Araújo, et al. (2017) integrated techniques from BR, metaheuristics, and MCS to address waste collection challenges involving stochastic factors. Furthermore, Delfani et al. (2020) introduced a multi-objective mathematical model targeting the hazardous waste location-routing problem, with the goal of minimizing overall costs. They devised a possibilistic chance-constrained programming method and implemented a robust possibilistic programming framework to manage uncertainties in model parameters.



(a) Basic VRP (b) Basic OVRP

Figure 3.1: Network structure for simple VRP and OVRP.

3.2 The Ridesharing Problem

Ridesharing is a key concept in modern transportation, providing societal benefits such as reducing traffic congestion, noise, and pollution, as well as lowering fares for passengers and operational costs for drivers. In ridesharing systems, individuals offer trips using their private vehicles, with dynamic systems that automate the matching of drivers and riders through a rideshare provider (Agatz et al., 2011).

The RSP involves coordinating a group of vehicles, each operated by a different owner, to transport passengers with similar travel routes. In this scenario, drivers offer the empty seats in their vehicles to users needing a ride. Each user submits a request for service, providing their current location, and drivers adjust their routes to pick them up. The vehicles in this problem have a limited capacity, allowing them to carry multiple passengers at once. Therefore, each

driver's route typically includes the starting point (such as their home), several pickup locations where passengers are collected, and a final destination. A key challenge in the RSP is that not all ride requests can be fulfilled due to the restricted vehicle availability and their seating limits. This means that drivers must decide which users to pick up, based on the proximity of users to the driver's current route and the amount users are willing to pay for the ride. The objective of the RSP is to maximize the total fees collected by the drivers by designing routes that efficiently pick up as many users as possible, within the constraints of vehicle capacity and the drivers' flexibility to adjust their routes (Martins, Torre, et al., 2021). Due to the complexity of RSP, researchers employ various heuristics and mixed-integer/integer linear programming models to solve large-scale instances. Figure 3.2 demonstrates a full solution for the basic RSP version, where users served by vehicles are represented by connected houses, and those not served are shown as non-connected houses.

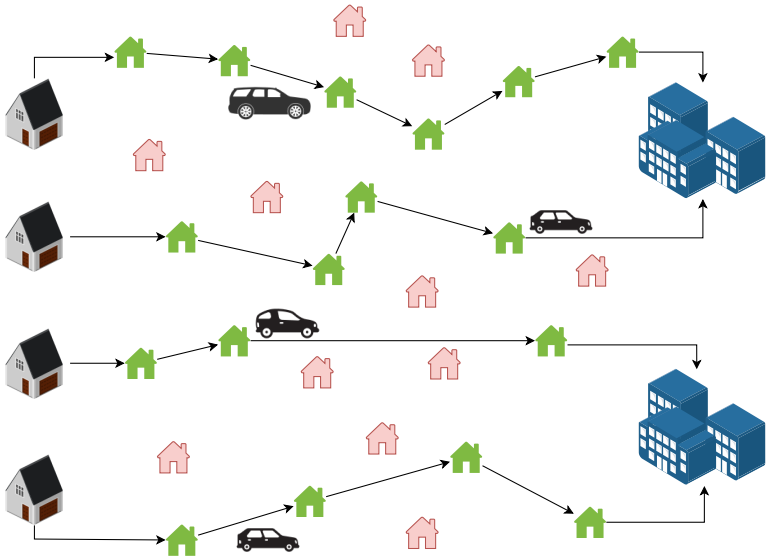


Figure 3.2: An RSP's fundamental network topology.

Source:Peyman, Copado, et al. (2021)

Dynamic ridesharing, enabled by low-cost geo-locating devices, smartphones, wireless networks, and social networks, arranges ad hoc shared rides in real time (Stach et al., 2011). This real-time nature is central to the dynamic RSP, where passenger requests are generated spontaneously. Huang et al. (2013) introduced a branch-and-bound algorithm to tackle ridesharing with service guarantees on road networks, along with a kinetic tree algorithm to improve scheduling of dynamic requests and route adjustments. Their adaptable algorithm accounts for changing road network configurations and traffic conditions, generating a server trip schedule

based on real-time locations and constraints. It integrates new requests with existing ones to share trips among customers, allowing for flexible pickup and drop-off locations. Bathla et al. (2018) presented four ridesharing system models based on varying pickup and drop-off locations. They introduced a dynamic algorithm for models where these locations differ for all users. Using Density-based spatial clustering, they simulated multiple ridesharing requests from nearby regions, calculating distances with the Haversine formula and the Google Maps Direction API. Their algorithm, evaluated on ridesharing metrics like request satisfaction and waiting time, also included a taxi distance minimization algorithm, effectively accommodating higher ridesharing among passengers. Alisoltani et al. (2022) developed a dynamic ridesharing matching algorithm combining exact methods, Artificial Intelligence (AI) techniques, clustering, and heuristics. Their approach efficiently provided an efficient solutions for large-scale problems, optimizing travel time, distance, and passenger waiting time, simulated using macroscopic fundamental diagrams and travel time prediction models. Aydin et al. (2020) introduced a ride matching algorithm considering participants' characteristics and preferences, with a joint socialness score for matching drivers and riders. Utilizing a modified Needleman-Wunsch algorithm and a first-come, first-served method, their heuristic approach effectively managed complex, large-scale problems despite longer computation times. Kornhauser et al. (1977) created a simulation for Trenton and New Jersey to assess the productivity potential of dynamic ridesharing systems on a hypothetical automated guide-way transit network, testing various policies based on vehicle origin-destination capabilities. Fagnant et al. (2018) focused on optimizing fleet sizing in dynamic ridesharing, improving model performance and offering a cost-benefit analysis for fleet operators.

3.3 The Team Orienteering Problem

The TOP, initially presented by Chao et al. (1996), represents a significant challenge in optimization that centers on the creation of effective routes for a fleet of vehicles. The primary objective is to enhance the overall reward earned from customers along these designated paths while complying with an established time limit, as illustrated in Figure 3.3. Addressing the TOP is crucial in fields such as search and rescue, surveillance, and logistics. For example, during a search and rescue mission, a fleet of UAVs must traverse a disaster zone to locate survivors and distribute supplies. UAVs have become essential for these operations due to their capability to undertake hazardous tasks and their expertise in sensing, monitoring, and navigating; however, they encounter significant challenges (Gupta et al., 2021).

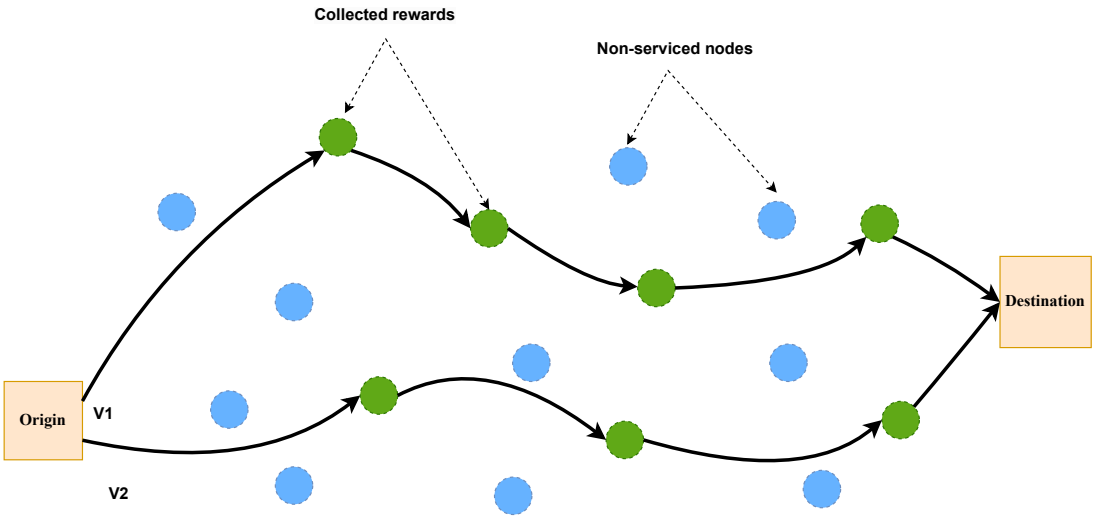


Figure 3.3: The basic concept of TOP.

In recent years, numerous significant studies have emerged. For instance, Tricoire et al. (2010) introduced an adaptive algorithm rooted in the Markov decision process to decide whether to persist on a particular route or opt for a shortcut based on an assigned deadline. Additionally, Mufalli et al. (2012) investigated the assignment of sensors and routing for UAVs aimed at optimizing intelligence collection, while tackling limitations such as battery capacity and sensor weight. Their work provided a thorough methodology that employed mathematical models and local search strategies, utilizing column generation to improve efficiency during larger missions. Similarly, Saeedvand et al. (2020) proposed a hybrid approach aimed at addressing the TOP during rescue operations that utilize humanoid robots. Their method emphasized enhancing energy efficiency, task performance, and decision-making by combining a learning algorithm with an evolutionary multi-objective framework.

Schmitt-Ulms et al. (2022) introduced a ML technique aimed at addressing a stochastic TOP with time constraints. They utilized a reinforcement learning strategy, particularly the Q-learning algorithm, which allows a delivery person to make routing choices despite unpredictable travel durations and fluctuating customer requirements. Furthermore, Lee and Ahn (2024) presented an efficient deep reinforcement learning method designed for tackling Multi-Start (MS) TOP in UAVs mission re-planning. Their approach adjusts to evolving circumstances by consistently refining a policy network informed by insights gained from previous UAV mission planning experiences. Xu, Xu, et al. (2020) explored a new application of EVs in TOP systems, specifically targeting the charging of energy-sensitive sensors using mobile chargers. Due to the limitations of the chargers, not all sensors could be visited, and costs were calculated by factoring in both charger energy consumption and travel expenses. The study also took into account

various vehicle types and their capabilities, with the possibility that sensors could be visited by multiple vehicles, impacting profit margins. An initial approximation algorithm was proposed, which was later refined to handle vehicles of the same type more effectively. In contrast, Sundar et al. (2022) proposed a concurrent multi-threaded branch-and-price algorithm with acceleration techniques for solving the TOP involving fixed-wing drones. They tackled kinematic constraints that restrict drones' maneuverability, especially concerning rapid turns and minimum turn radii. Although their approach provided optimal solutions in most cases, its dependence on exact methods limited its ability to deliver real-time solutions, which are essential for dynamic systems. Wang, Bian, et al. (2023) addressed the dynamic and stochastic orienteering problem in autonomous transportation systems. They developed self-adaptive heuristic algorithms to optimize task execution for intelligent agents, aiming to maximize rewards while ensuring timely returns to the starting point within a specified time frame. The orienteering problem allows agents to selectively visit locations and optimize routing sequences for reward maximization. The study introduced a hybrid simulated annealing–tabu search algorithm for initial routing plans and three multi-stage optimization strategies for real-time adjustments. Simulation experiments validated the algorithm's effectiveness, enhancing solution quality and ensuring timely arrivals for agents. Elzein et al. (2022) introduced an innovative multi-stage metaheuristic algorithm to efficiently address large orienteering problem instances. The algorithm divides a large set of candidate sites into smaller clusters, and a solver is applied to find near-optimal solutions within a limited time frame.

3.4 The Task Sequencing Problem

Understanding the impact of shared resource availability on task sequencing and scheduling is critically important. Since its introduction several decades ago (Dike, 1964), the Resource-Constrained Scheduling Problem (RCSP) has become a key area of extensive research in optimization (Ding et al., 2023). Over the years, this problem has evolved, drawing significant attention from researchers seeking to improve efficiency in environments where resources are shared and limited. This problem involves assigning specific execution times to a set of tasks or activities in order to minimize the total project duration or makespan, while ensuring that various constraints are satisfied. These constraints are usually grouped into two types: (i) the resources needed for each activity, and/or (ii) the sequence in which activities must occur, often known as precedence constraints (Van Eynde et al., 2020). Given its practical significance, the RCSP has been extensively studied. For example, Hartman et al. (2011) provides a comprehensive survey of RCSP and its extensions, emphasizing the importance of efficient resource allocation in achieving optimal scheduling solutions. Recent advancements in computational techniques have

driven the creation of innovative methods to address the challenges of the RCSP. In particular, Beek et al. (2023) proposed a hybrid differential evolution algorithm designed for the RCSP. This algorithm accounts for a flexible project structure and the dynamics of resource consumption and production, demonstrating the effectiveness of evolutionary algorithms in managing the complex tasks of resource allocation and task sequencing. Equally relevant to this study is a review of the literature on the Robotic Task Sequencing Problem (RTSP). The RTSP has been widely researched, with efforts focused on optimizing the sequence of tasks performed by robots across various industrial applications. Foundational work by Maimon (1990) established the basis for RTSP, focusing on the efficient use of robot task flexibility and offering a framework and classification for these problems. Since then, numerous approaches and methodologies have been developed to tackle the challenges associated with the RTSP. For instance, Suárez-Ruiz et al. (2018) made a notable contribution by developing a method to optimize the sequence in which a robot visits multiple targets in industrial settings. This approach led to faster RTSP solutions, which are particularly valuable in time-critical operations. Later, Li, Wang, et al. (2022) presented an efficient approach using a decoupling strategy to determine the optimal sequence of collision-free motions for robots performing repetitive tasks. This approach highlights ongoing efforts to improve both efficiency and safety in robotic operations, especially in environments where collision avoidance is crucial. Additionally, Chen, Chou, et al. (2021) explored optimizing joint-space tour scheduling to meet manufacturing criteria for each task point visit. Their work emphasizes the importance of planning the sequence of robotic tasks to achieve operational excellence in industrial environments. Some researchers have adapted RTSP concepts to AGVs applications, which serve as a prime example of Industry 4.0. These applications necessitate the integration of new technologies and concepts to boost productivity while minimizing labor costs, energy usage, and emissions, with a focus on optimizing the performance of individual AGVs units. For example, Li, Liu, et al. (2019) translated the RTSP into a traveling salesman problem, a common approach in robotics. They introduced a heuristic that reduced its complexity from $O(N^3)$ to $O(N^2)$ by employing auxiliary data structures. Another important problem family that focuses on optimizing task sequences executed by robots is the Multi-Robot Task Allocation (MRTA) problem. In Korsah et al. (2013), the researchers provided a comprehensive taxonomy for MRTA, creating a framework that supports the efficient use of robot task flexibility. This taxonomy helps clarify the features and complexity of MRTA problems, which have been extensively studied in various research efforts. For instance, Khamis et al. (2015) examined multi-robot systems performing collective behaviors to address complex tasks. Similarly, Alshaboti et al. (2021) explored cutting-edge algorithms and strategies used to solve MRTA problems, providing valuable insights into the evolving field of robot task allocation. Additionally, Chakraa et al. (2023) reviewed optimization techniques for multi-robot

task allocation, presenting current methods and highlighting future directions for improving task allocation in multi-robot systems.

3.5 The Storage Location Assignment Problem

Various interpretations of the SLAP have been examined in academic literature, resulting in a somewhat ambiguous definition that varies according to the researchers and the context involved. The differences among SLAP formulations arise from several factors, including the objective function (such as minimizing travel distances or maximizing available space), scope (for instance, whether it encompasses orders or picking processes), product characteristics, types of warehouses (like manual versus automated), and specific constraints taken into account (such as storage capacity, order-picking resource limits, and dispatching policies). In Reyes et al. (2019), a comprehensive literature review is presented, ranging from foundational papers that categorize the SLAP as an NP-hard problem to the latest developments in the field. In that study, the SLAP is defined as "an operational decision related to the accommodation [of products] and picking process, affecting batch definition, classification, routing, and order sequencing", highlighting the broad scope of the problem. The key policies, illustrated in Figure 3.4, include *random* storage, *class-based* storage, and *dedicated* storage. The expected minimum and maximum performance for each policy, in terms of space maximization, are analyzed in Fumi et al. (2013). In their approach to the SLAP problem, Fumi et al. (2013) reformulate it as a vertex coloring problem and solve it using mathematical programming. They conclude that the random policy yields the best performance, while the dedicated-storage policy performs the worst in this case. Integer linear programming is widely used for this type of problem because it provides a straightforward way to model the problem. As the complexity of the SLAP increases with larger instances, heuristics and metaheuristics have become popular solution methods. For example, both Xie et al. (2014) and Chen, Langevin, et al. (2011) use integer linear programming to formulate the problem, with the former applying a genetic algorithm and the latter utilizing a tabu search algorithm to solve it.

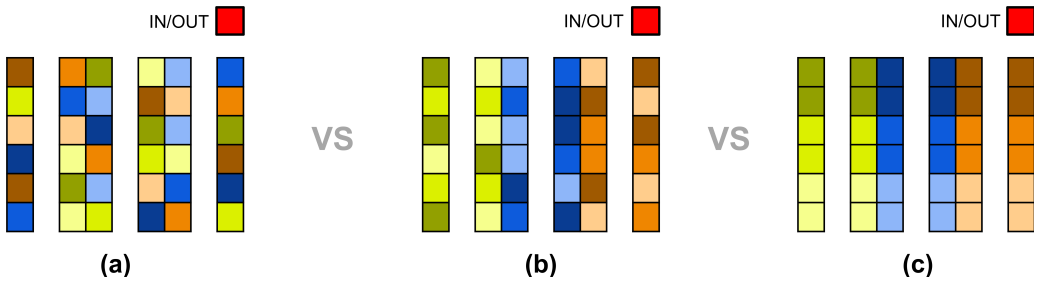


Figure 3.4: Different SLAP policies: (a) random; (b) class-based; (c) dedicated.

Source:Leon et al. (2023)

While many studies identify picking as the primary operation affected by the assignment of storage locations regarding efficiency, some scholars have pointed out that orders serve as an important element. Orders can be a major source of dynamism (time-sensitive changes), which is becoming more essential as markets require enhanced responsiveness and adaptability. Nevertheless, it is possible to recognize the value of incorporating orders in a static way without overly complicating the SLAP problem through dynamic integration. Implementing a dynamic version would require the warehouse to possess the technological capacity to manage the inputs and outputs associated with such a solution. This capability is generally only achievable for large enterprises (Boysen et al., 2019).

On a different note, addressing the static SLAP with predefined orders—where specific orders to be fulfilled within a designated planning period are clearly outlined and integrated into the optimization process—results in increased accuracy of the modeled system while reducing costs. This approach also improves decision-making in multiple real-world warehouse environments, including those that follow more conventional configurations. For instance, Silva et al. (2020) tackles an issue same as the one discussed in this context, focusing on the simultaneous resolution of the static SLAP alongside the order picking challenge. This method evaluates the effectiveness of both the SLAP and order picking operations in tandem. In Mantel et al. (2007), an order-oriented slotting approach is proposed, where the decision on item storage is based solely on the orders that need to be fulfilled. The overall approach is similar to the one we use: given a routing strategy and a set of orders, items are allocated with the goal of minimizing the total traveled distance. The primary distinction between these studies and our work lies in the use of deterministic linear mixed integer programming formulations and predefined routing strategies. In our approach, while we share the same motivation for integrating the problem, the optimal route is calculated concurrently with the solution evaluation, and the robustness of the solution is enhanced through the use of the simheuristic framework. Recent scientific literature on the SLAP reveals a growing trend among researchers toward a more holistic and integrated

approach to the storage location assignment problem, rather than addressing the SLAP in isolation. This shift highlights a gap in the existing body of SLAP research. For instance, Keung et al. (2021) addresses a variant of the SLAP by analyzing order patterns in a highly automated warehouse utilizing robotic systems. These systems can adapt their navigation (using the A* algorithm, as in our study) based on the provided inputs. The researchers further extended this application to incorporate additional aspects of the SLAP within the IoT context (Keung et al., 2022). In Zhang, Shang, et al. (2021), the SLAP is tackled by integrating production planning (static over a fixed planning horizon) with the storage location policy in a single model. The researchers discovered that, when adopting an integrated approach, the random policy can actually be the most effective for reducing costs in a real-world warehouse scenario. In Lanza et al. (2022), the SLAP is addressed by considering sequencing decisions that occur in their warehouse after products have been allocated. Similarly, Xu and Ren (2022) tackles the SLAP problem in a traditional (non-automated multi-picker) warehouse, focusing on aisle congestion and the demand correlation between products.Specifically, the picking routes are adjusted to alleviate congestion, with validation performed through a network numerical model. It is evident that, to enhance the effectiveness of their proposed SLAP solutions, researchers are increasingly expanding their focus to include operational areas beyond the isolated SLAP. The use of simulation, particularly simplified models to validate the proposed algorithms, also appears to be an emerging trend in the literature. For example, in Guo et al. (2021), a dynamic version of the SLAP is explored in a real-world industrial setting, where products arrive and need to be stored within the warehouse. To validate the proposed algorithm and compare it with the actual case, MCS was used. In Montanari et al. (2021), the relationship between a novel SLAP solution method and the routing policy is analyzed. A simulation built in Microsoft Excel was used to evaluate the efficiency of the proposed solution.

Part II

Agile Optimization

Chapter 4

Agile Optimization Framework

4.1 Main Concept

Real-world applications in ITS, particularly in urban environments where conditions can fluctuate rapidly, often require real-time decision-making. For instance, unexpected disruptions in traffic systems underscore the necessity for adaptive decision-making frameworks that can promptly offer users alternative solutions. In such dynamic scenarios, intelligent systems must rapidly adapt to provide optimal routes or adjustments. These problems, particularly the underlyingT&L problems, are known to be NP-hard, as demonstrated by previous studies (Nagy et al., 2007). Although traditional approaches, such as mathematical models and exact algorithms, have been used to address these challenges, the rise of technologies like IoT (Lin et al., 2017), the expansion of smart cities (Faulin et al., 2018), and the introduction of autonomous vehicles (Bagloee et al., 2016) require solutions that can operate in real-time. While exact algorithms can guarantee optimal solutions, their computational time (ranging from hours to days) makes them impractical for real-time applications. In contrast, heuristic and metaheuristic approaches, like BR heuristics, offer a more viable solution by providing high-quality or near-optimal results within a much shorter time frame. This document delves into the agile optimization framework, emphasizing BR heuristics, metaheuristics, and agile optimization algorithms. The following content is based on our published work, contributing to the understanding of these methods.

Heuristics are structured procedures aimed at solving specific problems by following a series of logical steps. Typically, these methods rely on deterministic actions, meaning that starting from the same initial conditions will always lead to the same results. While heuristics are particularly useful for generating feasible solutions quickly, especially for combinatorial optimization problems, their deterministic nature can sometimes restrict their ability to explore a broader solution space. To address this limitation, randomness or adaptive strategies can be incorporated, allowing heuristics to explore a more diverse set of solutions and avoid local

optima. Enhanced heuristics can better balance the exploration of the solution space while still focusing on promising areas, improving their overall effectiveness. Because of their ability to generate quick, feasible solutions, heuristics often form the basis for more advanced techniques, such as metaheuristics and simheuristics. Metaheuristics are high-level strategies designed to solve complex optimization problems across various domains by building upon heuristic solutions. Unlike heuristics, which are often problem-specific, metaheuristics offer versatile frameworks that can be adapted to different problem types. This adaptability is especially important for real-world, dynamic optimization problems where input variables (such as customer demand, travel times, or inventory levels) are constantly changing. For example, dynamic VRP with time windows or inventory routing problems often require adaptive, real-time solutions. Previous studies (Hezarkhani et al., 2016; Juan, Grasman, et al., 2014) have demonstrated that hybrid metaheuristics (such as variable neighborhood algorithms combined with tabu search or hybrid roll-out algorithms) are particularly effective in dealing with such challenges. These advanced techniques, including simheuristics and agile optimization algorithms, can efficiently process real-time data and optimize decisions dynamically, especially when integrated with IoT technologies. Over recent years, several techniques have been developed to tackle both stochastic and non-stochastic combinatorial optimization problems. Heuristics and metaheuristics have gained prominence due to their ability to provide near-optimal solutions within a reasonable time frame, especially in large-scale problems that require real-time re-optimization. One of the key challenges in these scenarios is balancing solution quality with computational efficiency. BR techniques, which introduce probabilistic elements to traditional heuristics, have emerged as a solution to this challenge. BR algorithms enhance classical heuristics by introducing non-symmetric probability distributions (e.g., geometric or triangular distributions) into the solution-building process, enabling the generation of alternative, high-quality solutions without compromising speed (Grasas et al., 2017; Estrada-Moreno et al., 2020). In BR algorithms, randomness is incorporated into the candidate selection phase, where actions or elements are selected based on skewed probability distributions rather than purely deterministic rules. This approach allows the algorithm to explore different regions of the solution space, increasing the likelihood of finding high-quality solutions and avoiding local optima. By repeating the algorithm multiple times with different random seeds, a broader range of solutions can be generated, leading to more diverse and effective outcomes. This concept is further enhanced when BR algorithms are embedded into a MS framework, where each run of the algorithm starts from the same initial conditions but explores different regions of the solution space, thanks to the probabilistic nature of the BR approach. As a result, the MS-BR approach can significantly improve solution quality by escaping local optima and promoting diversification.

Agile optimization strategies take this concept even further by leveraging parallel computing.

By running multiple instances of the BR algorithm in parallel, each thread or core executes a separate run of the algorithm, allowing for a pool of solutions to be generated simultaneously (Figure 4.1). This parallelization drastically reduces computation time while maintaining the ability to find near-optimal solutions. Agile optimization is particularly effective for solving large-scale, NP-hard problems in real-time, where frequent re-optimization is necessary as new data or environmental changes occur. Examples of agile optimization in practice include real-time routing for humanitarian logistics, where dynamic conditions necessitate constant re-optimization (C. Martins et al., 2021), and intelligent transportation systems, where real-time traffic conditions require frequent adjustments to vehicle routes and assignments (Martins, Tarchi, et al., 2021).

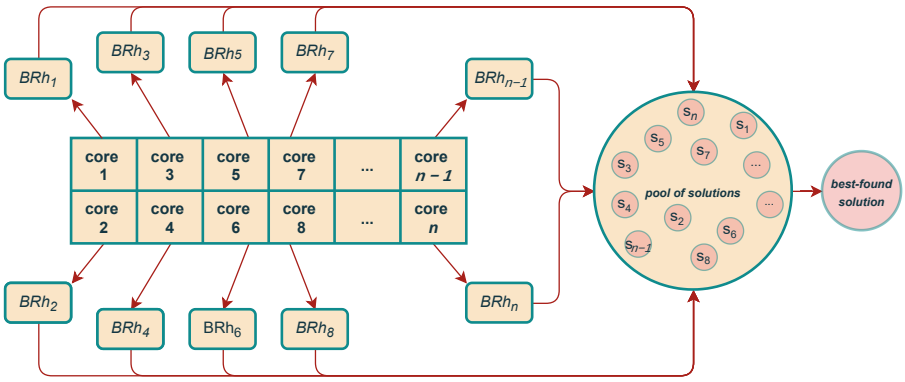


Figure 4.1: The concept of agile optimization.

Source:Peyman, Copado, et al. (2021)

4.2 Related Work

BR heuristics have proven effective in tackling NP-hard problems across various domains in recent years, as demonstrated in the Table 4.1. They have been applied to solve the VRP (Reyes-Rubiano et al., 2020; Dominguez Rivero et al., 2016; Dominguez et al., 2016), the location routing problem (Quintero-Araujo, Caballero-Villalobos, et al., 2017), the facility location problem (Estrada-Moreno et al., 2020), and the arc routing problem (Gonzalez-Martin et al., 2012). Additionally, they have been utilized in permutation flow-shop scheduling (Juan, Lourenço, et al., 2014), waste collection (Gruler, Juan, et al., 2018), horizontal cooperation (Quintero-Araujo, Gruler, et al., 2019), portfolio optimization (Kizys et al., 2019), marketing (Marmol, Goyal, et al., 2021; Marmol, Martins, et al., 2020), and catastrophe insurance (Bayliss, Guidotti, et al., 2020). Moreover, agile optimization has emerged as a promising approach, enabling flexible and adaptive decision-making in complex optimization scenarios. Agile optimization methodologies have been particularly valuable in addressing dynamic environments that require rapid adaptation, such as urban freight distribution during the COVID-19 pandemic, where changes in demand and logistics constraints necessitate real-time solutions. For instance, agile optimization has been applied to real-time routing algorithms in urban logistics, supporting the efficient delivery of parcels through micro-hub networks (Castillo et al., 2024). This approach also extends to the internet of vehicles, where rapid reallocation of resources is crucial to maintaining service quality in changing conditions (Martins, Tarchi, et al., 2022). Furthermore, agile optimization has been demonstrated to be effective in time-sensitive vehicle routing problems, such as humanitarian logistics, where quick decision-making is vital for rescue operations (C. Martins et al., 2021). These studies highlight the significant role of agile optimization in achieving high-quality solutions within tight time constraints, contributing to efficient and sustainable outcomes in various real-world applications.

Table 4.1: Summary of Related Work on Agile Optimization.

Reference	Area	Contribution
(Reyes-Rubiano et al., 2020)	Vehicle Routing Problem	Optimized routes for reducing costs and improving delivery times.
(Dominguez Rivero et al., 2016)	Vehicle Routing Problem	Developed methods for enhancing route efficiency in logistics.
(Dominguez et al., 2016)	Vehicle Routing Problem	Improved VRP solutions focusing on delivery efficiency.
(Quintero-Araujo, Caballero-Villalobos, et al., 2017)	Location Routing Problem	Applied methods for optimal location and routing decisions.
(Estrada-Moreno et al., 2020)	Facility Location Problem	Developed solutions for selecting facility locations.
(Gonzalez-Martin et al., 2012)	Arc Routing Problem	Optimized routes in arc routing scenarios, such as waste collection.
(Juan, Lourenço, et al., 2014)	Flow-shop Scheduling	Improved job scheduling efficiency, reducing processing times.
(Gruler, Juan, et al., 2018)	Waste Collection	Optimized waste collection routes, achieving fuel savings and time reduction.
(Quintero-Araujo, Gruler, et al., 2019)	Horizontal Cooperation	Enhanced logistics collaboration, improving resource sharing among partners.
(Kizys et al., 2019)	Portfolio Optimization	Optimized asset allocation, improving risk-return balance in financial portfolios.
(Marmol, Goyal, et al., 2021)	Marketing	Optimized marketing resource allocation, increasing campaign effectiveness.
(Marmol, Martins, et al., 2020)	Marketing	Improved targeting and allocation in marketing strategies.
(Bayliss, Guidotti, et al., 2020)	Catastrophe Insurance	Optimized risk management strategies, reducing potential losses.
(Castillo et al., 2024)	Urban Freight Distribution	Proposed real-time routing and micro-hub selection during the pandemic.
(Martins, Tarchi, et al., 2022)	Internet of Vehicles	Addressed dynamic resource allocation challenges in IoV scenarios.
(C. Martins et al., 2021)	Humanitarian Logistics	Developed quick decision-making techniques for vehicle routing in rescue operations.

Chapter 5

Predictive Analyses of Traffic Level in the City of Barcelona

As traffic prediction becomes increasingly crucial for urban transportation management, numerous traffic flow prediction models have been explored in recent decades. This chapter[1] employs visualization techniques, such as heatmaps, to demonstrate traffic density across different city sections throughout the day. By analyzing vehicle locations at various time intervals, it highlights mobility patterns and street traffic density. An interactive animation plugin is introduced, allowing users to customize the visual representation with features like overview, zoom, pause, play, loop, and adjustable playback speeds. Furthermore, traffic prediction is approached as a time series analysis problem, with ARIMA and XGBoost models used to forecast traffic flow across ten sections of the same street.

[1]The contents of this chapter are based on the following work:

- Garcia, Eloi, Laura Calvet, Patricia Carracedo, Carles Serrat, Pau Miró, and **Mohammad Peyman** (2024). *"Predictive analyses of traffic level in the city of Barcelona: from ARIMA to eXtreme gradient boosting"*. In: Applied Sciences 14.11, p. 4432.

5.1 Problem Description

Mobility has become a defining feature of modern life globally. Over recent decades, a primary policy goal has been to enable citizens to travel faster, further, and more safely and comfortably. Urban mobility has played a crucial role in this process, closely tied to the rise of mass motorization. In Europe, drivers spend an average of around 20 minutes daily behind the wheel on personal trips during working days, and between 25 to 30 minutes on business trips (Pasaoglu et al., 2012). Barcelona stands as Spain's second-largest city by population and is positioned seventh within the European Union (Eurostat, 2017). In 2017, the city recorded over 6 million daily journeys, with 35.3% attributed to active modes of mobility, 40.1% to public transportation, and 24.6% to private vehicles. Among the trips classified as private transport, 67.7% were conducted using cars while motorcycles accounted for 29.8% (Barcelona, 2018). Unfortunately, mobility also leads to negative consequences, including congestion, pollution, accidents, fatalities from traffic incidents, and greenhouse gas emissions (Agency, 2019; Commission, 2021). To meet its new climate goals, the European Union has introduced a series of policy initiatives aimed at mitigating the harmful effects of cars and, at the same time, promoting the use of public transport (Council, 2014). In this context, accurate real-time traffic condition predictions using statistical and AI methods are crucial for road users, the private sector, and governments. For instance, identifying commuting patterns can inform decisions about where to strengthen public transportation services. Understanding mobility trends and urban design can also help determine the optimal number and placement of public charging stations for EVs, potentially boosting the adoption of these vehicles. For businesses, estimating their social and environmental impacts with the help of intelligent methods enables them to design distribution processes that minimize these impacts, which are often linked to economic costs (Reyes-Rubiano et al., 2020). Therefore, high-quality data are invaluable for modeling and solving optimization problems related to urban mobility (Faulin et al., 2018). Additionally, Commission (2021) notes a lack of sufficiently detailed indicators and related data on urban mobility (such as modal split, environmental impacts, congestion, and energy use). This gap exists because cities are not mandated to collect and provide such data. However, with the growing adoption of IoT technologies (e.g., intelligent traffic lights, road sensors, or sensors in trash and recycling containers) and the increasing awareness of the potential of open data portals, it is expected that data availability will continue to rise in the coming years. Traffic modeling and prediction are challenging due to the high non-linearity and complexity of traffic flow. Recently, ML and deep learning methods have begun to outperform traditional statistical models in traffic prediction tasks (Khan et al., 2023). This shift occurs because traditional methods are limited in their ability to make medium- to long-term predictions and often fail to account for the spatial and temporal dependencies in the

data (Yu et al., 2018).

5.2 Problem Modeling

The fundamentally complex and non-linear character of traffic dynamics makes accurately modeling and predicting traffic flow a substantial issue. Traffic data is highly variable and influenced by a variety of factors, including time of day and seasonality, as well as spatial dependencies (such as road network structure and local events). These qualities make it challenging to predict traffic flow using traditional statistical models, which frequently aim for simpler, more linear correlations. Traditional statistical methods, while helpful in making short-term predictions, frequently fail to estimate medium- to long-term trends. This problem comes mostly from their inability to fully capture the complex spatial and temporal correlations found in traffic data. As a result, these methods may not produce reliable predictions over long time periods especially in complicated urban areas where traffic conditions change quickly and are influenced by a variety of interdependent factors.

5.3 Methodological Approach

This section analyzes traffic levels in Barcelona using data from the Ajuntament de Barcelona's open data service, applying visualization techniques, time series analysis, and the XGBoost algorithm.

5.3.1 Visualization Techniques

Visual representation of traffic data plays a vital role in efficient city traffic management. It helps to understand the movement dynamics and reveal patterns related to traffic flow, social behavior, spatial geography, and economic trends (Chen, Guo, et al., 2015). Key applications of traffic visualization include identifying real-time traffic jams by monitoring traffic conditions, uncovering mobility patterns of vehicles and pedestrians, and optimizing route planning based on traffic density. Common visualization techniques include line charts, bar charts, heatmaps, histograms, and geospatial maps. Among these, geospatial maps are the most frequently used for traffic data visualization, with 2D representations being more prevalent in the literature than 3D visualizations (Clarinval et al., 2021). Although 2D geospatial maps effectively represent the spatial aspects of data, they fall short in capturing the temporal dimension, which is crucial for traffic data visualizations. Various approaches exist to represent spatial data with temporal components, with the most widely studied being the Space Time Cube (STC) model (Kraak,

2003). In an STC model, the spatial components are displayed on the x and y axes, while the temporal dimension is represented on the z axis. This approach is effective for small to medium-sized datasets but becomes challenging to interpret when dealing with large volumes of data. Another prevalent method to illustrate spatial and temporal dynamics involves the use of animated maps that showcase geospatial transformations over time. In this article, the HeatMapWithTime plugin from the Folium library (Story, 2020) is used for visualization. Heatmaps are a highly effective tool for visualizing geospatial data and conveying stories. This plugin enables the creation of interactive heatmap animations, allowing end users to customize the visual representation based on their specific needs. Potential interaction options include overview, zoom, pause, play, loop, and the ability to adjust playback speed for a selected time period. The heatmap enables the visualization of traffic density across various sections of the city throughout the day. Each dot on the heatmap represents a vehicle's location. By examining vehicle locations at different time intervals, potential mobility patterns and street traffic density can be identified. While the dataset used in this study (described in 5.4.1) does not include vehicle-specific data, it does provide information on street sections. Using this data, we can manually assign a certain number of vehicles to each street section based on traffic density, with higher-density sections assigned more vehicles and lower-density sections fewer vehicles.

5.3.2 Time Series Analysis

The predictive models employed include ARIMA models (Douc et al., 2014). The general form of an ARIMA (p ,d,q) model is expressed as follows:

$$Y_t = \alpha_1 Y_{t-1} + \cdots + \alpha_p Y_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} \tag{5.1}$$

where the parameters α are the autoregresive part (ar), and θ are the moving average part (ma). The models are built by section because the behavior is more stable. They are implemented using free software R (R Core Team, 2023) and the forecast package for R (Hyndman et al., 2008). Given the extensive use of these models across various applications over time, readers seeking more detailed information are encouraged to refer to Shumway et al. (2000).

5.3.3 eXtreme Gradient Boosting

XGBoost (Chen and Guestrin, 2016) is a highly scalable ML method for gradient boosting using decision trees. It shares similarities with other gradient boosting techniques by constructing a mathematical model to estimate y_i from the input $\boldsymbol{x}_i$. Specifically, XGBoost creates an ensemble $\mathcal{F} = \{f(\boldsymbol{x}) = w_{q(\boldsymbol{x})}\}$, where $q : \mathbb{R}^m \to T$ and $w \in \mathbb{R}^T$, consisting of K decision trees. Each f_k

corresponds to a specific tree q, defined by its structure (number of leaves T) and the associated set of weights w for the leaves.

With this setup, the output prediction is computed using an additive function involving K components, as follows:

$$\hat{y}_i = \phi(\boldsymbol{x}) = \sum_{k=1}^{K} f_k(\boldsymbol{x}_i),\ f_k \in \mathcal{F} \tag{5.2}$$

In this case, the input data $\boldsymbol{x}_i$ includes elements such as the geographic location of the section being evaluated, the timestamp of the data, and additional engineered features detailed later. Unlike traditional decision trees that primarily classify data, this model incorporates continuous values derived from the weights w_i assigned to each leaf across the ensemble of trees q.

To train the K trees in the model, a regularized objective function based on a convex loss is utilized:

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} l(\hat{y}_i, y_i) + \sum_{k=1}^{K} \Omega(f_k) \tag{5.3}$$

Here, l denotes a differentiable loss function that quantifies the difference between the target value $\hat{y}_i$ and the predicted value y_i, while Ω serves as a regularization term to penalize overfitting in the model. The regularization term is expressed as follows:

$$\Omega(f) = \lambda T + \frac{1}{2}\lambda ||w||^2 \tag{5.4}$$

Rather than relying on optimization techniques based in Euclidean space, XGBoost utilizes an additive training approach. In this method, at the t-th iteration, a unique function f_t is introduced for the i-th instance to minimize the objective function, which is expressed as follows:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(\boldsymbol{x}_i)) + \Omega(f_t) \tag{5.5}$$

Here, $\hat{y}^{(t)}$ represents the prediction at the given instance and iteration. This approach implies that f_t is incrementally added based on its contribution to improving performance across optimization instances and iterations. To facilitate optimization, the objective function is derived by expanding the loss function using a second-order Taylor series with respect to $\hat{y}_i$. It is expressed in terms of the first- and second-order gradient statistics g_i and h_i as follows:

$$\begin{aligned}\mathcal{L}^{(t)} &\simeq \sum_{i=1}^{n}\left[l(y_i,\hat{y}^{(t-1)}+g_i f_t(\boldsymbol{x}_i)+\frac{1}{2}h_i f_t^2(\boldsymbol{x}_i)\right]+\Omega(f_t)\\ &\text{where } g_i=\partial_{\hat{y}^{(t-1)}}l(y_i,\hat{y}^{(t-1)}) \text{ and}\\ &h_i=\partial^2_{\hat{y}^{(t-1)}}l(y_i,\hat{y}^{(t-1)}).\end{aligned} \tag{5.6}$$

$$\tilde{\mathcal{L}}^{(t)} \simeq \sum_{i=1}^{n}\left[g_i f_t(\boldsymbol{x}_i)+\frac{1}{2}h_i f_t^2(\boldsymbol{x}_i)\right]+\Omega(f_t) \tag{5.7}$$

Furthermore, the objective function can be expanded by incorporating its regularization term Ω for a leaf set I_j within a given structure $q(x_i)$, as expressed below:

$$\tilde{\mathcal{L}}^{(t)} \simeq \sum_{i=1}^{n}\left[g_i f_t(\boldsymbol{x}_i)+\frac{1}{2}h_i f_t^2(\boldsymbol{x}_i)\right]+\gamma T+\frac{1}{2}\lambda\sum_{j=1}^{T}w_j^2 \tag{5.8}$$

This formulation allows for the calculation of the optimal leaf weight w_j^* for leaf j and evaluates the overall fit of the structure $q(x)$ using the following:

$$w_j^* = -\frac{\sum_{i\in I_j} g_i}{\sum_{i\in I_j} h_i+\lambda} \tag{5.9}$$

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{\left(\sum_{i\in I_j} g_i\right)^2}{\sum_{i\in I_j} h_i+\lambda}+\gamma T \tag{5.10}$$

XGBoost utilizes a greedy algorithm that begins with a single leaf and incrementally adds branches at each iteration. This process uses the scoring function in Equation (5.10) to evaluate improvements in the overall tree structure q. The potential gain is calculated for an instance $I = I_L \cup I_R$, where I_L and I_R represent the sets of nodes on each branch following a split.

$$Gain = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i+\lambda}+\frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i+\lambda}-\frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i+\lambda}\right]-\gamma \tag{5.11}$$

5.3.4 Evaluation Metrics

The evaluation of the classification system is performed using the area under the Receiver Operating Characteristic (ROC) curve, combined with an Over to Rest (OrV) strategy. This approach is a widely accepted performance metric for ML models. In the OrV strategy, any misclassification is treated as an error, regardless of which two classes are confused. The Area Under the Curve (AUC) method is defined as follows:

$$\text{AUC} = \sum_i \left\{ (1 - \beta_i \times \Delta\alpha) + \frac{1}{2}[1 - \beta \cdot \Delta\alpha] \right\} \tag{5.12}$$

Here, α represents the probability of a false positive, while $1 - \beta$ denotes the probability of a true positive.

5.3.5 Shapley Additive Explanations

Shapley Additive Explanations (SHAP), grounded in game theory, is a tool used to interpret ML models (Lundberg and Lee, 2017). It evaluates the contribution of each variable to the model's output by iteratively introducing one variable at a time and calculating the expected value of the output function. By accounting for all possible variable orderings, SHAP determines the average contribution of each variable to the final prediction.

The average impact of variable x on model f is determined using the following formula:

$$g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i x'_i \tag{5.13}$$

where $x' \in \{0, x_i\}^M$ represents the count of input variables, and $\phi_i \in \mathbb{R}$.

SHAP, specifically Tree SHAP (Lundberg, Erion, et al., 2020), provides a detailed understanding of how variables influence tree-based ML models. It adheres to three key principles: local accuracy ensures that the function mapping x to x', denoted as $h_x(x')$, corresponds to the set of variables x, ensuring that the approximation of f aligns with its output; missingness asserts that variables not present in the input ($x'_i = 0$) have no effect on the output, leading to $\phi_i = 0$; and consistency guarantees that if a variable's contribution to the model increases or remains unchanged, its Shapley value does as well, irrespective of other variables. Tree SHAP evaluates the model using an input dataset X with dimensions $N \times M$, producing a matrix of SHAP values for each variable in every data point in X. This approach ensures consistent explanations for individual predictions while identifying contributions with corresponding sign indicators.

5.3.6 Variable Analysis

After the model has been trained, importance scores for each variable can be obtained. These scores represent the absolute contribution of each selected variable, averaged across all the trees that comprise the XGBoost model. The importance of each parameter is analyzed using three primary scoring systems:

- **Gain:** Represents the relative contribution of each variable to the model by summing

its contribution across all trees. It highlights the importance of a variable in generating predictions.

- **Cover:** Reflects the relative importance of a variable based on the number of observations associated with it. This is computed by summing the second derivative of the loss function for all training data points assigned to each node defined by the variable.
- **Frequency:** Indicates the proportion of times a variable appears in the trees of the model relative to the total number of trees.

The model is implemented using the standard Python 3.10.8 distribution (Van Rossum et al., 2009), along with its associated libraries for XGBoost implementation.

5.4 Computational Experiments

5.4.1 Description of the Dataset

The "Traffic state information by sections of the city of Barcelona" dataset is available in the Open Data Barcelona catalogue (opendata-ajuntament.barcelona.cat/en). Open Data Barcelona aims to make public resources accessible, allowing public data to be used for the common good. This dataset contains traffic data for 527 sections of Barcelona, collected since December 2017 and updated monthly. Data is recorded every 5 minutes, with traffic states ranging from 0 (no data) to 6 (cut off/closed). States include: 1 = very fluid, 2 = fluid, 3 = dense, 4 = very dense, and 5 = congestion. The data is obtained from sensors embedded in the asphalt (e.g., magnetic, infrared, and cameras).

To analyze the traffic information for March 2022, R version 4.0.3 (10 December 2020) and RStudio were used (R Core Team, 2023; Team, 2019). The dataset has five variables: section ID (road segment), data (year, month, day, hour, minute, second), current state, and expected state. With updates every 5 minutes, it includes $4,599,656$ rows. The data was transformed to ensure exact 5-minute intervals, introducing rows with a 0 (no data) where necessary. The distribution of the seven current traffic states is depicted in Figure 5.1. Congestion (0.7%) and 'no data' (42.9%) represent the lowest and highest proportions, respectively. The remaining states, ranked by proportion, are fluid (29.1%), very fluid (20.1%), dense (4.7%), very dense (1.7%), and cut off (0.8%).

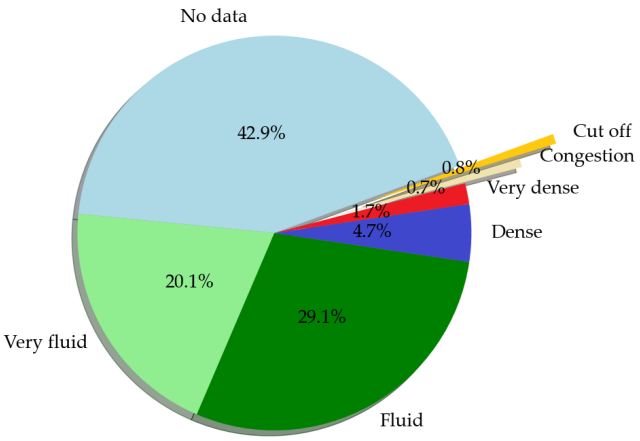


Figure 5.1: Traffic state distribution. Data source: "Traffic status information by sections of the city of Barcelona." (Open Data BCN)—March 2022.

5.5 Analysis of Results

5.5.1 Visualization Insights

Figure 5.2 illustrates the traffic state in Barcelona on Friday, 25 March 2022, at various time intervals. The lines represent street sections, with colors indicating traffic conditions: light blue for very fluid, dark blue for fluid, yellow for dense, orange for very dense, and red for congestion. As observed, the majority of the city's sections exhibit traffic states of 1 to 2 (very fluid to fluid). During rush hours (9:00 a.m.), more sections experience dense or higher traffic states (3–5) compared to non-rush hours (3:00 p.m.).

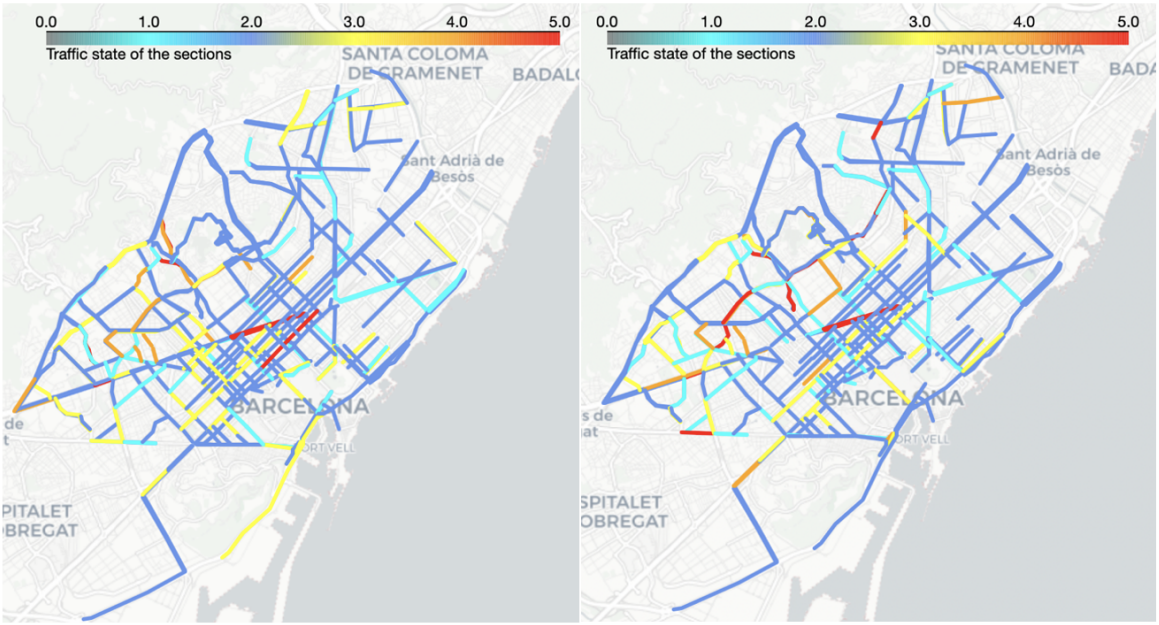


Figure 5.2: Geospatial traffic map. Traffic conditions at 9:00 AM (left) and 3:00 PM (right).

Figure 5.3 presents the traffic state for the same time intervals as Figure 5.2, but in the form of a heatmap. The top two subfigures display a wider area, while the bottom two subfigures focus on the central region. As observed, the denser areas in the heatmap (marked in orange) correspond to sections with higher traffic levels or regions with a greater concentration of street sections.

Figure 5.3: Heatmap traffic map. Traffic situations at 9:00 (left) and at 15:00 (right).

5.5.2 Time Series Analysis

The predictive models are evaluated using traffic data gathered from Barcelona in March 2022. Given the large number of monitored sections, it is essential to determine whether their performance is consistent. For this evaluation, ten sections from a single street are chosen. It is important to note that even though these sections belong to the same street, their traffic conditions may differ considerably, with variations seen between weekdays and weekends. Table 5.1 displays the outcomes of the time series models. The initial two columns represent the section and the corresponding fitted model, succeeded by five columns that list the coefficients. The final two columns detail the performance metrics, namely Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC). There is no need to differentiate the series since the models demonstrate stability. Table 5.2 reports the Mean Error (ME) and Root-Mean-Square Error (RMSE) for each model. Additionally, tests for independence (Box–Ljung test for residuals), homoscedasticity (Box–Ljung test for squared residuals), and normality (Shapiro–Wilk normality test) are applied, with their p-values displayed. None of the models satisfies the initial assumptions, confirming that these methods do not adequately capture the behavior of the data. In summary, the high non-linearity and complexity of traffic flow make classical methods ineffective for accurate predictions, as they fail to account for the spatiotemporal structure of the traffic data. Consequently, a ML method is applied in the next subsection.

Table 5.1: Output of time series by section.

Section	Model	ar1	ar2	ar3	ma1	ma2	AIC	BIC
1	ARIMA(2,0,2)	1.568	−0.655		−0.839	−0.315	934.270	961.940
2	ARIMA(3,0,1)	0.067	0.824	−0.211	0.965		105.650	133.320
3	ARIMA(2,0,1)	1.799	−0.872		−0.910		−39.040	−15.980
4	ARIMA(2,0,2)	1.668	−0.742		−0.639	−0.193	1126.190	1153.860
5	ARIMA(2,0,2)	1.506	−0.583		−0.622	−0.249	1469.640	1497.310
6	ARIMA(1,0,1)	0.638			0.259		1172.080	1190.520
7	ARIMA(2,0,1)	1.785	−0.850		−0.900		69.040	92.100
8	ARIMA(1,0,0)	0.589					1138.790	1152.620
9	ARIMA(2,0,0)	0.788	−0.227				1541.360	1559.800
10	ARIMA(2,0,1)	1.778	−0.838		−0.867		−33.030	−9.970

Table 5.2: Time series error outputs by section.

Section	Model	ME	RMSE	Independence (p-Value)	Homoscedasticity (p-Value)	Normality (p-Value)
1	ARIMA(2,0,2)	5.27×10^{-5}	0.449	**0.7556**	$<2.2 \times 10^{-16}$	$<2.2 \times 10^{-16}$
2	ARIMA(3,0,1)	0.0008	0.257	3.5×10^{-11}	**0.078**	$<2.2 \times 10^{-16}$
3	ARIMA(2,0,1)	−0.0005	0.234	1.594×10^{-11}	4.331×10^{-6}	$<2.2 \times 10^{-16}$
4	ARIMA(2,0,2)	−0.0001	0.511	0.00015	5.707×10^{-6}	$<2.2 \times 10^{-16}$
5	ARIMA(2,0,2)	0.0007	0.644	**0.2001**	3.201×10^{-10}	$<2.2 \times 10^{-16}$
6	ARIMA(1,0,1)	0.0009	0.529	5.668×10^{-9}	2.065×10^{-14}	$<2.2 \times 10^{-16}$
7	ARIMA(2,0,1)	−0.0001	0.251	5.951×10^{-8}	**0.093**	$<2.2 \times 10^{-16}$
8	ARIMA(1,0,0)	−0.0002	0.518	**0.1294**	1.208×10^{-8}	$<2.2 \times 10^{-16}$
9	ARIMA(2,0,0)	0.0003	0.678	1.056×10^{-5}	1.762×10^{-8}	$<2.2 \times 10^{-16}$
10	ARIMA(2,0,1)	3.99×10^{-5}	0.235	4.441×10^{-16}	1.478×10^{-7}	$<2.2 \times 10^{-16}$

5.5.3 eXtreme Gradient Boosting Results

To simplify the analysis of parameter importance, we categorize the various densities derived from the original dataset into classification labels: 0 = very fluid, 1 = fluid, 2 = dense, 3 = very dense, and 4 = congestion. The dataset, structured as shown in Table 5.3, covers the entirety of 2019. For evaluation, the dataset is divided into separate sets for training and testing. Specifically, 70% of the data is allocated for training, while the remaining 30% is reserved for parameter evaluation. Additionally, data from January to March 2022 are processed for final analysis and predictions, as detailed in the model performance section.

Table 5.3: Dataset description for the XGBoost model.

Variable	Description	Type	Range
Status	Current traffic status of the section.	number	0 to 4
FromNorth	Starting Latitude (North).	number	2099 to 2222
FromWest	Starting Longitude (West).	number	41,338 to 41,450
ToNorth	Ending Latitude (North).	number	2100 to 2223
ToWest	Ending Longitude (West).	number	41,338 to 41,449
DailyHour	Measurement hour.	number	0 to 23
DailyMinute	Measurement minute.	number	0 to 60
Weekday	Weekday of the measurement.	number	1 to 7
DayMonth	Measurement day of the month.	number	1 to 31
Holiday	Boolean value representing the existence of a holiday on that day.	Boolean	False or True
Number of records: 24,533,298			

To enhance the performance of XGBoost, parameter tuning is performed using grid search combined with five-fold cross-validation. This approach explores multiple parameter configurations within reasonable ranges, as outlined in Table 5.4.

Table 5.4: Parameter matrix for XGBoost.

Parameter	**Selected value**	**Options**
max_depth	10	5, 7, 9, 10, 11
eta	0.3	0.1, 0.2, 0.3, 0.4
gamma	1	0.5, 1, 1.5, 2, 5
subsample	1	0.6, 0.8, 1
objective	multi:softmax	-
num_class	5	-

The AUC values shown in Figure 5.4 highlight the consistently robust performance of the model, with AUC values surpassing 80% across all ROC curves. Notably, the model performs exceptionally well in extreme cases, achieving a true positive rate above 85%. However, a slight drop in performance is observed for intermediate classes. On average, the model achieves an accuracy rate of 74.48% on the 2019 dataset. Thus, it can be concluded that this model accurately represents the importance of the parameters, demonstrating a significant correlation between the true labels and the predicted ones. Computing time is also considered a key aspect of model performance after training, as shown in Table 5.5. For reference, the average processing times for predicting values over a day, week, or month of data are provided. The training and execution of this model were conducted on an 11th Gen Intel(R) i5-1135G7 CPU running at 2.40 GHz, with 16 GB of RAM and Windows 10 Pro 22H2 (64-bit) as the primary operating system. It is concluded that this technology is well-suited for dynamic data interpretation and can be effectively implemented as a predictive tool by the Barcelona council.

For this model, Figure 5.5a displays the parameters ranked by frequency of appearance. Notably, *FromNorth*, *DailyHour*, and *DayMonth* account for the majority of the relevance in classifying the data rows, with *FromWest* also showing relatively high importance. These findings align with our initial hypotheses, as geographical location, time, and day of the month significantly influence and help explain the current traffic status. Conversely, variables such as *DailyMinute* and *Weekday* appear to have a lesser impact on the model.

Additionally, analyzing not only the frequency of appearance but also the cover (the relative number of observations associated with each variable in prediction) in Figure 5.5b and the gain (the relative contribution of each variable to the prediction across all trees) in Figure 5.5c, it becomes evident that variables such as *ToWest* and *ToNorth*, despite having a lower frequency of appearance compared to others, exhibit significant gain and cover values. This indicates that

these variables also play a key role in defining the model's behavior and have a notable impact on its outcomes.

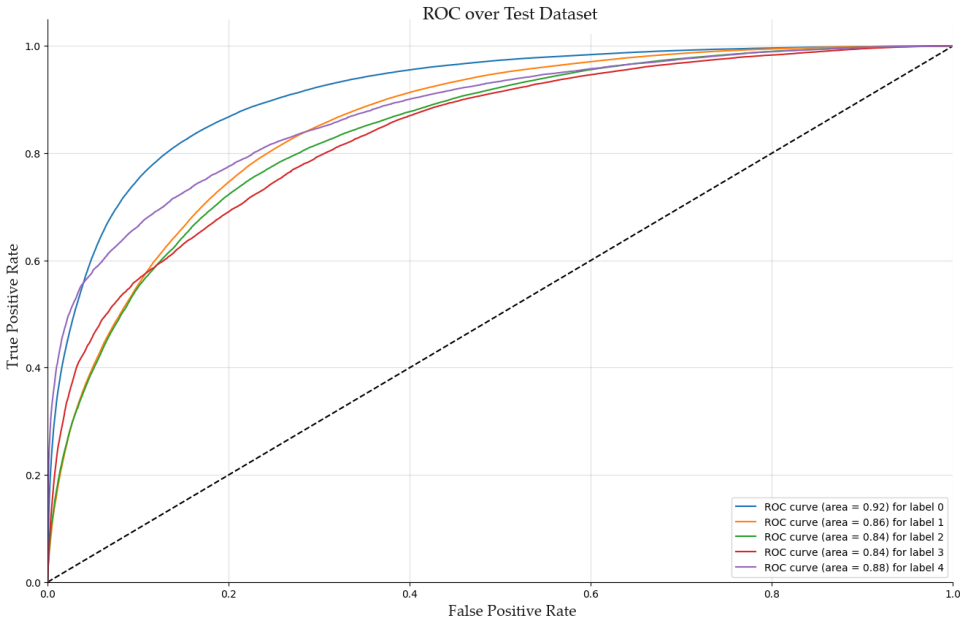
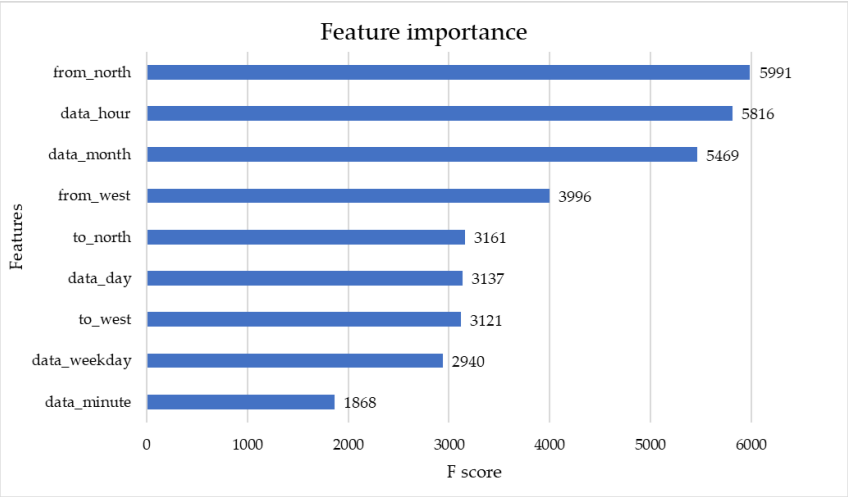


Figure 5.4: AUC for the XGBoost model.

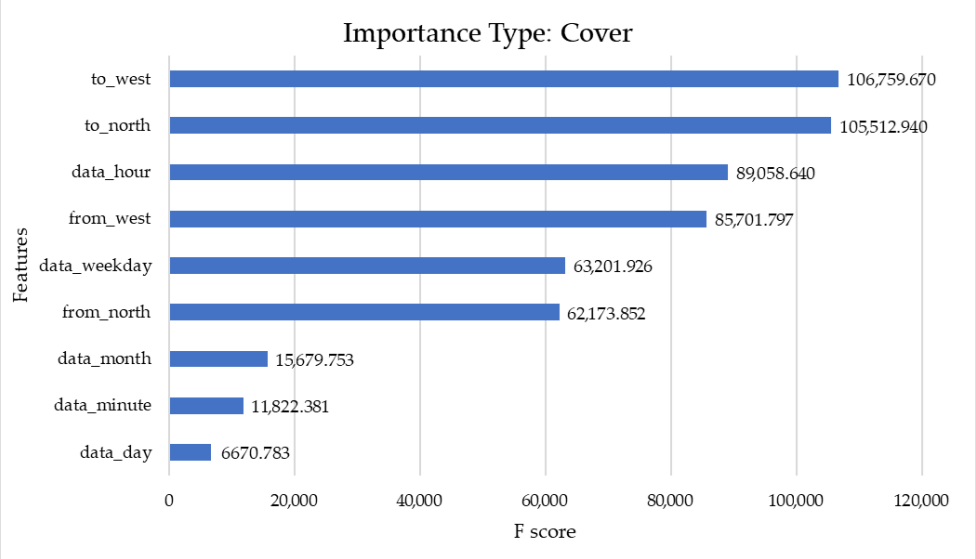
Table 5.5: Computing times.

Time Lapse	**Average Times after 50 Runs (s)**
One day	0.037577
One week	0.229319
One month	1.363848

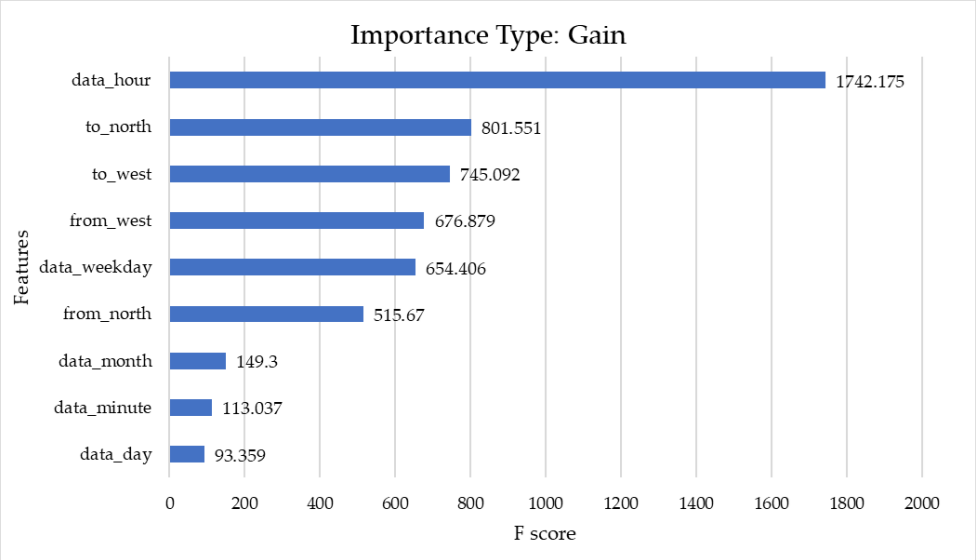


(a) Frequency

Figure 5.5: *Cont.*



(b) Cover



(c) Gain

Figure 5.5: XGBoost model variable analysis.

As shown in Figure 5.6, classes 0, 1, and 2 derive most of their explainability from *DayMonth*. In contrast, the remaining classes primarily attribute their explanation impact to coordinates such as *ToWest* and *FromWest*.

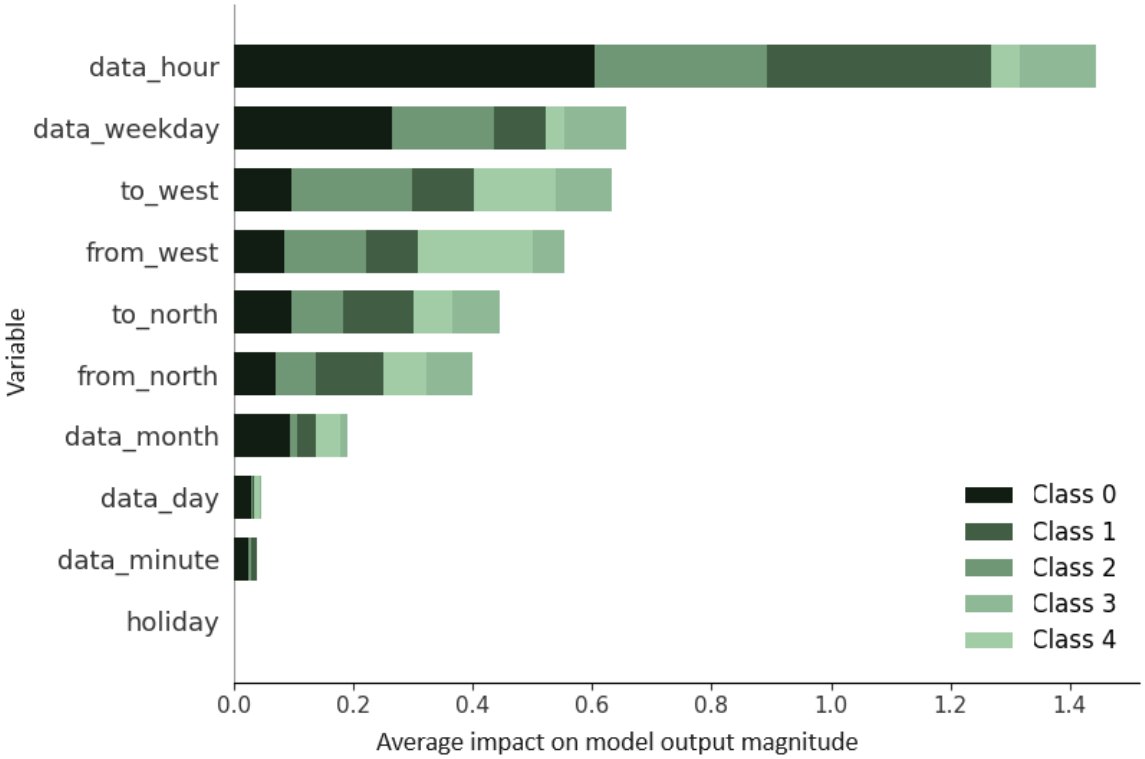
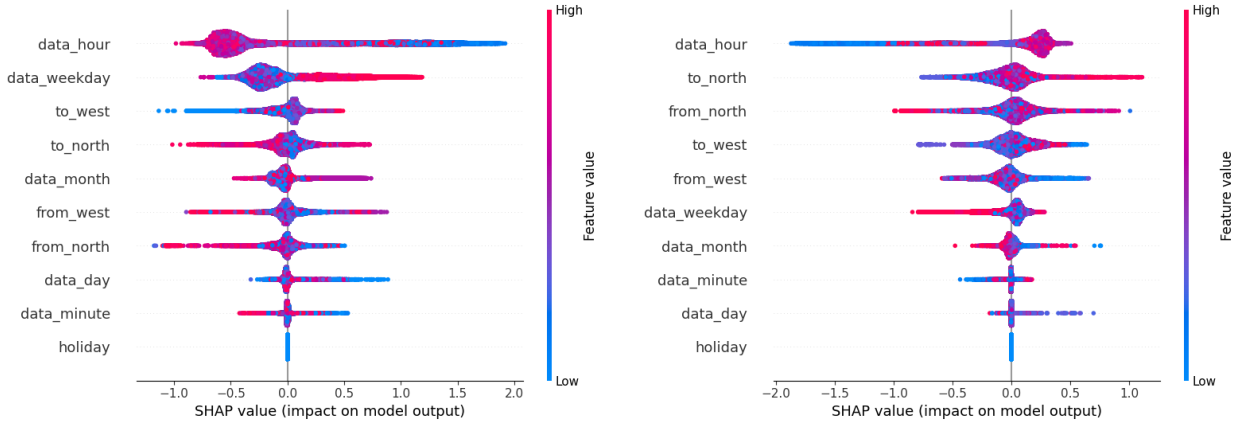


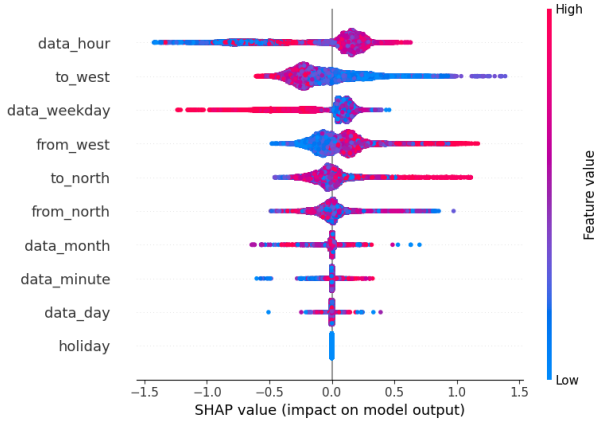
Figure 5.6: SHAP variable importance plot by classes.

Lastly, Figure 5.7 depicts the model's influence across different variable combinations for each class. For class 0, the hour and day of the week have the highest impact. In class 1, the hour and *North* coordinates are the most significant. For class 2, the hour and *West* coordinates play the largest role. In class 3, the hour and day of the week coordinates are the most influential. Finally, in class 4, all four coordinates collectively have the greatest impact on the model.

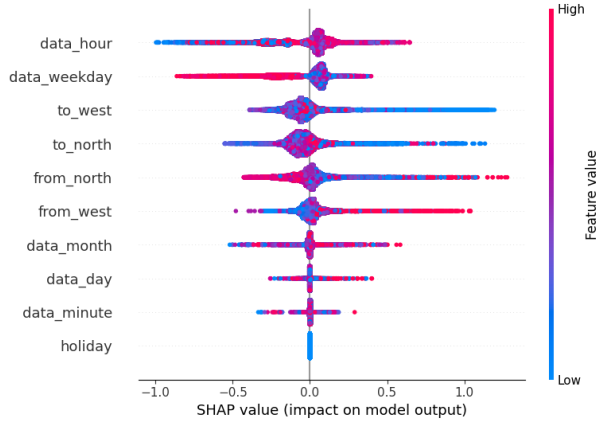


(a) Class 0

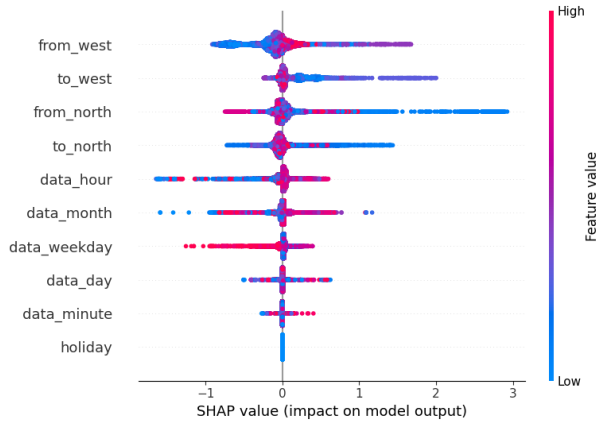
(b) Class 1



(c) Class 2



(d) Class 3



(e) Class 4

Figure 5.7: Predicted class impact on model output.

Historically, ARIMA models have been widely used for traffic prediction because they are simple to implement and offer a comparatively high level of accuracy (Ahmed et al., 1979). The

complex non-linear spatio-temporal relationships, along with external influences like weekends, holidays, and weather conditions, pose challenges that exceed the limitations of ARIMA. Our findings reveal that none of the constructed ARIMA models meet the assumptions of independence, homoscedasticity, and normality. As a result, traditional methods face difficulties in generating accurate predictions and are poorly suited to the spatio-temporal nature of traffic data. To address this issue, an XGBoost model is utilized for traffic prediction. XGBoost effectively manages non-linear variables, requires minimal prior knowledge of traffic patterns, and consistently delivers strong performance, with area under ROC curves surpassing 80%. Its implementation enables the analysis of variable importance and improves interpretability through SHAP. In conclusion, while visualization techniques support real-time traffic density analysis and interactive animations for better pattern recognition, the XGBoost model offers interpretable outputs, insights into variable importance, and precise predictions. Precisely forecasting traffic trends allows policymakers to make more informed decisions about transportation infrastructure, such as the location and number of roads, highways, and public transit options. This facilitates the optimization of limited resources and enhances the efficiency of the transportation network (Zhu et al., 2018). Accurate traffic prediction plays a vital role in reducing congestion and emissions by helping decision-makers implement strategies to alleviate traffic and enhance air quality. For example, they can adopt demand-management measures such as variable tolls or congestion pricing. These measures are designed to encourage drivers to choose alternative modes of transportation or travel during off-peak hours. Additionally, traffic prediction can identify high-risk areas for accidents, enabling the implementation of strategies to reduce accident likelihood and enhance road safety. By anticipating and identifying the variables that influence traffic forecasts, this chapter's contribution supports decision-makers in making swift and effective decisions to improve traffic management and reduce congestion (Habtemichael et al., 2016). This section highlights the key variables influencing the forecast, including geographical location(particularly when the section originates in the North direction)along with time and the day of the month.

Chapter 6

Dynamic Reactive Automated Guided Vehicles Task Assignment

This chapter[1] presents a heuristic-based approach to optimize task scheduling for AGVs, focusing on improving efficiency in dynamic industrial environments where disruptions are inevitable. The core methodology involves three key components: (i) predicting circuit delays through a dynamically generated delay matrix that accounts for potential interruptions; (ii) constructing an initial itinerary based on these predictions to minimize the total task time; and (iii) dynamically updating the schedule in real-time as new data regarding delays and disruptions become available.

This chapter builds on the idea that effective task scheduling is vital for improving AGVs performance, especially in smart factories where disruptions (such as interactions with other vehicles, equipment malfunctions, or human operators) can significantly delay task completion. The approach introduced here not only seeks to minimize these delays but also emphasizes the importance of agile optimization strategies. By continually refining task sequences through real-time data updates, AGVs operations can become more adaptive and resilient to dynamic changes in the environment.

[1]The contents of this chapter are based on the following work:

- Martin, Xabier A, Sara Hatami, Laura Calvet, **Mohammad Peyman**, and Angel A Juan (2023). *"Dynamic reactive assignment of tasks in real-time automated guided vehicle environments with potential interruptions"*. In: Applied Sciences 13.6, p. 3708.

6.1 Problem Description

As noted by Zhang and Chen (2020), Industry 4.0 introduces automation to industrial systems through computerized processes and advanced machinery. This shift is driven by technologies like robotics, AI, and autonomous vehicles, alongside IoT, cloud computing, data analytics, and ML. These technologies enable efficient data collection and control of production systems, helping forecast events that might negatively impact productivity. Effective plant management requires considering various internal and external factors, with process automation being crucial. Smart factories utilize sensors, embedded software, and robotics, including AGVs, which can transport both small and large loads. AGVs operate autonomously, without human intervention, and have become integral to Industry 4.0 (Javed et al., 2021). Though AGVs have been in use for over 25 years (Vis, 2006), their popularity has surged due to reduced costs and improved productivity. Key benefits of AGVs include enhanced safety, reduced damage, efficient operations in harsh environments, and lower maintenance costs compared to conventional vehicles (Bechtsis et al., 2017). Each AGVs consists of a vehicle, guidance, control, and communication systems (Zhan et al., 2018). The control and communication systems allow AGVs to navigate and adapt to dynamic environments, coordinating with other vehicles and machinery to avoid collisions. Wireless communication helps AGVs react to obstacles, either stopping or re-routing as needed. These adjustments can cause delays, affecting the plant's overall efficiency (Zhao et al., 2020).

This chapter focuses on optimizing task scheduling for AGVs in dynamic environments, minimizing total task completion time. Production managers must adapt task orders to account for varying travel and processing times due to disruptions. The optimization problem is illustrated in Figure 6.1. In a warehouse, products start at a depot station and are moved by an AGVs to various workstations for processing. The AGVs must minimize delays caused by dynamic conditions, reassigning tasks to optimize the overall workflow. This problem can be interpreted as an extended (dynamic) variant of the single-machine scheduling problem, where the AGVs serves as the single machine. As single-machine scheduling problems are generally classified as NP-hard (Lenstra, Kan, and Brucker, 1977), solving the dynamic version in this context may necessitate the use of a fast and adaptive heuristic method to enable real-time decision-making with dynamic data inputs. To tackle this challenge, we propose an 'agile' optimization algorithm (Martins, Tarchi, et al., 2022), specifically designed for the mobile robotics AGVs problem. This algorithm is a highly efficient reactive heuristic aimed at minimizing the total time required for an AGVs to complete all assigned tasks in a dynamic environment with potential disruptions. The reactive approach employs a dynamic matrix that incorporates these delays. The main contributions of this chapter are as follows: (i) the introduction of a single AGVs task order determination method that accounts for unexpected interruptions to minimize total time or

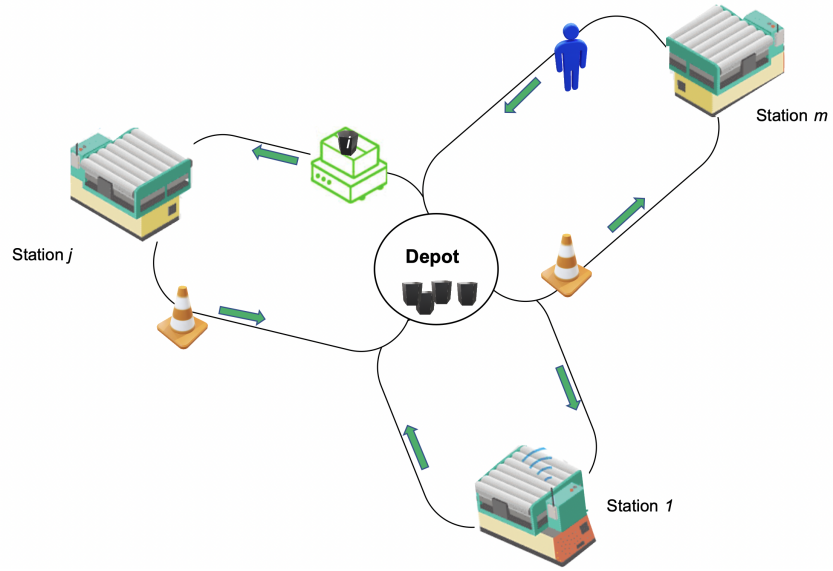


Figure 6.1: Visualization of the AGV optimization problem.

makespan; (ii) the application of data analytics techniques to estimate AGVs interruptions and the associated delays for each determined path over specific time spans; and (iii) the development of a reactive heuristic (Raba et al., 2020; Tordecilla, Copado-Méndez, et al., 2021), which examines the task permutations generated using three different approaches.

6.2 Problem Modeling

Consider a warehouse configuration in which each product $i \in I$ is required to be processed at various workstations $j \in J$. The workday concludes at a final time $t_f > 0$. Initially, all products are located at a depot station, 0. An AGVs must transport each product $i \in I$ (one at a time) from depot 0 to each workstation $j \in J$. After processing at workstation j, the AGVs must return the product i to depot 0. For a product $i \in I$ and a workstation $j \in J$, the total time required for product i to complete a circuit from $0 \to j \to 0$, denoted by $T_{ij} \geq 0$, is the sum of the travel and processing times under ideal conditions, $t^*_{ij} > 0$, and the sum of the actual delays encountered during the circuit, $D_{ij} \geq 0$. In other words, $T_{ij} = t^*_{ij} + D_{ij}$, where D_{ij} is a time-dependent function, $D_{ij} = D_{ij}(t)$, with $t \in [0, t_f)$. It is important to note that delay times are dynamic, meaning they depend on the system conditions at each moment in time, $t \in [0, t_f)$. This dynamic nature of delays is what makes the optimization problem non-trivial. Additionally, completing any single circuit $0 \to j \to 0$ should not exceed a user-defined time threshold $t_0 > 0$. Given these constraints, the objective is to minimize the total time required for the AGVs to complete all assigned tasks. The problem can be mathematically defined as follows:

$$\min \sum_{i \in I} \sum_{j \in J} T_{ij} \tag{6.1}$$

subject to:

$$T_{ij} = t^*_{ij} + D_{ij} \qquad \forall i \in I, \forall j \in J \tag{6.2}$$

$$D_{ij} = D_{ij}(t) \qquad \forall i \in I, \forall j \in J, \forall t \in [0, t_f] \tag{6.3}$$

$$T_{ij} \leq t_0 \qquad \forall i \in I, \forall j \in J \tag{6.4}$$

6.3 Methodological Approach

The suggested methodology employs a heuristic-based strategy to address the task scheduling issue of dynamic AGVs. We concentrate on two straightforward yet efficient and incredibly rapid heuristics. These heuristics operate on the premise that we can predict the anticipated delay for each circuit at any point in time, with the delay determined by existing system conditions. This is achieved through the creation of a dynamic matrix of expected delays, which utilizes historical data to project the delay for a particular circuit based on current environmental factors. Essentially, this dynamic matrix offers revised delay estimations for every potential circuit at the present moment. Algorithm 1 presents the pseudo-code for our first heuristic, which is a non-reactive approach.

Algorithm 1 Non-Reactive Heuristic for Dynamic Delays

```
1: Data: List of tasks T, and matrix of expected delays D
2: Result: Permutation of completed tasks P
3: P ← ∅
4: t_c ← 0
5: while T ≠ ∅ do
6:     T ← estimateDelays(T, D, t_c)
7:     T ← sort(T)
8:     t ← selectTask(T)
9:     P ← P ∪ {t}
10:    t_c ← updateSystem(t_c, t)
11: end while
12: return P
```

The non-reactive heuristic algorithm requires the following input parameters: (i) the list of tasks T that a specific AGVs must complete, and (ii) the dynamic matrix of expected delays D. Each task in the list consists of a product-workstation pair, along with the time required for the AGVs to complete the corresponding circuit. The dynamic matrix provides the estimated delays for each circuit at different times of the day. The non-reactive heuristic operates as follows: initially, an empty permutation of tasks P is created (line 3). The tasks from the list will be added to this permutation in the order in which the assigned tasks are completed. Additionally, the system conditions monitor the current time t_c, which is initialized to the starting time $t = 0$. The list of tasks, T, is then iterated through until all tasks are completed (lines $5 - 11$). During each iteration, the delays for each circuit are predicted based on the system conditions (line 6). In particular, the delay values for each circuit are obtained from the dynamic matrix of estimated delays, using the current time t_c. Furthermore, the estimated delay for each circuit is added to the time required for each task in the list to complete the specific circuit it needs to traverse. This allows us to calculate the total estimated time for the AGVs to complete each assigned task. The tasks are then sorted in ascending order of their estimated total time (line 7). The task with the lowest delay t is then extracted from the list (line 8). The task is then

appended to the end of the list of completed tasks (line 9). After the task is added, the system conditions are updated by incorporating the estimated total time required to complete the task into the variable that tracks the current time (line 10). Once all tasks have been processed, the generated sequence of completed tasks is returned (line 12).

The non-reactive heuristic presented in this discussion selects the task associated with the minimal estimated delay for the circuit, utilizing predictions derived from past data. Nevertheless, the real delays experienced during circuit traversal can vary from these initial estimates. The subsequent heuristic addresses this discrepancy by adopting a reactive strategy: it takes into account the prevailing system conditions, which are refreshed following the completion of each circuit.

Algorithm 2 presents the pseudo-code of our reactive heuristic.

Algorithm 2 Reactive Heuristic for Dynamic Delays

1: **Data: List of tasks T, and matrix of expected delays D**
2: **Result: Permutation of completed tasks P**
3: $P \leftarrow \emptyset$
4: $t_c \leftarrow 0$
5: **while** $C \neq \emptyset$ **do**
6: $T \leftarrow \text{estimateDelays}(T, D, t_c)$
7: $T \leftarrow \text{sort}(T)$
8: $t \leftarrow \text{selectTask}(T)$
9: $t \leftarrow \text{executeTask}(t)$
10: $P \leftarrow P \cup \{t\}$
11: $t_c \leftarrow \text{updateSystem}(t_c, t)$
12: **end while**
13: **return** P

The reactive heuristic approach, which takes the same input parameters as the non-reactive heuristic for dynamic delays, operates as follows. First, an empty permutation of tasks P is created (line 3). The system conditions also track the current time t_c, initialized to the starting time $t = 0$. Next, the list of tasks to be completed, T, is iterated through until all tasks are processed (lines $5 - 11$). In each iteration, the delays for each circuit are estimated based on the current system conditions. The list of tasks is then sorted in ascending order of the estimated delays (lines $6 - 7$). The first task from the newly sorted task permutation t is selected and executed by the AGVs (lines $8 - 9$). The expected total travel time is then revised based on the actual time the AGVs spends completing the circuit. In other words, we calculate the real total time for the AGVs to finish task t, factoring in the actual delays encountered during the journey. Subsequently, the task is appended to the end of the list of completed tasks (line 10). Finally, the system conditions are updated by adding the actual time required to complete the task to the system variable that tracks the current time (line 10), rather than using the estimated time. Once all tasks in the list have been completed, the generated sequence of completed tasks is returned (line 12).

It is important to note that the reactive heuristic, like the non-reactive heuristic, selects the task with the lowest estimated delay at each iteration. However, the reactive heuristic differs by updating the system conditions with the actual total time the AGVs requires to complete the circuit. Additionally, it recalculates the permutation of remaining tasks based on the current system conditions in subsequent iterations. This allows the reactive heuristic to adapt to potential disruptions.

6.3.1 An Illustrative Example of our Reactive Approach

This section presents a simple example that demonstrates how the proposed reactive algorithm solves the problem iteratively, with new data being updated in the system at the end of each round trip (i.e., each time the AGVs returns to the depot from a station). Figure 6.2 illustrates a toy example with 2 products and 2 workstations. Let $f_{ij}(t)$ represent a function that estimates the actual round trip time for assigning product i to station j at time $t \in [0, t_f)$, where t^t_{ij} denotes the round trip time, and t_f is the end time of the working day. The main idea here is as follows: (i) we will first use $f_{ij}(t)$ to create an initial assignment plan based on the estimated travel times; and (ii) after each round trip when the AGVs returns to the depot, since the actual times may differ from the estimated ones, we will recompute the remaining assignments using $f_{ij}(t)$ again. In other words, each time the AGVs passes through the depot (and as long as there are pending tasks), the current time $t_c > 0$ will be noted, and the previous plan will be revised (from t_c to t_f), incorporating the new estimates provided by $f_{ij}(t)$.

Thus, the figure illustrates an initial assignment plan (E1) where: (i) product 1 (P1) is first sent to station 2 (S2) and then returned to the depot (0) once processed; (ii) product 2 is delivered to station 1 and then returned to the depot; (iii) product 2 is then sent to station 2 and returned to the depot; and (iv) product 1 is sent to station 1 and brought back to the depot. If the estimated travel times were completely accurate (i.e., without any delays), this plan would be completed by time t_{E1}. However, as the plan is executed, unforeseen changes in the estimated travel times (typically in the form of delays) may occur, as seen in the real execution plan R1, potentially resulting in a revised estimated makespan t_{R1}. At this point, once product P1 has returned to the depot, we cannot alter the fact that product 1 has already been processed at station 2. However, we can revise our initial plan by considering the current time t_c and $f_{ij}(t)$. This leads to the creation of plan E2. It is important to note that, at least theoretically, plan E2 can improve the makespan associated with R1. This iterative process of comparing the estimated travel times with the actual ones, and adjusting future trips accordingly, continues until all products have been processed at all stations. In our scenario, this results in a final makespan t_{R3}, which, although not as optimal as t_{E1}, is certainly better than t_{R1} due to our ability to reactively adjust the plan based on the current system conditions (current time each

time the AGVs visits the depot and the estimation function, which depends on the time t when a trip is initiated). It is important to note that this is a simplified example, but in a real-world situation where the number of products n is large and the number of stations m also increases, the solution space grows significantly, leading to combinatorial explosion. Furthermore, when multiple AGVs and depots are involved, the problem becomes even more complex.

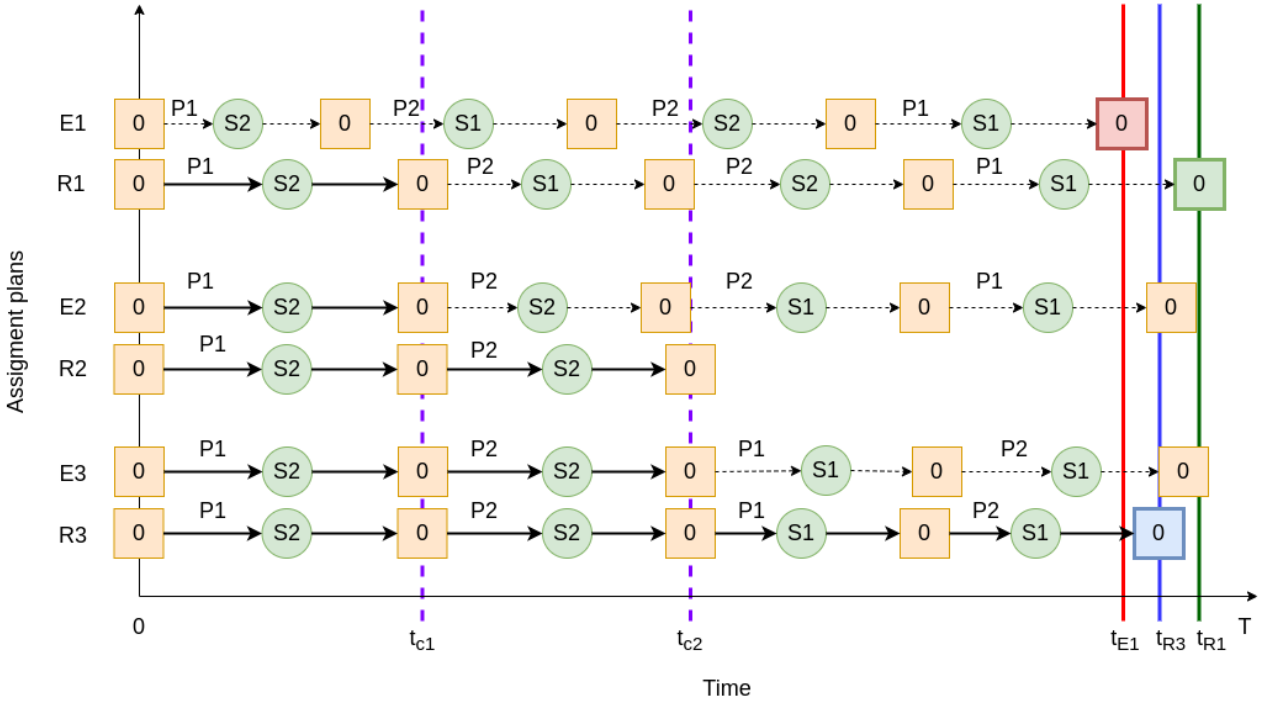


Figure 6.2: Illustration of the proposed algorithms.

6.4 Computational Experiments

The proposed heuristic methods were implemented using Python 3.10 (Van Rossum et al., 2009) and tested on a workstation equipped with a multi-core Intel Xeon E5-2630 v4 processor and 32 GB of RAM. The rest of this section outlines the computational experiments conducted to validate and demonstrate the effectiveness of our reactive heuristic. First, we describe the test instances used in the experiments. Then, we provide a detailed explanation of the three different approaches employed to test the instances, as well as the method used to calculate the actual delays of the circuits. For the computational experiments, each generated instance was executed 10 times per approach. Each run was conducted with a different seed for the random number generator used to simulate the actual delays of the circuits (based on historical data), and the averages are reported for different levels of dynamism. Specifically, we tested with low, medium, and high levels of dynamism.

6.4.1 Test Instances

As there are no publicly available benchmark instances for the problem we have addressed, we created a set of instances to evaluate the performance and quality of the proposed algorithm. For testing, we utilized real historical data from a company, specifically a single dataset collected over several days by an AGVs. This dataset contained various variables, including date, time, circuit, destination node, and location (X and Y coordinates). The dataset records the starting time of the AGVs through date and time, which are updated every second from 5 a.m. to 11 p.m. over a week. It includes six circuits, each linked to a particular task. During these circuits, the AGVs visits different nodes marked as destination points, with the X and Y coordinates specifying the AGVs position at any given moment. To support our methodology, we identified key features such as distance and filtered out instances where the distance was zero, which allowed us to capture the times and locations where the AGVs halted due to certain conditions. We then calculated the delay time per hour for each circuit and constructed a dynamic matrix table to feed into our approach. This matrix contains the average delay time for each circuit pair per hour, designed to execute a specific task. We defined four different scenarios: *small*, *medium*, *large*, and *very large*, representing cases where three stations handle 30, 60, 120, and 240 products, respectively. Additionally, the travel time without delays for each circuit is given in seconds, and the AGVs starts its operation at 5 a.m.

6.4.2 Test Approaches

The three methods evaluated in our computational experiments are outlined below:

- *FIFO*: In this approach, the task list is processed in a First-in, First-out (FIFO) sequence without considering the estimated delays for each circuit. After iterating through the tasks, the actual delays for the FIFO task order are calculated.
- *Non-Reactive*: This method generates the task sequence using the non-reactive heuristic for dynamic delays discussed earlier. The task with the lowest predicted delay for its circuit is added to the task sequence at each iteration. Then, the actual delays for the task sequence are computed.
- *Reactive*: In this approach, the task sequence is created using the reactive heuristic algorithm described in Section 6.3. A new sequence of remaining tasks is generated along with their corresponding actual delays, and the system is updated with the actual time taken to traverse each circuit.

In our numerical experiments, we use white noise to simulate the actual delay values that the AGVs will experience when traversing each circuit. Specifically, each white noise sample follows a normal distribution with a mean of zero, and this sample is added to the estimated delays to generate the actual delay values. In other words, the actual delays are modeled by adding a white Gaussian noise sample to the estimated delays. In the course of the experiments, samples are generated with a mean of $\mu = 0$ and a standard deviation of σ. The standard deviation σ allows us to adjust the dynamism in the experiments, with lower σ values indicating less dynamism and higher σ values indicating greater dynamism. For our experiments, we set σ to 100, 200, and 300 to represent low, medium, and high dynamism levels, respectively.

6.5 Analysis of Results

The results from the computational experiments were analyzed using R (R Core Team, 2022). Tables 6.1 to 6.3 display the outcomes for varying levels of dynamism, ranging from low to high. The first column lists the instances, while the subsequent columns show the total time (in seconds) taken by the AGVs to complete all assigned tasks for the three approaches tested. We begin by presenting the results for the FIFO approach, where tasks are processed in a FIFO manner. The next section of the table shows the results for the non-reactive approach. The first column indicates the total time required to complete the tasks in the order determined by the non-reactive heuristic for dynamic delays, and the second column compares Average FIFO Solutions (OBF) with Average Non-Reactive Solutions (OBN). The last section displays the results for the reactive approach, with the first column showing the total time when tasks are completed based on the reactive heuristic's order, and the second column comparing OBF with Average Reactive Solutions (OBR). Finally, the average values are shown in the last row.

Table 6.1: Test results for low-dynamism instances.

Instance	FIFO Approach	Non-Reactive Approach		Reactive Approach	
	OBF [1]	OBN [2]	GAP(%) [1-2]	OBR [3]	GAP(%) [1-3]
Small	86735.7	82403.1	-5.00	82236.4	-5.19
Medium	173105.4	166014.1	-4.10	164528.6	-4.95
Large	346699.7	335395.6	-3.26	328056.4	-5.38
Very large	691926.7	677652.9	-2.06	655090.7	-5.32
Average:	324616.9	315366.4	-3.6	307478.0	-5.2

Table 6.2: Test results for medium-dynamism instances.

Instance	FIFO Approach	Non-Reactive Approach		Reactive Approach	
	OBF [1]	OBN [2]	GAP(%) [1-2]	OBR [3]	GAP(%) [1-3]
Small	87921.7	83161.8	-5.41	82761.6	-5.87
Medium	175289.6	168075.4	-4.12	165001.2	-5.87
Large	351553.2	338148.2	-3.81	329533.5	-6.26
Very large	699450.7	681942.1	-2.50	656655.8	-6.12
Average:	328553.8	317831.9	-4.0	308488.0	-6.0

Table 6.3: Test results for high-dynamism instances.

Instance	FIFO Approach	Non-Reactive Approach		Reactive Approach	
	OBF [1]	OBN [2]	GAP(%) [1-2]	OBR [3]	GAP(%) [1-3]
Small	90084.8	84662.7	-6.02	83959.6	-6.80
Medium	179177.4	171094.5	-4.51	166810.5	-6.90
Large	360197.0	344949.7	-4.23	334807.1	-7.05
Very large	716942.5	690075.2	-3.75	665469.4	-7.18
Average:	336600.4	322695.5	-4.6	312761.6	-7.0

As anticipated, all the percentage gaps are negative, signifying that both the non-reactive and reactive approaches outperform the FIFO method. The objective function value grows as the instance size increases, as a larger number of assigned tasks demands more time for the AGVs to complete them. The percentage gaps between the solutions obtained using the non-reactive and reactive approaches and the FIFO approach are typically large in absolute terms. Specifically, the reactive approach shows the greatest percentage gap compared to the FIFO approach. This indicates that the reactive approach is the fastest in completing the assigned tasks, positioning it as the most efficient method for solving the given problem.

The barplot in Figure 6.3 illustrates the average gaps for the non-reactive and reactive approaches compared to the FIFO approach, taking into account different levels of dynamism. It is important to note that, in absolute terms, the mean gap is consistently larger for the reactive approach. Furthermore, the average gaps widen as the level of dynamism increases. This highlights the effectiveness of using a reactive approach in a dynamic environment where delays are subject to unexpected disruptions. Specifically, the reactive approach enables the AGVs to adapt to unforeseen disruptions during its traversal, adjusting the task assignments to minimize the total time. This is possible because the reactive approach takes into account the actual delays encountered as the AGVs completes the assigned tasks, rather than just relying on estimated delays that might occur during task completion.

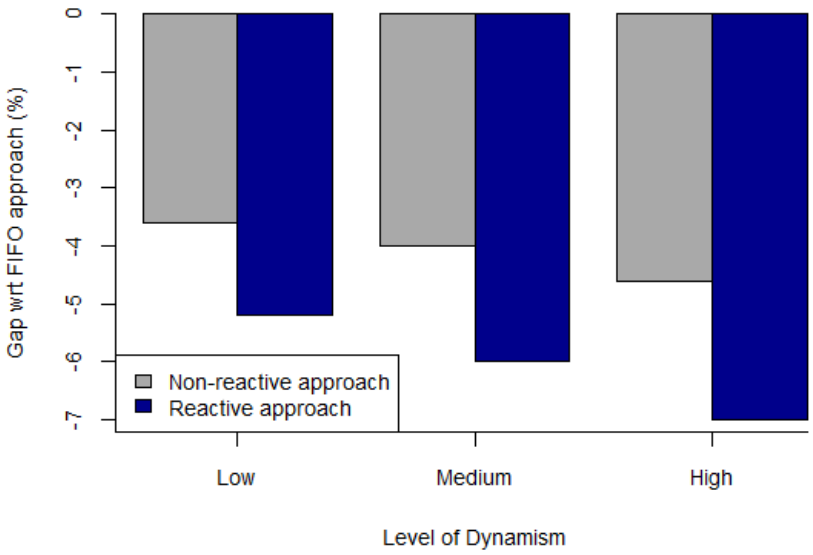


Figure 6.3: Barplot of mean gaps (%) between non-reactive and reactive approaches compared to the FIFO approach across varying dynamism levels (low to high).

An Analysis of Variance (ANOVA) test is then performed to assess the variations in the means. The gap serves as the dependent variable, while the approach, dynamism level, and instance act as independent variables. The results of this analysis are presented in Table 6.4. The analysis concludes that statistically significant differences exist in the means of the various approaches and dynamism levels, but no significant differences are observed between the means of the different instances.

Lastly, Tukey's multiple comparisons of means at the 95% family-wise confidence level are conducted to identify significantly different means among the approaches. The results are

displayed in Table 6.5. It is important to note that there are significant differences between the non-reactive and reactive approaches, as the *p-value* is below 0.05.

Table 6.4: Results from the ANOVA analysis.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Approach	1	24.24	24.24	50.37	0.00000
Level of Dynamism	2	7.88	3.94	8.19	0.00324
Instance	3	4.56	1.52	3.16	0.05181
Residuals	17	8.18	0.48		
Signif. codes 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1					

Table 6.5: Tukey's multiple mean comparisons at a 95% confidence level.

Pair of approaches	Difference	Lower	Upper	P adjusted
Reactive - Non-Reactive	-2.01	-2.6075	-1.4125	0.00000

To sum up,this chapter focused on minimizing the total time required by an AGV to complete a series of tasks in a dynamic environment, where disruptions such as unexpected or random delays can occur along predefined pathways. We introduced a reactive heuristic that integrates a dynamic matrix of expected delays, derived from historical data to estimate delays for each circuit in real time. Additionally, different task permutation strategies for the reactive heuristic were explored and compared with three alternative approaches. Data analytics techniques were applied to estimate AGV interruptions along specified pathways during different time intervals, and White Gaussian noise was used to simulate scenarios with varying levels of circuit delays (low, medium, and high). Computational evaluations were performed on four instance groups (small, medium, large, and very large) and the results were analyzed using ANOVA and Tukey tests. The findings showed that the FIFO approach consistently underperforms compared to the other methods, while the reactive approach yielded better results, especially as the level of dynamism increased. This highlights the effectiveness of the proposed reactive method in handling dynamic environments, such as those involving AGVs.

Chapter 7

Dynamic Team Orienteering Problem with Mandatory Visits

This chapter[1] presents an integration of IoT-based data streams with agile optimization algorithms to address Transport and Mobility (T&M) challenges in smart cities. It focuses on the use of cognitive, descriptive, predictive, and prescriptive analytics algorithms to re-optimize routing decisions in real-time, using continuous data from IoT sensors. The chapter illustrates the benefits of this integration through a numerical example involving a waste collection problem in Barcelona, modeled as a DTOP-MV. It introduces a dynamic solving methodology that adapts to real-world data, enhancing the classical TOP with dynamic traveling times and a trigger mechanism for real-time optimization.

[1]The contents of this chapter are based on the following work:

- Li, Yuda, **Mohammad Peyman**, Javier Panadero, Angel A Juan, and Fatos Xhafa (2022). *"IoT analytics and agile optimization for solving dynamic team orienteering problems with mandatory visits"*. In: Mathematics 10.6, p. 982.

7.1 Problem Description

T&M plays a key role in global economies, driving social and economic development but also contributing to energy consumption and environmental issues. The rise of demand-based economies and e-commerce has increased T&M operations, particularly in urban areas, which impacts the development of smart cities. These cities use IoT devices to collect real-time data, which is analyzed through cloud systems to improve urban services (Beneicke et al., 2019). To promote sustainability, cities are adopting new mobility paradigms such as carsharing and ridesharing, alongside zero-emission vehicles (Juan, Mendez, et al., 2016). These innovations require efficient real-time routing plans for autonomous vehicles, driven by continuous IoT data streams. This data is processed through IoT platforms, providing real-time insights used by optimization algorithms. Data is pre-processed at the edge or fog level and stored in the cloud for further analysis, enabling cognitive, descriptive, predictive, and prescriptive models to enhance decision-making. Governments are promoting open data initiatives to foster the rapid development of new smart city services. Figure 7.1 illustrates a smart city scenario where recycling bins equipped with embedded sensors provide real-time data streams on their saturation levels. This information, combined with data from traffic cameras, is transmitted to the On-board Units (OBUs) systems installed on garbage trucks. These systems use agile algorithms to dynamically re-optimize the routing plans for waste collection.

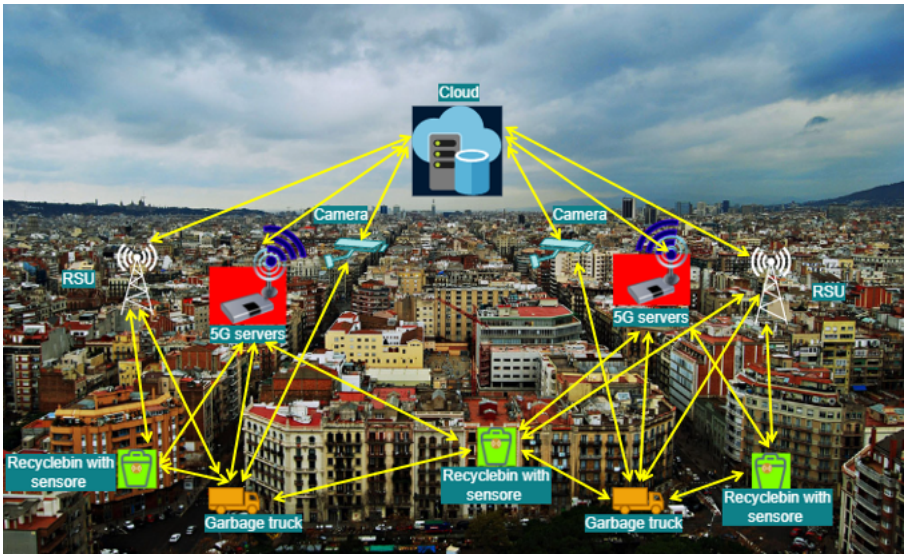


Figure 7.1: IoT-enabled waste collection management in smart cities.

This chapter focuses on presenting analytical algorithms encompassing cognitive, descriptive, predictive, and prescriptive tasks within the T&M domain, alongside their integration with continuous data streams collected via IoT. Additionally, it explores the advantages of combining

agile optimization algorithms with IoT analytics. This integration facilitates re-optimizing the system at frequent intervals by utilizing continuously updated and context-aware information about system dynamics. To illustrate the benefits of IoT, a numerical example is provided, focusing on the waste collection problem in Barcelona, which has been modeled as a TOP, which has been explained in Section 3. While the research on the TOP is substantial, certain gaps persist. A common assumption in most studies is that inputs like travel times between node pairs are static and fully known beforehand. This chapter, however, addresses a more practical scenario where travel times vary dynamically. To tackle this issue, we employ a method that integrates a planning horizon approach with a trigger mechanism. In this framework, the solution is recalculated at the start of each planning period whenever the trigger condition is satisfied. This approach is further refined with an agile BR heuristic, utilizing open real-world data from the city of Barcelona. As new information becomes available through the data stream, the algorithm incorporates it to update the solution within milliseconds. Within this framework, the key contributions of this chapter are as follows: (i) presenting a more realistic variant of the TOP that incorporates dynamic travel times and mandatory nodes; (ii) proposing a dynamic solution methodology that combines a planning horizon with a trigger mechanism; and (iii) validating the algorithm in a realistic dynamic setting using real-world data from open-access repositories.

7.2 Problem Modeling

The static waste collection TOP involves creating a series of open routes to collect waste (Figure 7.2). This waste is distributed across multiple collection points throughout the city, which are linked by edges representing streets. Each collection point is associated with a specific reward, reflecting the waste level in the container, and is defined by coordinates such as latitude and longitude. Each route is assigned a single vehicle, which visits each collection point exactly once. The fleet is assumed to be homogeneous, with a fixed number of vehicles. Furthermore, each vehicle is subject to a maximum time constraint for completing its route. Given this time limitation and the finite number of vehicles, it is not feasible to visit all containers. Consequently, the challenge lies not only in designing the routes but also in selecting which containers to prioritize for collection. Our selection criteria prioritize containers based on their saturation level and their proximity to the vehicle's origin and destination points. The objective of this TOP is to maximize the total amount of waste collected by each vehicle while adhering to the time constraints. In practical waste collection operations, containers that are full or have reached a critical saturation level must be emptied. Therefore, this TOP is extended to the Team Orienteering Problem with Mandatory Visit (TOP-MV), incorporating mandatory visits to such containers.

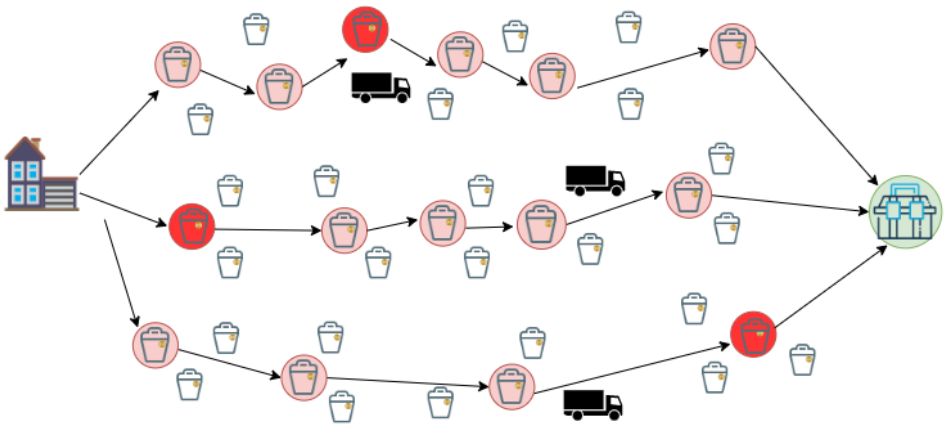


Figure 7.2: TOP-MV waste collection.

Formally, our TOP-MV is defined on a directed graph $G(N, E)$, where N represents the set of nodes, and E denotes the set of edges connecting these nodes, i.e., $E \subseteq N \times N = \{(i, j) \mid i \in N, j \in N, i \neq j\}$. The starting and ending nodes are fixed as nodes 1 and $|N|$, respectively. The set N consists of a set I of collection points, a singleton set O representing the origin facility, and a singleton set F representing the destination facility, such that $N = I \cup O \cup F$. Within I, there exists a subset M, which includes all mandatory collection points. Each collection point $i \in I$ is

characterized by a non-negative reward d_i and a service time S_i. The reward corresponds to the amount of waste to be collected, while the service time represents the duration required to empty the container. Each edge $(i, j) \in E$ has an associated traversal time t_{ij}. The routes are executed by a set K of vehicles, and the total accumulated travel time v_i for a vehicle after visiting collection point i must not exceed the maximum time budget T_{max}. Each collection point can be visited at most once, and only by one vehicle. Additionally, all mandatory collection points must be visited without exception. Each vehicle is restricted to a single route, which must begin at the origin facility $o \in O$ and conclude at the destination facility $f \in F$. The objective of the TOP is to identify a set of routes, constrained by a predefined time budget T_{max}, that visit a subset of N, including all mandatory nodes, while maximizing the total collected reward.

The Team Orienteering Problem Time Window with Mandatory Visit (TOPTW-MV) can be formulated as a mixed-integer linear programming model using a binary decision variable x_{ij}. This variable equals 1 if edge $(i, j) \in A$ is traversed by a vehicle to collect the reward at node j, and 0 otherwise. The objective function aims to maximize the total reward collected:

$$\max \sum_{(i,j)\in A} d_i x_{ij}. \tag{7.1}$$

Subject to constraints:

$$\sum_{j\in N\setminus O} x_{ij} \leq 1, \quad \forall i \in I, \tag{7.2}$$

$$\sum_{i\in N\setminus F} x_{ij} \leq 1, \quad \forall j \in I, \tag{7.3}$$

$$\sum_{k\in N\setminus F} x_{ki} = \sum_{j\in N\setminus O} x_{ij}, \quad \forall i \in I, \tag{7.4}$$

$$\sum_{k\in N\setminus F} x_{ki} = \sum_{j\in N\setminus O} x_{ij} = 1, \quad \forall i \in M, \tag{7.5}$$

$$\sum_{j\in I} x_{oj} = \sum_{i\in I} x_{if}, \tag{7.6}$$

$$\sum_{j\in I} x_{oj} \leq k, \tag{7.7}$$

$$v_j \leq Tmax, \quad \forall j \in N \setminus O, \tag{7.8}$$

$$v_i + t_{ij}\ x_{ij} - v_j \leq Tmax - Tmax\ x_{ij}, \quad \forall i \in N \setminus F, \forall j \in N \setminus \{O, i\}, \tag{7.9}$$

$$\sum_{i\in N} x_{ii} + x_{of} = 0, \tag{7.10}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in A, \tag{7.11}$$

$$v_i \geq 0, \quad \forall i \in N \setminus O. \tag{7.12}$$

Constraint (7.2) ensures that each customer node has at most one outgoing edge. Similarly, constraint (7.3) ensures that each customer node has at most one incoming edge. Constraint (7.4) guarantees flow conservation at each customer node, ensuring the number of incoming edges equals the number of outgoing edges. Mandatory visits are enforced by constraint (7.5). Constraint (7.6) ensures that the number of vehicles departing from the origin facility equals the number arriving at the destination facility. Constraint (7.7) limits the number of routes to the total number of available vehicles k. Constraints (7.8) and (7.9) ensure solution connectivity and adherence to the maximum travel time limit. Constraint (7.10) prevents degenerate routes. Lastly, Constraints (7.11) and (7.12) define the range and nature of the decision variables.

The dataset utilized to validate our approach was sourced from Open Data Barcelona, particularly, Petrol stations in the city of Barcelona is used to model our waste collection problem. The locations of petrol stations across the metropolitan area of Barcelona are used as proxies for the locations of waste containers. It is assumed that the waste level of each container is reported daily by sensors installed in these containers. The dataset includes 61 records, of which 59 represent the waste containers to be serviced, while the remaining 2 correspond to the origin and destination nodes. In Figure 7.3, the origin depot is marked with a red point, the destination treatment facility with a black point, and the three mandatory visit points with green points. The remaining collection locations are represented by blue points.

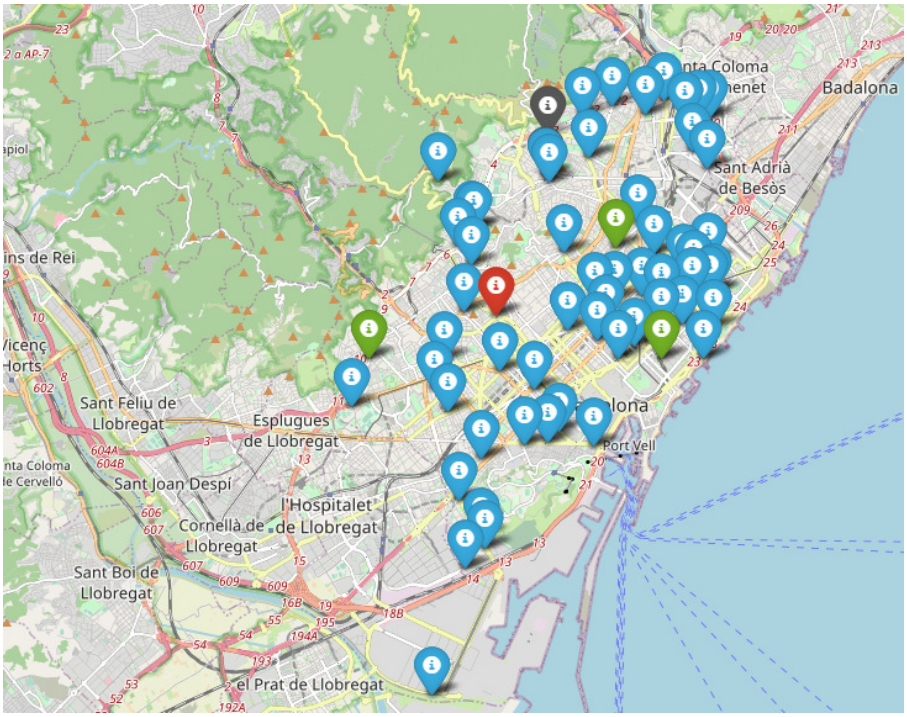


Figure 7.3: Map of waste containers and depots for the experimental instance.

The static version of the TOP-MV assumes that the travel times between nodes are predetermined and remain constant. Once the solution is designed, no further modifications can be made. However, in real-world scenarios, travel times are highly dynamic and can be affected by external factors such as traffic congestion, accidents, and weather conditions. To address this, we developed a case study focusing on a DTOP-MV, in which traffic state information obtained from Traffic state information by sections of the city of Barcelona are considered when constructing the solution routes. This dataset offers updated traffic state information for various sections of Barcelona every five minutes. The traffic data is primarily gathered through sensors embedded beneath the asphalt, which detect changes in the magnetic field caused by passing vehicles. Additionally, sensors utilizing infrared technology and cameras with image processing are employed. The data from each detection station is translated into a service level for the corresponding street section based on predefined thresholds specific to each station. The traffic state is categorized as $s \in \{0, 1, 2, 3, 4, 5, 6\}$, representing seven states: no data, very-fluid, fluid, dense, very-dense, congestion, and cut off. The geometry of the street sections, defined by latitude and longitude, is detailed in this dataset, Street sections relations of the public road of the city of Barcelona. The static travel time between each pair of nodes is computed using the Open Source Routing Machine (OSRM) API, designed specifically for handling large georeferenced datasets, OSRM leverages OpenStreetMap data to determine the optimal route for various travel modes, including car, bicycle, and walking (Luxen et al., 2011).

7.3 Methodological Approach

This section outlines the approach implemented to solve the DTOP-MV. Generally, two strategies exist for addressing dynamic optimization problems, regardless of the specific optimization technique employed. The first strategy involves running the optimization algorithm continuously, allowing it to react in real-time to changes in the environment. The second strategy divides the planning horizon of length T into discrete time periods. The algorithm is then executed once per time period. During each execution, any new information collected within the current time period is forwarded to the next iteration of the algorithm, scheduled for the subsequent time period. The problem instance for the current time slot remains static throughout the duration of that time period (Okulewicz et al., 2017).

In practice, the second approach is widely adopted due to its practicality and efficiency. In this study, we enhance this approach by integrating a trigger mechanism for the re-planning process. The planning horizon of length T is divided into m equidistant time periods, each with a duration p, where $p = T/m$. At the start of each time period, at time $\tau = np$ where $0 \leq n < m$, the algorithm evaluates whether the current traffic conditions differ from those in the preceding

period. If changes in the traffic conditions are detected, the re-planning mechanism is activated to re-optimize the route for the remaining unvisited collection points. Algorithm 3 provides an outline of the savings-based heuristic used to solve the DTOP-MV during a specific time period n. This heuristic is incorporated within a MS metaheuristic framework to balance solution quality and computational efficiency. The input parameters include: the starting node of the route, a list of containers defined by their location coordinates and waste levels, the end depot with its corresponding location coordinates, the number of vehicles, and the α and β parameters used for calculating the savings list and the BR heuristic, respectively. The algorithm proceeds as follows: in stage 1, an initial dummy solution is created, consisting of a set of $|I|$ routes. Each collection point $i \in I$ is initially connected only to the origin depot and the end depot. In stage 2, an enhanced savings list of edges is created. The savings for each edge (i, j), connecting two nodes $i \in I$ and $j \in I$, are calculated as $s_{ij} = \alpha(t_{oj} + t_{if} - t_{ij}) + (1 - \alpha)(d_i + d_j)$. The parameter α is selected within the range $(0.1, 0.9)$, where t_{ij} is the time required to traverse edge $(i, j) \in E$, t_{oj} is the travel time from the origin depot o to node j, and t_{if} is the travel time from node i to the end depot f. Additionally, d_i and d_j represent the reward associated with each node. These computed savings incorporate both the travel time and the cumulative reward obtained by visiting each pair of nodes $(i, j) \mid i \neq j$ within the network. The resulting savings list is then sorted in descending order based on their computed savings values. The main loop continues as long as the savings list is not empty. During each iteration, the edge at the top of the savings list is selected for the merging procedure. To ensure that mandatory nodes are prioritized, edges involving these nodes are always ranked highest on the list, guaranteeing their selection. Routes associated with the selected edge are merged only if the resulting route complies with the time constraint. Regardless of whether the merging is successful, the selected edge is removed from the savings list after processing.

This savings heuristic operates deterministically, as the merging process consistently selects the edge with the highest savings from the savings list. To enhance this approach, we extend it into a BR heuristic by incorporating a controlled level of randomness into the original greedy constructive algorithm, as outlined in Algorithm 3. To incorporate BR behavior, a geometric distribution, Geom(β), with $\beta \in (0, 1)$, is utilized in this study. By setting $0 < \beta < 1$, the BR heuristic can explore various promising alternative solutions until the stopping criteria of the MS framework (either a maximum time limit or a maximum number of iterations) is met. The solution yielding the highest reward is then selected and returned as the final static solution. This final solution serves as the initial input for the dynamic solution. Algorithm 4 presents the planning horizon-based approach for addressing the DTOP-MV. At the start of the planning horizon (period $n = 0$), the algorithm generates an initial static solution based on the current traffic state, with routes remaining fixed for the duration of this period. In subsequent periods,

the algorithm evaluates whether the re-planning trigger condition is satisfied, i.e., whether the traffic conditions in the current period differ from those of the previous period. If the trigger condition is met, the algorithm maintains the nodes visited during previous periods and recalculates the routes for the remaining unvisited nodes, leveraging the current traffic data and using the first stop of the current period as the new origin. If the trigger condition is not satisfied, the existing solution is carried forward into the subsequent period. This procedure is reiterated at the start of each time period until the planning horizon concludes. The core concept of this approach is to enable vehicles to periodically re-optimize their routes based on updated traffic data streamed from the open data server. By doing so, the methodology aims to maximize the total collected reward while adhering to the maximum allowable route time constraints.

Algorithm 3 Multi-Start approach algorithm for solving a static TOP-MV

input:
  origin: starting node
  containers: list of containers and the end depot
  k: number of vehicles
  α: parameter for computing the savings list
  β: parameter for the geometric distribution
end input
function `Biased-Randomized Algorithm`(origin, containers,k,β,α)
$sol_k \leftarrow$ dummySolution(origin, containers)
savings $\leftarrow$ genSavingsList(origin, containers, α)
while savings $\neq \emptyset$ **do**
    $edge_{ij} \leftarrow$ pick(savings,β)
    r_i, $r_j \leftarrow$ getRoutes($edge_{ij}$)
    if checkMerging(r_i, r_j, k) **then**
        $route_i \leftarrow$ merge($route_i$,$route_j$)
        $sol_k \leftarrow$ replace(sol_k,$route_i$)
    end if
    savings $\leftarrow$ remove(savings,$edge_{ij}$)
end while
sol $\leftarrow$ add(sol,sol_k)
return sol
output:
  sol: a solution
end output
function `Multi-Start`(origin, containers,k,β,α)
bestSol $\leftarrow$ Heuristic(origin,containers,k)
while end not reached **do**
    sol $\leftarrow$ BRA(origin, containers,k,β,α)
    if fee(sol) $>$ fee(bestSol) **then**
        bestSol $\leftarrow$ sol
    end if
end while
return bestSol
output:
  bestSol: the best solution
end output

Algorithm 4 Planning Horizon approach for solving DTOP-MV

input:
  origin: starting node
  containers: list of containers and the end depot
  k: number of vehicles
  α: parameter for computing the savings list
  β: parameter for the geometric distribution
end input
function DTOP-MV(origin, containers,k,β,α)
dynamicSol ← Multi-Start(origin, containers,k,β,α)
for $n \in \{1, ..., m\}$ **do**
  if TrafficState$(n-1) \neq$ TrafficState(n) **then**
    for $v \in \{1, ..., k\}$ **do**
      origin, containers ← update(origin,containers, n)
      sol ← BRA(origin,containers, 1,β,α)
      dynamicSol ← update(dynamicSol,sol)
    end for
  end if
end for
return dynamicSol
output:
  dynamicSol: the dynamic solution
end output

7.4 Computational Experiments

This section presents the outcomes of numerical experiments designed to demonstrate both the problem and the proposed solution methodology, comparing various approaches. A total of 36 instances were tested, varying in fleet size and starting times. The reward assigned to each waste container was generated randomly and remained consistent across all instances. Traffic conditions for each time period n are characterized by a coefficient $w^n \in \{1.5, 1, 1.5, 2.5, 3.5, 5, 10\}$, which correspond to the six traffic states described in Section 7.2. The w^n is gathered from the current traffic situation of tram 506 of the Traffic state information from sections of the city of Barcelona in period n. It is assumed to represent the general traffic state of the city, given that most areas exhibit similar traffic patterns within the same time frame. The coefficient w^n directly influences the time t_{ij} required to traverse the edge (i, j). For example, if t_{ij} represents the static time needed to travel from container i to container j, the actual time for traversing the edge (i, j) during period n, adjusted for traffic conditions, is calculated using Equation (7.13). Figure 7.4 illustrates the traffic conditions in the city between 9 a.m. and 9 p.m., segmented into 5-minute intervals across three distinct days. As observed, the predominant traffic state in the city is 2, indicating a fluid traffic situation. During peak hours, such as from 9 a.m. to 11 a.m. (time periods 0 to 25) and from 5:30 p.m. to 7 p.m. (time periods 100 to 120), the city experiences higher traffic density and greater variability compared to other time intervals.

Additionally, working days (November 2 and 6) exhibit higher traffic density and variability

compared to weekend days (November 11). To evaluate our approach under varying levels of dynamism, a series of computational experiments are conducted, incorporating rush hours, non-rush hours, working days, and weekend days. The algorithm's planning horizon T is set to 2 hours, with each period p lasting 30 minutes. The maximum allowable route time $Tmax$ is defined as 3 hours, and the service time S_i for each container is uniformly set to 5 minutes. The maximum computational time allocated for the BR process is restricted to 1 second. Furthermore, if the route time exceeds the predefined limit $Tmax$, a penalty of 20 reward points per additional minute is deducted from the total reward of the route.

$$t_{ij}^{n} = t_{ij} * w^{n} \tag{7.13}$$

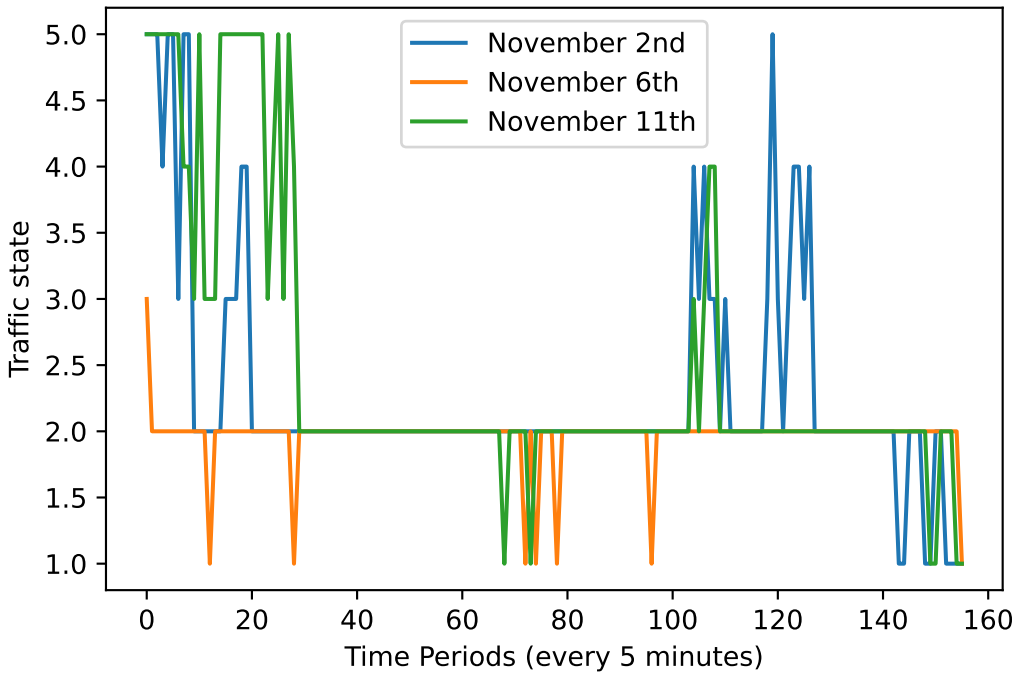


Figure 7.4: Traffic variation in Barcelona between 9:00 AM and 9:00 PM.

The algorithm presented in this section was implemented using Python 3.8. The computational experiments were performed on a system equipped with an i7-8750 CPU operating at 2.20 GHz and 16 GB of RAM.

7.5 Analysis of Results

The results of the experiments comparing the best static solution (OBS) and the best dynamic solution (OBD) are summarized in Table 7.1. The structure of the table is as follows: the first column identifies the name of the instance being evaluated. This instance name specifies the starting day, the starting hour of the route, and the number of vehicles available. For instance, Nov-11d-09h-1v refers to an instance that starts on November 11th at 9:00 a.m. with 1 vehicle. The analysis includes a combination of 3 different days with 4 different starting hours, comprising 2 rush hours (9:00 a.m. and 6:00 p.m.) and 2 non-rush hours (12:00 p.m. and 3:00 p.m.), with varying fleet sizes. The following six columns in the table detail the number of nodes visited, the cost (actual travel time of the routes considering the updated traffic conditions), and the reward achieved by both approaches. Each time value in the table is presented in the format *hours:minutes:seconds*, encompassing both the travel time along the route and the service time at each collection point. Finally, the gap shown in the last column indicates the percentage difference between the reward obtained by the OBD and that achieved by the OBS. A positive gap implies that the OBD yields a higher reward compared to the OBS. Instances with a 0.0% gap signify that no traffic variations occurred during the planning horizon, and as a result, no re-planning procedure was triggered, leaving the dynamic solution identical to the static solution. As anticipated, the majority of these instances occur during non-rush hours, where traffic variability is minimal. The average values for each set of instances are presented in the last row of the table. Notably, no negative gaps are observed between OBS and OBD, and the average gaps remain consistently positive across all instance sizes. Specifically, the average gap is 17.47%, 14.11%, and 10.88% for each respective set of instances. The reduction in the average gap relative to fleet size is primarily attributed to the increased reward achieved by both solutions. However, the positive difference in reward between OBD and OBS tends to grow as the fleet size expands.

These positive gaps between OBS and OBD highlight the critical importance of accounting for the system's inherent dynamism. On average, the reward achieved by OBD consistently surpasses that of OBS, demonstrating the significant advantage of using a solution that adapts to external changing conditions over one that remains static and unresponsive. It is noteworthy that OBS solutions exhibit a higher number of instances violating the time constraint compared to OBD solutions. However, the average cost of OBD across all three instance groups is closer to the $Tmax$ than that of OBS. This outcome is expected, as OBD enables vehicles to re-plan their routes based on real-time traffic updates. Consequently, vehicles in OBD can visit more containers and maximize the use of the remaining time during periods of reduced traffic density or, alternatively, complete routes earlier to avoid exceeding the time limit during periods of

Table 7.1: Case study results for varying start times and fleet sizes.

	Our Best Static Solution OBS			Our Best Dynamic Solution OBD			
Instances	**Visited nodes**	**Cost**	**Reward**	**Visited nodes**	**Cost**	**Reward**	**Gap**
Nov-02d-09h-1v	3	1:54:18	235	7	2:53:02	418	77.87%
Nov-06d-09h-1v	6	1:54:46	426	9	2:59:43	551	29.34%
Nov-11d-09h-1v	3	2:26:27	235	6	2:51:51	375	59.57%
Nov-02d-12h-1v	11	2:39:20	630	11	2:39:20	630	0.0%
Nov-06d-12h-1v	11	2:39:20	630	11	2:39:20	630	0.0%
Nov-11d-12h-1v	11	2:39:20	630	11	2:39:20	630	0.0%
Nov-02d-15h-1v	11	2:39:20	630	11	2:39:20	630	0.0%
Nov-06d-15h-1v	17	3:27:48	357	13	2:58:32	712	99.43%
Nov-11d-15h-1v	11	2:39:20	630	11	2:39:20	630	0.0%
Nov-02d-18h-1v	6	2:18:46	426	8	2:55:19	508	19.24%
Nov-06d-18h-1v	11	2:39:20	630	11	2:39:20	630	0.0%
Nov-11d-18h-1v	4	1:47:18	337	10	2:28:02	465	37.98%
Average	8.75	2:28:47	483	9.91	2:45:12	567	17.47%
Nov-02d-09h-2v	7	3:46:07	464	11	5:52:52	600	29.31%
Nov-06d-09h-2v	19	4:20:01	972	24	5:54:49	1245	28.08%
Nov-11d-09h-2v	7	5:03:08	464	13	5:29:35	713	53.66%
Nov-02d-12h-2v	24	5:31:45	1372	24	5:31:45	1372	0.0%
Nov-06d-12h-2v	24	5:31:45	1372	24	5:31:45	1372	0.0%
Nov-11d-12h-2v	24	5:31:45	1372	24	5:31:45	1372	0.0%
Nov-02d-15h-2v	24	5:31:45	1372	24	5:31:45	1372	0.0%
Nov-06d-15h-2v	33	6:36:53	1022	28	5:42:28	1508	47.55%
Nov-11d-15h-2v	24	5:31:45	1372	24	5:31:45	1372	0.0%
Nov-02d-18h-2v	19	4:56:58	972	20	5:57:12	1117	14.91%
Nov-06d-18h-2v	24	5:31:45	1372	24	5:31:45	1372	0.0%
Nov-11d-18h-2v	13	3:41:53	729	24	5:53:40	1254	72.01%
Average	20.16	5:07:57	1071	22	5:40:05	1222	14.11%
Nov-02d-09h-3v	13	5:55:57	711	24	8:49:36	1214	70.74%
Nov-06d-09h-3v	27	6:11:38	1401	33	8:32:44	1746	24.62%
Nov-11d-09h-3v	13	8:02:08	711	15	8:58:51	796	11.95%
Nov-02d-12h-3v	41	8:31:34	2016	41	8:31:34	2016	0.0%
Nov-06d-12h-3v	41	8:31:34	2016	41	8:31:34	2016	0.0%
Nov-11d-12h-3v	41	8:27:06	2016	41	8:27:06	2016	0.0%
Nov-02d-15h-3v	41	8:49:25	1676	41	8:49:25	1676	0.0%
Nov-06d-15h-3v	51	9:37:23	1693	42	8:47:41	2029	19.84%
Nov-11d-15h-3v	41	8:40:29	1856	41	8:40:29	1856	0.0%
Nov-02d-18h-3v	27	7:10:45	1401	30	8:56:07	1565	11.70%
Nov-06d-18h-3v	41	8:31:34	2016	41	8:31:34	2016	0.0%
Nov-11d-18h-3v	21	5:42:50	1081	35	8:35:10	1672	54.67%
Average	33.16	7:51:02	1549	35.41	8:40:59	1718	10.88%

increased traffic density.

Figure 7.5 illustrates the distribution of travel costs across all instances. The plots are grouped by the number of vehicles, ranging from 1 to 3, and distinguish between the OBS (blue boxplots) and the OBD (purple boxplots). The mean value of each sample is depicted by a white circle, while the black diamond markers identify outliers within the dataset. This figure clearly demonstrates that the dynamic approach significantly reduces travel time variability, as evidenced by the narrower range between the extreme points in each boxplot. Compared to OBS solutions, OBD solutions exhibit travel times that are more closely aligned with the time budget $Tmax$, maximizing the remaining time to visit additional containers. Simultaneously, OBD solutions show a much lower likelihood of exceeding the time limit, thereby reducing the probability of incurring penalization costs compared to OBS solutions.

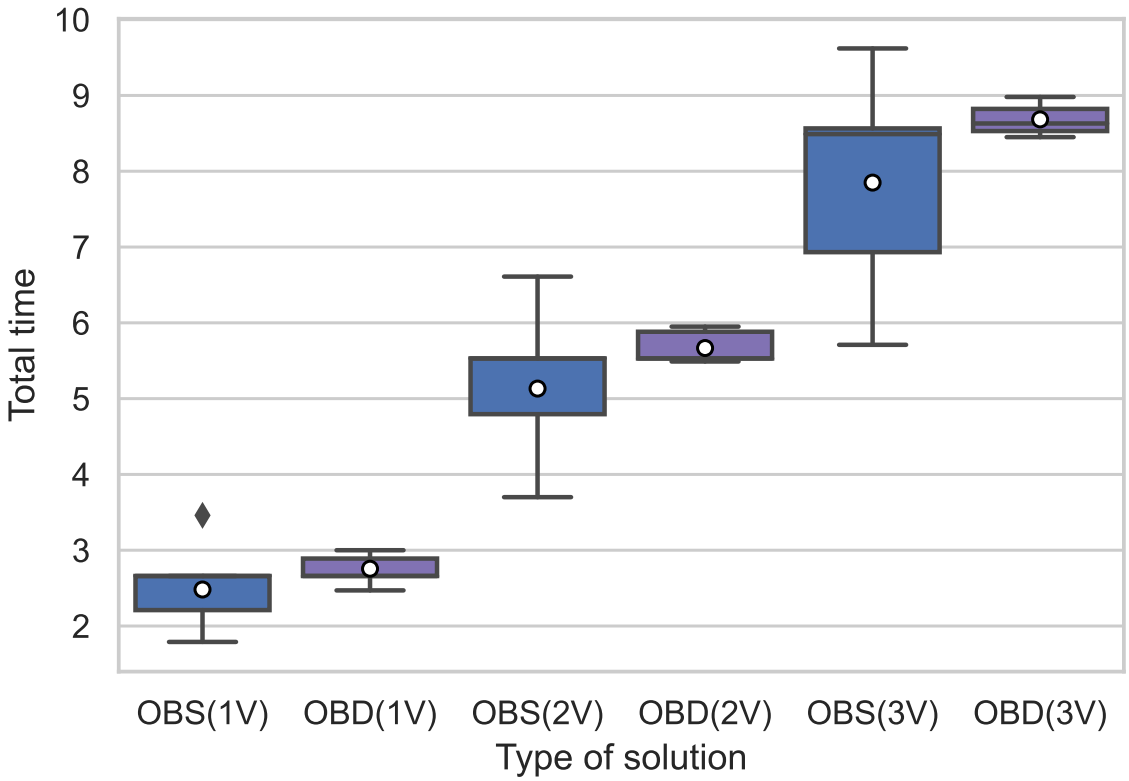


Figure 7.5: Costs for each solution type at different fleet sizes.

Figure 7.6 illustrates the distribution of rewards across all instances. The white circle in each boxplot indicates the mean reward, revealing that the dynamic solution consistently achieves higher average rewards compared to the static solution, irrespective of the fleet size. Additionally, both the upper and lower bounds of the rewards obtained by the dynamic solution surpass those of the static solution. This demonstrates that, in both the best- and worst-case scenarios, the dynamic solution consistently yields higher rewards. These findings highlight that, with respect

to rewards, the dynamic approach consistently outperforms the static approach, which lacks adaptability to changing traffic conditions. The results further validated the adaptability and scalability of our algorithm across various fleet sizes, demonstrating its suitability for urban environments with fluctuating travel times. Beyond waste collection, this methodology can be extended to other domains, such as planning tourist itineraries, optimizing resource allocation (e.g., deploying fire trucks in wildfire emergencies), and vehicle routing with optional stops. Although the current model excludes loading capacity constraints, these could be incorporated to address scenarios involving the transport of larger items.

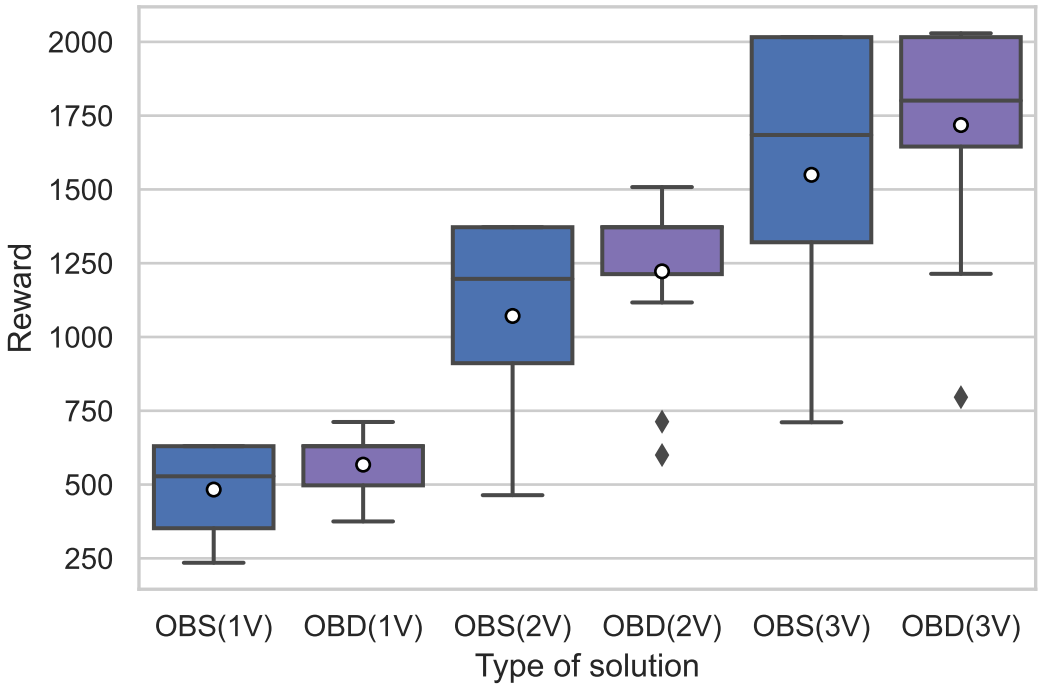


Figure 7.6: Rewards for each solution type across different fleet sizes.

Part III

Simheuristics

Chapter 8

Simheuristics Framework

8.1 Main Concept

Real-world optimization problems are often influenced by uncertainties that arise from unpredictable factors such as fluctuating demand, variable travel times, and external conditions like traffic or weather. These uncertainties complicate the decision-making process and require models that can not only optimize under ideal conditions but also adapt to varying real-world scenarios. Incorporating uncertainty into optimization frameworks is crucial to ensure that solutions remain robust and effective when applied to dynamic, real-life environments. This chapter explores how uncertainty can be represented and managed in optimization problems and introduces simheuristics as an approach that combines simulation with heuristic optimization techniques. All our published articles contribute to this chapter.

The values of variables in real-world problems are often uncertain. This uncertainty can arise from factors such as delays that increase travel time or unexpected fluctuations in demand and order size. For example, travel time between two points in an urban network is typically estimated based on speed and distance. However, variables like traffic congestion and weather conditions can cause actual travel times to vary over time. In many cases, this uncertainty can be represented by stochastic variables following a probability distribution, such as lognormal or Weibull distributions. These distributions are useful for modeling skewed and non-normal data. As a result, travel times can be seen as a combination of ideal travel times and delays influenced by random variables. In optimization problems, incorporating uncertainty into the model is essential for accurately representing real-world conditions. Travel time, demand, or other uncertain variables can be integrated into the optimization framework, leading to more robust solutions. One approach to address this uncertainty is simheuristics, which integrates simulation into the optimization algorithm to handle stochastic variables effectively. This process begins by simplifying the problem into its deterministic version, where uncertain variables are

replaced by their expected values, such as the mean travel time between nodes (Figure 8.1). This provides a more manageable starting point before introducing probabilistic elements to account for the full range of uncertainty in the solution process. The deterministic version of the problem is solved, and solutions are identified using an iterative optimization algorithm. In the simheuristic approach, it is assumed that solutions to the deterministic version of the problem are likely to be good candidates for the original problem, which includes uncertainty. Since these solutions relate to the deterministic version, they are evaluated using simulation under uncertain conditions. The simulation could be in the form of a MCS or a DES. A small number of simulation runs are conducted to estimate the objective value corresponding to each deterministic solution. The most promising solutions are stored in a list of elite solutions, which is continually updated throughout the optimization process. The generation and evaluation of solutions continue until a predefined stopping criterion is met. At this point, the best solutions in the elite list are further analyzed with additional simulation runs. This deeper analysis allows for comparison between solutions and includes risk analysis. Based on these evaluations, the best solutions are recommended.

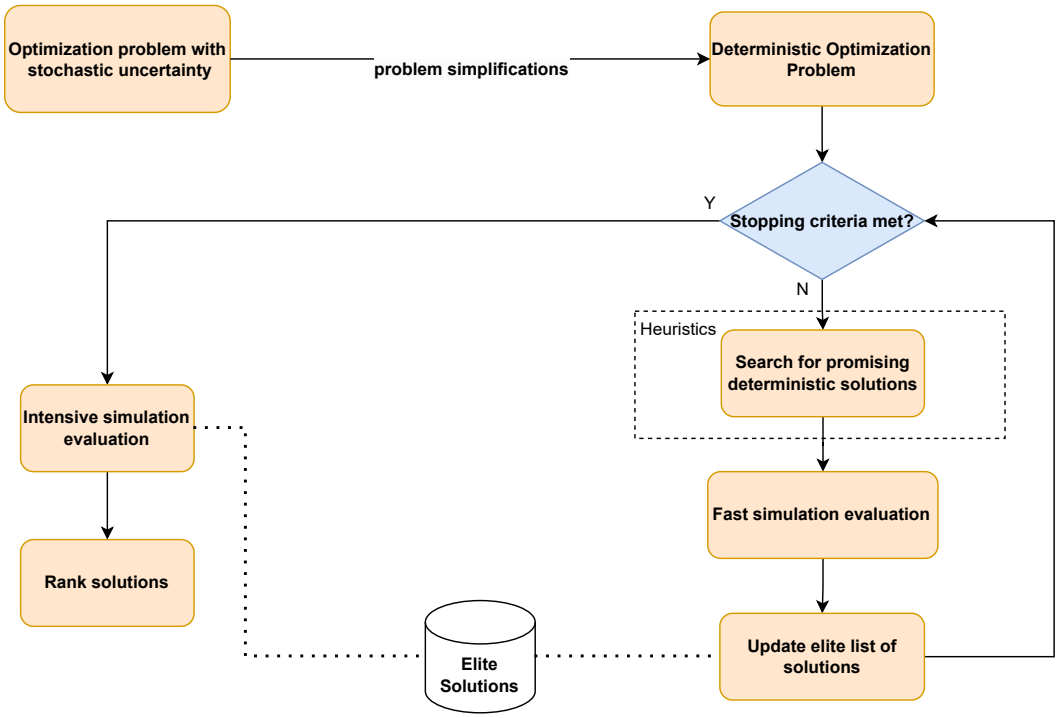


Figure 8.1: Basic scheme of the simheuristic approach.

Source:Juan, Faulin, et al. (2015)

8.2 Related Work

Simulation-optimization techniques have been applied across various domains to enhance decision-making under uncertainty (Table 8.1). For example, Kumar, Rajalakshmi, et al. (2020) proposed a simulation-optimization method using Industry 4.0 components like IoT for improving service quality in critical infrastructure, demonstrated with Dehradun's bus transport system. Oliva et al. (2020) introduced fuzzy simheuristics to handle both stochastic and non-stochastic uncertainties, showcasing their approach with the TOP. Peidro et al. (2024) addressed the uncapacitated facility location problem using a simheuristic with stochastic service costs, combining tabu search with simulation to find near-optimal solutions. Gruler, Pérez-Navarro, et al. (2020) applied a simheuristic approach to the time-dependent waste collection problem, integrating MCS to account for time dependencies and stochastic travel speeds, using data from Sabadell, Spain. Martins, Bayliss, et al. (2020) used a simheuristic to solve the omnichannel vehicle routing problem with stochastic travel times, extending the problem to a two-echelon network with savings-based heuristics and MCS. Tordecilla, Juan, et al. (2021) reviewed hybrid simulation-optimization methods for resilient supply chain design, emphasizing the importance of handling disruptions like COVID-19 through robust strategies. Simheuristics are particularly effective in route optimization problems with stochastic elements like varying waste generation rates and traffic conditions (Gruler, Fikar, et al., 2017; Panadero, Juan, Ghorbani, et al., 2024). Other studies have focused on uncertainty in facility location problems using mean-variance objectives (Taymaz et al., 2020). Kim et al. (2024) analyzed drone facility locations, considering distances and delivery schedules. De Armas et al. (2017) addressed stochastic uncapacitated facility location problem with a savings-based heuristic and simulation. Herrera et al. (2022) combined survival analysis with simheuristics to estimate delays in routing plans, aiding in optimal strategy selection. Additionally, agile two-stage simheuristics have been successful in urban ridesharing with uncertain travel times.

Table 8.1: Summary of Related Work on Simheuristics.

Reference	**Application Area**	**Contribution**
(Kumar, Rajalakshmi, et al., 2020)	Critical Infrastructure	Proposed a simulation-optimization method to improve the service quality of a bus transport system.
(Oliva et al., 2020)	TOP	Introduced fuzzy simheuristics for handling stochastic and non-stochastic uncertainties.
(Peidro et al., 2024)	Facility Location	Developed a simheuristic with stochastic service costs, combining tabu search.
(Gruler, Pérez-Navarro, et al., 2020)	Waste Collection	Applied a simheuristic approach to time-dependent waste collection, using MCS for stochastic travel speeds.
(Martins, Bayliss, et al., 2020)	Omnichannel VRP	Extended simheuristics to a two-echelon network with savings-based heuristics and MCS.
(Tordecilla, Juan, et al., 2021)	Supply Chain	Reviewed hybrid simulation-optimization methods for resilient supply chain design.
(Gruler, Fikar, et al., 2017)	Route Optimization	Demonstrated the effectiveness of simheuristics in managing route optimization with stochastic elements.
(Panadero, Juan, Ghorbani, et al., 2024)	Route Optimization	Used simheuristics to manage variability in urban environments.
(Taymaz et al., 2020)	Facility Location	Addressed uncertainty in facility location problems for healthcare services.
(Kim et al., 2024)	Drone Logistics	Analyzed drone facility locations, focusing on distances and schedules.
(De Armas et al., 2017)	Facility Location	Tackled the stochastic uncapacitated facility location problem using a savings-based heuristic with simulation.
(Herrera et al., 2022)	Routing Optimization	Combined survival analysis with simheuristics for delay estimation in routing plans.

Chapter 9

The Open Vehicle Routing Problem with Stochastic Service and Travel Times

Real-world problems often exhibit high complexity, making it challenging to find optimal solutions within a short computational time frame. To overcome this challenge, this chapter[1] introduces a hybrid approach that combines heuristic methods with MCS. The standard versions of BR, MS, and metaheuristic techniques, discussed in Part II, are not well-suited for managing stochastic variables. In contrast, MCS provides effective tools for handling uncertainty. This integration of heuristics and MCS, known as "simheuristics", was comprehensively discussed in the previous chapter. Here, we demonstrate the application of simheuristics through a case study based on the COVID-19 pandemic crisis. Specifically, we hybridize BR, MS, and MCS methods to effectively solve an OVRP.

[1]The contents of this chapter is based on the following work:

- **Peyman, Mohammad**, Yuda Li, Rafael D Tordecilla, Pedro J Copado-Mendez, Angel A Juan, and Fatos Xhafa (2021). *"Waste collection of medical items under uncertainty using internet of things and city open data repositories: A simheuristic approach"*. In: 2021 Winter Simulation Conference (WSC). IEEE, pp. 1–11.

9.1 Problem Description

The emergence of the COVID-19 pandemic has led to a profound global socio-economic crisis while also significantly affecting the environment. In an effort to mitigate the spread of COVID-19, governments and public health authorities worldwide implemented strict measures, including lockdowns, quarantines, and border restrictions. Although these actions have temporarily improved environmental conditions by reducing air pollution, the effect may not be long-lasting, as pollution levels are expected to rebound as global recovery progresses. Furthermore, the widespread use of Personal Protective Equipment (PPE), such as masks and gloves, during the pandemic has resulted in the generation of billions of units of contaminated waste. Even now, COVID-19 remains a persistent challenge for global public health. Saberian et al. (2021) estimate that approximately 6.88 billion face masks (equivalent to around 206,470 tons) are produced globally each day. In several cities, the daily consumption of face masks (in terms of quantity) can be roughly calculated by multiplying the city's population by the mask usage acceptance rate (Nzediegwu et al., 2020). Many of these masks consist of petroleum-based, non-renewable polymers that are non-biodegradable (Dharmaraj et al., 2021), requiring hundreds or even thousands of years to decompose in the environment. A significant portion of the discarded PPE ends up in landfills or the marine ecosystem, leading to environmental contamination and adversely impacting the flora and fauna. Consequently, the proper disposal and management of PPE, along with other sanitary and medical waste, has become an urgent priority. Ignoring this critical issue could result in severe environmental and public health consequences. A practical solution to address this issue involves ensuring that both PPE and medical waste are directed to specialized processing facilities, thereby preventing the generation of additional contaminating waste. In the context of the COVID-19 pandemic in Barcelona, one approach could be the deployment of dedicated medical waste containers in densely populated areas and at sanitary or medical facilities. Additionally, small sensor devices can be installed in each container to monitor their saturation levels (Figure 9.1). The waste level data from each container are transmitted to the city's open data center. This information is periodically accessed by the relevant authorities, allowing them to assign cargo vehicles to visit and empty the containers as needed. By utilizing small sensors in the containers, unnecessary visits to containers with sufficient remaining capacity can be avoided, thereby minimizing overall transportation costs. Implementing specialized PPE waste containers provides at least two key advantages:

The use of specialized PPE waste containers offers at least two significant benefits:

1. Research has demonstrated that SARS-CoV-2 can persist on hard surfaces for extended periods (Choi et al., 2021). Improper handling of PPE and medical waste may increase the likelihood of COVID-19 transmission in the environment and pose a risk of infection

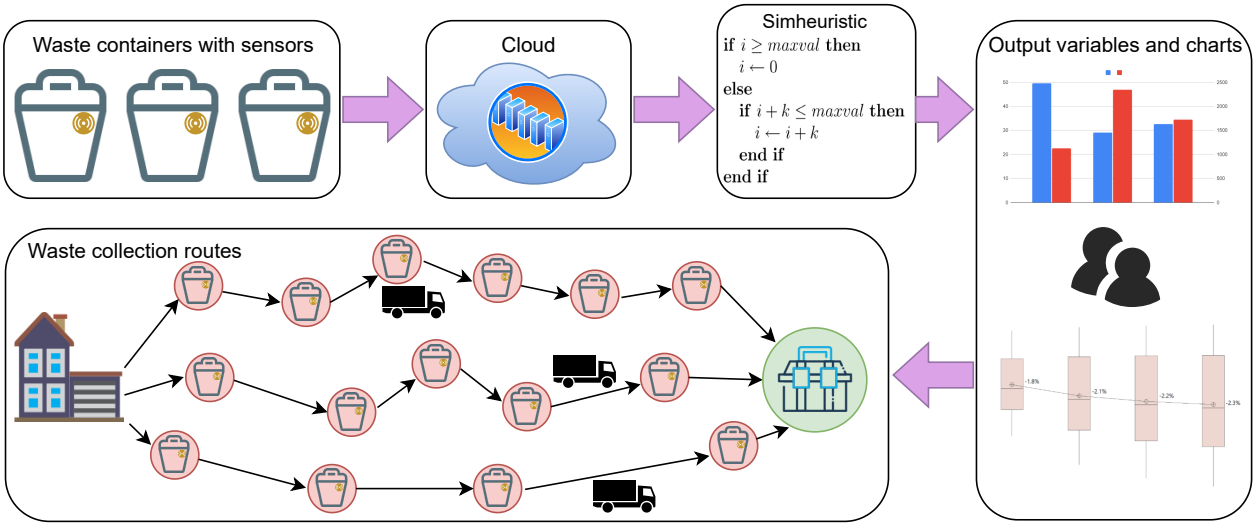


Figure 9.1: Information flow schema for the medical waste collection problem.

to waste management workers. Specialized waste containers help mitigate these risks by ensuring that PPE and medical waste are stored and handled separately from general waste.

2. The gathered PPE and medical waste can be transformed for multiple uses, including serving as recycled concrete aggregate in pavement construction (Saberian et al., 2021). The processes of recycling and reusing PPE and medical waste require intricate sorting methods; however, these processes can be enhanced by employing specialized waste containers that facilitate the separation of materials.

   In this chapter, the waste collection problem is formulated as an OVRP. Unlike the traditional VRP, the OVRP accounts for distinct origin and destination depots (Li, Golden, et al., 2007). To identify the optimal route with the shortest travel time, a BR heuristic algorithm is employed, following the approach outlined by Belloso et al. (2019). Traditionally, this problem has been analyzed under deterministic conditions, assuming that travel and service times are known and constant. However, in real-world scenarios, these activities are inherently stochastic. To address this, a simulation component is incorporated into the BR framework to evaluate the robustness of solutions in the presence of stochasticity, where both travel and service times exhibit variability. As a result, data on waste levels can be accessed from the cloud and utilized to feed a simheuristic algorithm. The simheuristic generates a set of high-quality solutions, which are then evaluated by decision-makers. Key performance indicators, such as cost and reliability, along with descriptive charts, can be employed during the evaluation process. Ultimately, a single

routing plan is selected and implemented to carry out the waste collection tasks.

9.2 Problem Modeling

The medical waste collection OVRP involves designing a series of open routes to collect medical waste. This waste is distributed across multiple collection points throughout the city, all of which need to be visited. These collection points are linked by edges that represent city streets. Each point is characterized by a specific demand and associated coordinates, such as latitude and longitude. Each route is assigned a single vehicle, which visits each collection point exactly once. It is also assumed that the fleet consists of homogeneous vehicles and that the fleet size remains constant. Additionally, the vehicles are considered to have unlimited capacity, as the type of waste being collected (e.g., surgical masks, syringes, hypodermic needles, scalpel blades, etc.) occupies negligible space. Given that loading capacity is not a limiting factor, the primary constraint is the maximum amount of time available for each vehicle to complete its route, introducing a time-dependent capacity constraint. Service times correspond to the duration required to collect medical waste at each collection point, whereas travel times represent the time spent traversing each edge. Both travel and service times are treated as stochastic variables, as they may be influenced by various random factors (e.g., traffic conditions, weather, road disruptions due to accidents, etc.). It is assumed that travel times are independent of one another. Additionally, the origin and destination points of the vehicles are different, meaning that the designed routes are open (Figure 9.1). For example, each vehicle may begin its route at the firm's headquarters, visit its designated collection points, and conclude at a treatment facility. The Government of Catalonia has outlined a protocol for carrying out these activities across its territory, including Barcelona (*Health Care Waste* 2021; *Gestió extracentre dels residus sanitaris* 2019). In practice, the headquarters and treatment facility may either share the same location or be situated in different parts of the metropolitan area. To account for this, we consider the more general scenario where their locations can vary. The objective is to minimize the total time required by the fleet of vehicles to complete the collection tasks. It is important to note that if a route exceeds its maximum allowable time, the vehicle must make a mandatory rest stop before continuing with the collection plan. Lastly, while the problem addressed in this study focuses on medical waste collection, it is worth highlighting that the approach can also be applied to the collection of other waste products with sizes significantly smaller than the vehicle capacity, such as batteries or cell phones.

Formally speaking, the problem can be defined on a directed graph $G(N, E)$, in which N is the set of nodes, and E is the set of edges linking these nodes, such that $E \subseteq N \times N = \{(i, j) \mid i \in N, j \in N, i \neq j\}$. The set N is formed by a set I of collection points, a singleton set O

representing the origin facility, and a singleton set F representing the end facility, such that $N = I \cup O \cup F$. Each collection point $i \in I$ has a service time S_i, as well as a deterministic and known demand d_i. This demand refers to the quantity of medical items to be collected. Each edge $(i, j) \in E$ is traversed in a time T_{ij}. Both S_i and T_{ij} are random variables and follow known probability distributions. These distributions are assumed to be based on historical data. A set V of uncapacitated vehicles is available to perform the routes. Each collection point must be visited once by only one vehicle. Each vehicle is assigned to only one route. Each route must start in the origin facility and finish in the end facility. The total time of each route must not exceed a given time limit t_{max}. Hence, the problem consists in designing a set of $|V|$ routes that meet the aforementioned constraints, such that the expected total time of performing all routes is minimized. Notice that no loading capacity constraints are considered since, as explained before, it is assumed that the medical items to be collected are of small size.

9.3 Methodological Approach

Initially, a BR heuristic is proposed to address the deterministic version of the waste collection OVRP. This BR heuristic is subsequently integrated into a MS metaheuristic framework (Martí et al., 2013). The MS BR heuristic is capable of identifying feasible and high-quality solutions within short computational times. However, it is specifically designed to solve the problem under the assumption that all input data are deterministic. In practical applications, uncertainty plays a vital role in the decision-making process. To address this, the MS BR heuristic is enhanced and extended into a simheuristic algorithm, enabling it to handle more realistic scenarios involving uncertainty. Simheuristics have recently been utilized to address complex stochastic optimization problems, such as stochastic inventory routing problems (Gruler, Panadero, et al., 2018; Gruler, Panadero, et al., 2020) and stochastic facility location problems (Pagès-Bernaus et al., 2019). These algorithms integrate heuristics or metaheuristics with simulation techniques to effectively manage uncertainty (Juan, Kelton, et al., 2018). As a result, they can produce solutions that balance the trade-off between minimizing the expected total time and maximizing solution reliability. In this study, reliability is defined as the probability of successfully executing a routing plan without failures, meaning that no route exceeds the maximum allowable time to complete the waste collection process. In this study, we integrate the proposed MS BR heuristic framework with MCS to handle scenarios where both the service times at collection points and the travel times between node pairs are modeled as random variables. These random variables follow specified probability distributions, which are assumed to be derived from fitting historical data for each collection point and each travel edge within the city's network.

Thus, the reliability rate reflects the robustness of the deterministic solution when applied

to stochastic scenarios, indicating whether the total travel time of a route remains acceptable or feasible under such conditions, given the maximum allowable route travel time. Our simheuristic approach is embedded within the MS-BR heuristic framework, as illustrated in Figure 9.2. The algorithm operates as follows. First, an initial solution S^* is generated using the BR heuristic with β set to 1. Next, in solution S^*, deterministic travel and service times are replaced with their stochastic counterparts. A brief simulation (with a small number of replications) is then conducted to estimate the average stochastic travel time and its corresponding reliability rate. At this stage, the solution S^* is considered the best deterministic and stochastic solution identified so far. Throughout the iterations, the best solution S^*, based on both deterministic and stochastic times, is retained. Following the approach of Rabe et al. (2020), a new solution S' is generated in each iteration using the BR heuristic. However, the stochastic time of S' is only evaluated through a brief simulation if S' is deemed promising. This condition is met when the deterministic time of S' is lower than that of S^*. If the stochastic time of S' also outperforms that of S^*, then S^* is updated to S'. Otherwise, S' is discarded. Once these steps are completed, the best solution is subjected to a single extended simulation (with a large number of replications) to enhance the accuracy of its stochastic time estimate and its reliability level.

9.4 Computational Experiments

The data used to evaluate our approach is derived from a deterministic real-world scenario that emerged during the initial months of the COVID-19 pandemic in Barcelona. It is assumed that these data are reported daily by sensors installed in 96 medical waste containers distributed across the metropolitan area. The origin depot and the destination treatment facility are represented as red cylinders in Figure 9.3, while the remaining collection points are depicted as blue dots.

In this case study, a time limit of four hours (t_{max}) is imposed for each route, reflecting the mobility restrictions in place during the pandemic lockdown. Additionally, six vehicles are available for routing, and all must be utilized. Preliminary tests with varying fleet sizes revealed that using five or fewer vehicles results in infeasible solutions due to the t_{max} constraint. To address the problem, the original deterministic instance is converted into a stochastic one by incorporating both random travel times T_{ij} and random service times S_i, as described below:

- The travel time T_{ij} is modeled using a log-normal distribution with a minimum travel time t_{ij}^{min} along edge (i, j), which represents the optimal travel time under ideal conditions. If the stochastic component of T_{ij} is represented by Θ_{ij}, where $\Theta_{ij} \sim \log N(\lambda, \gamma)$ (a

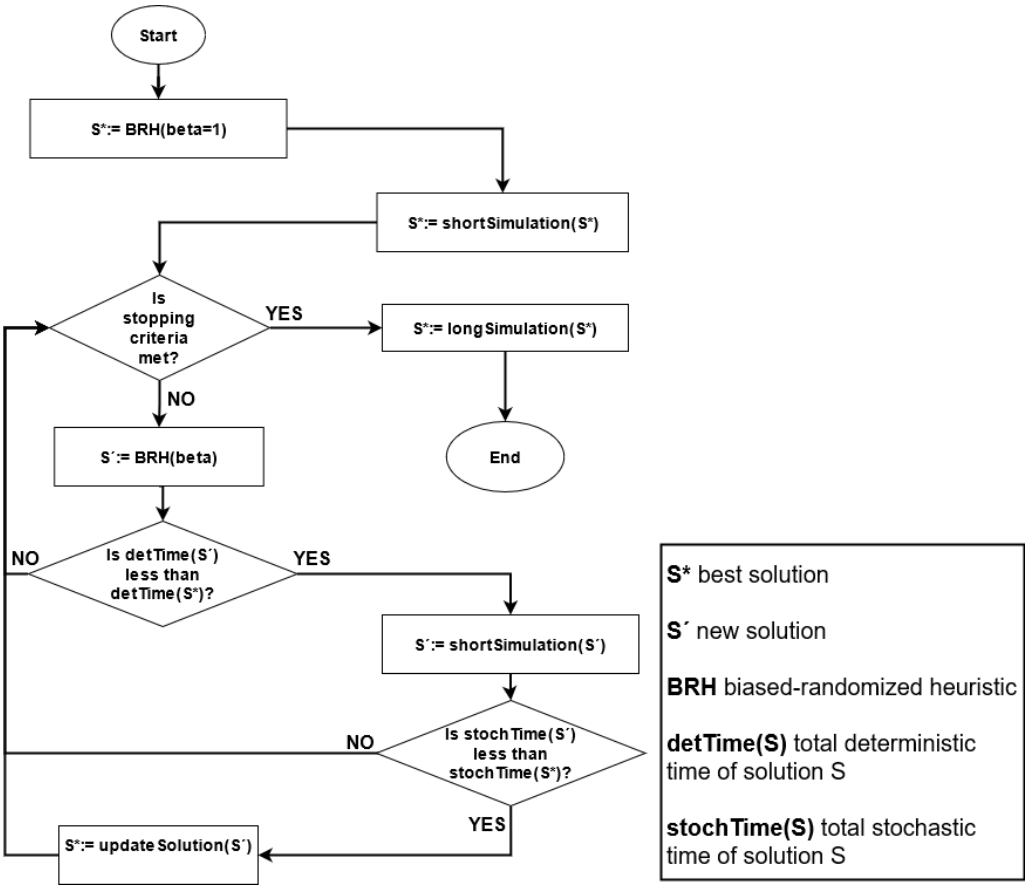


Figure 9.2: Flowchart of the simheuristic algorithm.

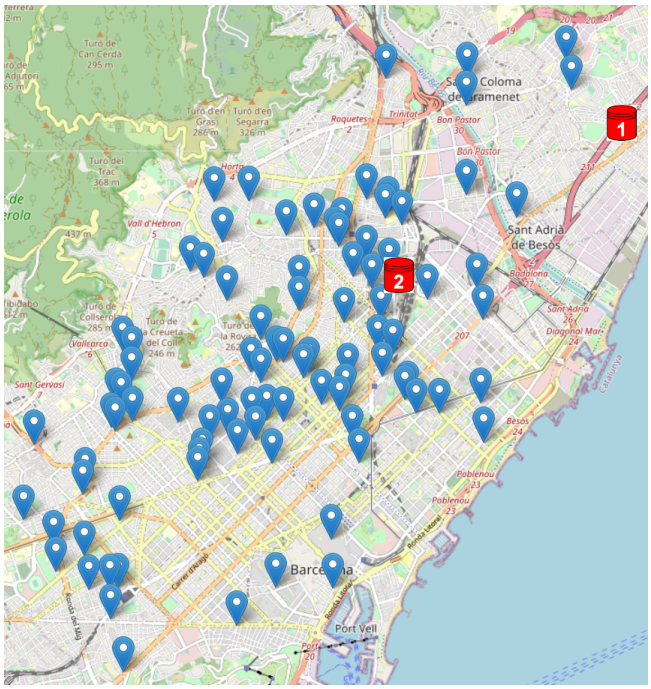


Figure 9.3: Map of collection points and depots for the experimental instance.

log-normal distribution with location parameter λ and scale parameter γ), the travel time is computed as:

$$t_{ij} = t_{ij}^{min} + k\theta_{ij},$$

where k represents the variability level, and t_{ij} and θ_{ij} are realizations of T_{ij} and Θ_{ij}, respectively. For the experiments, we use $\lambda = 0$, $\gamma = 1$, and $k \in \{5, 6, 7, 8, 9, 10\}$ to explore various levels of variability. These parameters ensure a diverse set of scenarios for analysis. Additionally, the values of t_{ij}^{min} for each edge are derived from a web mapping service.

- The service time S_i at each collection point follows a log-normal distribution, $S_i \sim \log N(\lambda, \gamma)$. Using real-world data as inspiration, the parameters λ and γ are adjusted such that the expected service time, $E[S_i]$, is 420 seconds and the variance, $Var[S_i]$, is 5. These values ensure the service time reflects realistic conditions.

The variability level k defines six distinct scenarios: very-low, low, medium-low, medium-high, high, and very-high. For the algorithm parameters, the β parameter used in the BR heuristic is assigned a random value within the interval $(0.05, 0.25)$. In each iteration, the β value is randomly selected from this range, which, based on our preliminary experiments, appears to yield reasonably good solutions. The number of short simulation runs is fixed at 100, while the long simulation consists of $1,000$ runs. For a more detailed analysis of the appropriate number of simulation runs, refer to Rabe et al. (2020). Additionally, if the stochastic travel time of a route exceeds t_{max}, a penalty of $1,500$ seconds is applied. This penalty accounts for the mandatory stop required whenever a driver reaches the maximum allowable driving time.

The algorithm has been implemented in Python 3 and executed on a personal computer equipped with 16 GB of RAM and an Intel Core $i7$-8750H processor running at 2.2 GHz. For each variability level, the maximum allowed computational time is limited to 60 seconds.

9.5 Analysis of Results

For each uncertainty scenario, Table 9.1 presents a comparison between our Best Deterministic Solution (OBD) (representing the best solution identified for the deterministic version of the problem) and our Best Stochastic Solution (OBS) (denoting the best solution obtained under conditions of uncertainty). All time values in Table 9.1 are presented in the format *hours:minutes:seconds*, incorporating both the travel duration along the routes and the service time at the collection points. The Longest-Route Time (LRT) indicates the longest time recorded for an individual route when comparing the total times of all routes. The results indicate that, in the deterministic scenario, the four-hour maximum time limit is never surpassed, and all six available vehicles are utilized. The deterministic total time refers to the sum of the times for all routes in the solution, calculated under conditions of complete certainty. Naturally, this value remains constant across all variability levels.

Table 9.1: Results for a case study considering different levels of uncertainty.

Variability level	**Our best deterministic solution (OBD)**				**Our best stochastic solution (OBS)**		
	Longest-route time	**Deterministic total time**	**Stochastic total time**	**Reliability**	**Longest-route time**	**Stochastic total time**	**Reliability**
Very-low	3:58:37	22:26:05	23:03:32	15.63%	3:55:26	22:44:20	97.50%
Low	3:58:37	22:26:05	23:13:00	4.75%	3:55:26	22:48:41	91.33%
Medium-low	3:58:37	22:26:05	23:22:45	1.40%	3:55:26	22:54:05	81.04%
Medium-high	3:58:37	22:26:05	23:31:43	0.31%	3:55:26	23:01:15	64.54%
High	3:58:37	22:26:05	23:40:00	0.07%	3:55:26	23:08:31	48.70%
Very-high	3:58:37	22:26:05	23:48:40	0.02%	3:55:26	23:16:33	32.53%
Average	3:58:37	22:26:05	23:26:37	3.70%	3:55:26	22:58:54	69.27%

After obtaining the deterministic solution, it is evaluated within a stochastic environment to determine the expected total time required for its implementation under uncertain conditions. Additionally, the reliability value is calculated, representing the estimated probability that the collection plan can be executed without any route surpassing its maximum allowable time. The OBS columns display the same key performance indicators as those in the OBD columns. However, these indicators are derived from executing the simheuristic algorithm. The results demonstrate the clear advantage of the simheuristic approach, as the stochastic total time is consistently lower than that of the OBD across all instances and variability levels. Furthermore, the reliability of the OBS exceeds that of the OBD, indicating that the simheuristic algorithm ensures a higher probability of completing each route within the maximum allowable time. Additionally, as the four-hour time limit is never exceeded, our solution avoids incurring any penalty time. It is also worth noting that the LRT of the OBS is smaller than that of the OBD. Since the OBD does not account for stochastic conditions, the route times are often very close to the four-hour limit. However, the closer the LRT is to the time limit, the higher the likelihood of violating this constraint in a stochastic environment. As a result, the LRT of the

OBS is smaller. Additionally, our findings reveal a significant decline in the performance of the deterministic collection plan as the variability level increases. Figure 9.4 illustrates the distribution of stochastic total time results obtained from long simulations. The displayed boxplots correspond to the Very-Low (VL) and Very-High (VH) variability levels, comparing the OBD (depicted in pink) and the OBS (depicted in green). The cross-circle marker represents the mean value of each sample. In stochastic scenarios, our results demonstrate a clear reduction in total time when addressing the problem using a simheuristic approach rather than relying on a deterministic solution. The greater the variability considered, the more pronounced the reduction in total time. Moreover, the simheuristic approach minimizes result variability, as evidenced by the narrower range between the extreme points in each box plot.

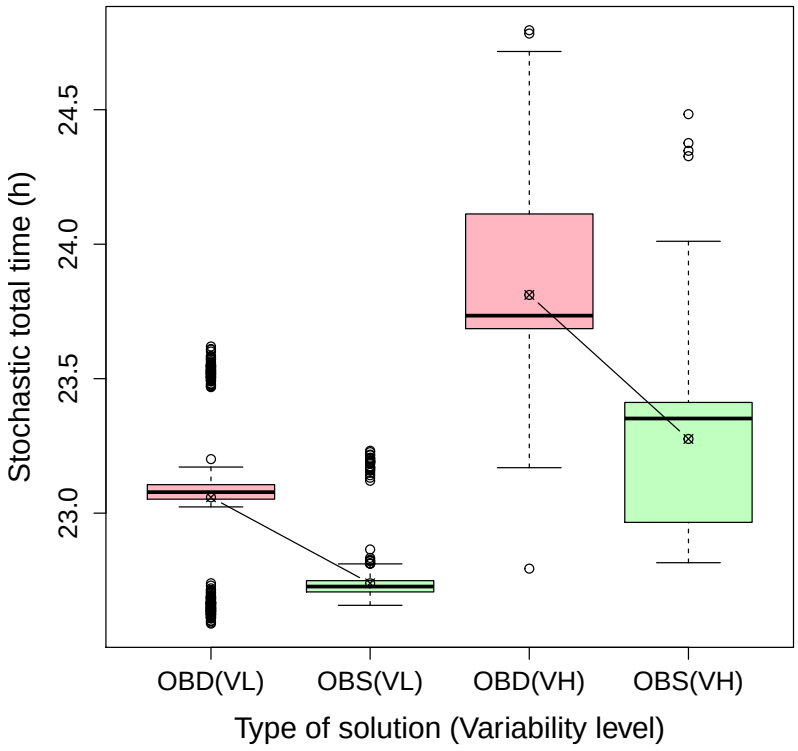


Figure 9.4: Total simulation time results across varying variability levels.

Part IV

Discrete-Event Heuristics

Chapter 10

Discrete-Event Heuristics Framework

10.1 Main Concept

The creation of discrete-event heuristics emerged as a response to the need for tackling realistic optimization problems where agent synchronization is essential. For example, in logistics, drivers must be synchronized with the arrival of riders or items needing transport. discrete-event heuristics achieve this by combining: (i) a discrete-event list that schedules system events (new events may arise later based on decisions made to address previous events, much like in any DES); (ii) a ranked list of possible decisions tied to each event, ordered by priority based on an efficiency criterion; and (iii) optimization techniques that generate numerous alternative solutions, all aligned with the efficiency criterion but allowing slight variations to explore new paths while managing the event list. As shown in Figure 10.1, the heuristic component handles decision-making, while the simulation updates the system state event by event, assessing how the heuristic's decisions impact the entire system, including its parallel processes, dynamic nature, and synchronization. This approach allows each decision to be based on the current, precise state of the system. A discrete-event heuristics offers greater flexibility and ease of maintenance, as system changes typically require only minor adjustments to the simulation component without altering the heuristic. Additionally, the simulation and heuristic components can function as separate entities, making it possible to implement them on different workstations or endpoints. This setup enables the same simulation to be used by multiple heuristics.

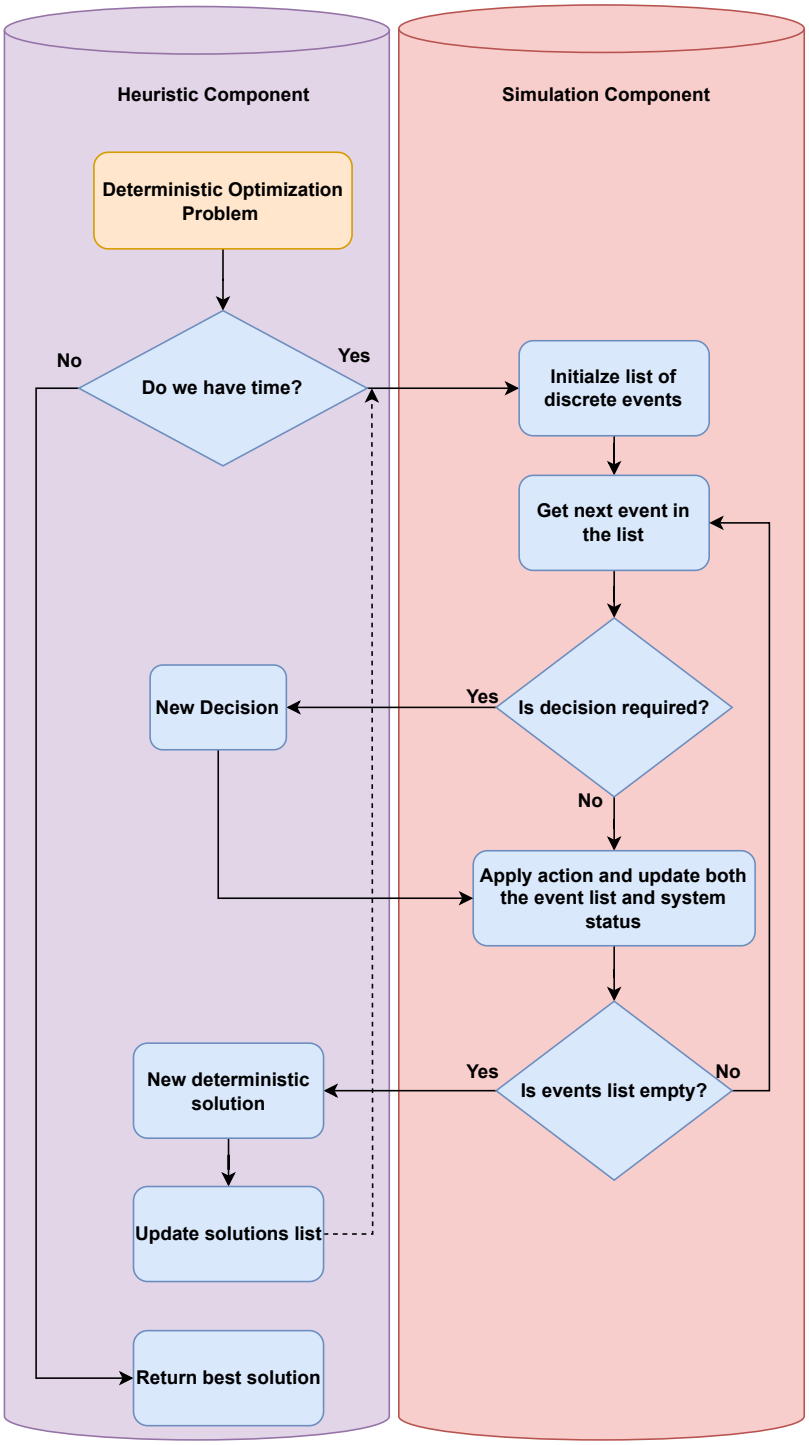


Figure 10.1: Basic scheme of the discrete-event heuristics.

Source:Castaneda et al. (2022)

10.2 Related Work

In the context of discrete-event heuristics, researchers have explored combining DES with heuristics to tackle NP-hard optimization problems that require synchronization (Table 10.1). For example, Fikar et al. (2016) proposed a BR discrete-event heuristics approach for a dynamic home service routing problem, where operators and vehicles adapt their assignments throughout the day based on new service requests and cancellations. This method uses an event list to update routes dynamically, minimizing the number of required vehicles. Bayliss, Martins, et al. (2020) applied this combination approach to solve the omnichannel vehicle routing problem, which involves coordinating store replenishment and online order deliveries. Their method combines BR heuristics with local search, achieving effective solutions. Arnau et al. (2022) used it to manage container deliveries with deadlines across a network of hubs. Their approach employs BR to select containers for transport and integrates an iterated local search to find efficient solutions. These techniques have also been applied to various flow-shop scheduling problems (Ferrer et al., 2016), vehicle routing problems (Belloso et al., 2019; Dominguez Rivero et al., 2016). They have also been used to extend and enhance traditional metaheuristic frameworks (Ferone et al., 2019). In manufacturing, Laroque, Leißau, Copado, Panadero, et al. (2021) and Laroque, Leißau, Copado, Schumacher, et al. (2022) utilized BR discrete-event heuristics for flow-shop problems in the semiconductor industry, optimizing job scheduling and batching to minimize makespan. They also integrated MCS for solution evaluation. Juan, Copado, et al. (2020) addressed a re-entry flow-shop problem where reprocessed jobs must return to the same machine. They used a discrete-event list and biased randomization to find solutions quickly. These contributions highlight the effectiveness of combining heuristics with discrete-event methods in addressing dynamic and complex challenges across transportation, manufacturing, and other domains.

Table 10.1: Summary of Related Work on Discrete-Event Heuristics.

Reference	Application Area	Contribution / Usefulness
(Fikar et al., 2016)	Dynamic Home Service Routing	Useful for dynamically adjusting operator and vehicle assignments to minimize the number of required vehicles.
(Bayliss, Martins, et al., 2020)	Omnichannel VRP	Useful for coordinating store replenishment and online order deliveries efficiently.
(Arnau et al., 2022)	Container Delivery	Useful for scheduling and managing container transport deadlines in a network of hubs.
(Ferrer et al., 2016)	Flow-Shop Scheduling	Useful for optimizing job scheduling in flow-shop environments.
(Belloso et al., 2019; Dominguez Rivero et al., 2016)	Vehicle Routing Problems	Useful for improving vehicle routing solutions.
(Ferone et al., 2019)	Metaheuristic Frameworks	Useful for enhancing traditional metaheuristic frameworks.
(Laroque, Leißau, Copado, Schumacher, et al., 2022)	Semiconductor Flow-Shop	Useful for job scheduling and batching in semiconductor production, with MCS integration for evaluation.
(Juan, Copado, et al., 2020)	Re-entry Flow-Shop	Useful for managing reprocessing of jobs in flow-shops.

Chapter 11

Discrete-Event Based Heuristic for Dynamic Ridesharing Problem

This chapter[1] presents a methodology for solving the DRSP in smart cities, where dynamic traffic conditions continuously influence decision-making processes. The proposed solution uses a discrete-event-driven metaheuristic, enhanced with BR techniques, to efficiently explore the solution space. The goal is to design optimal routes that maximize the total reward by picking up passengers, while adapting to real-time traffic updates and vehicle states. This approach addresses the challenges of handling large-scale, dynamic data generated by IoT devices, cloud computing, and edge computing in modern urban environments.

[1]The contents of this chapter is based on the following work:

- **Peyman, Mohammad**, Pedro J Copado, Rafael D Tordecilla, Leandro do C Martins, Fatos Xhafa, and Angel A Juan (2021). *"Edge computing and IoT analytics for agile optimization in intelligent transportation systems"*. In: Energies 14.19, p. 6309.

11.1 Problem Description

In today's modern society, urban centers are facing the so-called booming of information. Due to the population growth in many countries around the globe, and recent innovations in information and telecommunication technologies, several activities and related challenges have jointly arisen. People are increasingly consuming more information through their mobile devices, vehicles are equipped with different intelligent systems, devices are distributed around the cities for gathering and generating information, and urban areas are continuously taking advantage of these information technologies and big data. Consequently, the so-called smart cities have emerged, whose scope combines sustainable development with the intelligent management of gathered data in order to enhance the operation of different services within urban areas, such as waste collection management (Gruler, Quintero-Araújo, et al., 2017), carsharing/ridesharing activities (Martins, Torre, et al., 2021), optimal location of recharging stations for EVs (Frade et al., 2011), among others. In this matter, during the past few years, the IoT has become a popular term that plays a significant role to expand and produce a lot of data through sensors and allows citizens and things to be connected in any situation or with anyone (Beneicke et al., 2019). Also, fog and cloud computing come to support IoT to manage the large amount of generated data (Aazam et al., 2014). Integrating IoT and open data initiatives in smart cities allows governments, public, and private sectors to develop new services and applications by ensuring the effective handling and managing of data that are constantly shared among individual citizens and different industries (Ahlgren et al., 2016). For instance, sensing real-time traffic flow and mobility tracking data, such as vehicle states (e.g., location, speed, etc.), intersection information (e.g.,the length of the queue waiting at the intersection, etc.), and the situation of the road (e.g.,under repair, traffic accidents, etc.) can be analyzed (Tang, Xia, et al., 2019) for explaining the dynamics of urban vehicles as micro and macroscopic simulations, traffic flow, and travel time estimations (Jiménez-Meza et al., 2013). In this context, mobile internet technology is one of the actors which enables dynamic and on-demand sharing activities. In ridesharing systems, for example, people are allowed to offer trips for riders by using their own private vehicles, i.e., its core idea is to foster that personal private vehicles are shared by a group of people, instead of being used only by the car owner. Nowadays, the massive use of apps and interconnected smartphones facilitates the immediate contact between drivers and users for sharing trips. Furthermore, ridesharing activities provide multiple benefits for drivers, users, and the entire community (Bistaffa et al., 2019), such as reduction in costs, pollution, and traffic congestion. Hence, utilizing the cloud and edge computing helps to handle terabytes of data extracted from IoT devices, including information about vehicles mobility and traffic conditions. Also, by analyzing these data and combining it with the concept of ridesharing,

some mentioned urban problems can be reduced or even solved. In this context, optimization techniques, such as approximate methods –i.e., heuristics and metaheuristics– have been proved to be both efficient and capable of generating high-quality solutions for large-scale and complex real-world problems (Grasas et al., 2017). This means that heuristics have a high potential to provide agility and real-time responses, which are necessary issues for a good performance of intelligent transportation systems and, in general, of this type of systems. Nevertheless, after reviewing some related work, very few articles combining heuristics with IoT analytics by utilizing the cloud and edge computing have been found. Hence, in order to fill this gap, a DRSP is addressed in this section, where dynamic conditions usually encountered in modern urban centers affect the decision-making processes. In other words, the DRSP considers dynamic traffic conditions that might lead to several changes on the initially designed routes due to the incorporation of updated information, such as traffic conditions and vehicles states. In this problem, a set of routes must be designed so that the total reward collected by picking up passengers is maximized. A discrete-event driven metaheuristic is proposed to solve this problem. This solution method is enhanced with BR techniques to provide an efficient exploration of the solution space. Hence, our main goal is to propose a methodology for solving the DRSP.

11.2 Problem Modeling

The static version of the ridesharing problem (Martins, Torre, et al., 2021) consists of a finite set of capacitated heterogeneous vehicles, each one driven by an individual owner, who offers empty seats to users with similar itineraries. Each user requests a service, providing their current location, and drivers pick them up in these locations. This means that drivers have some kind of flexibility to adapt their routes so they visit the pickup point. Moreover, the vehicle capacity allows for more than one user to be transported. Hence, the route performed by each vehicle consists of an origin point (each driver's home), a set of locations where the driver picks up the users, and an arrival point. We assume that a driver can pick up each user only if the destination of all of them is the same. However, the destination points can be the same or different for each vehicle. Since the vehicle capacity and the number of vehicles are limited, not all users requesting a service can be picked up. Hence, the challenge is not only to design the routes, but also to select the users that will be picked up. This selection process is carried out based on both the distance between the user location and the driver's origin and destination points, as well as the fee that is paid by the user to the driver for being transported. The objective of the ridesharing problem is then to maximize the total collected fee. Figure 3.2 displays an example of a complete solution for the static version of the addressed problem. Connected houses represent the users who are served by the vehicles, whereas non-connected

houses represent the non-served users.

The static version of the ridesharing problem assumes that, once all routes have been designed, they cannot be further modified. Nevertheless, real-world events, such as traffic conditions, new orders, or cancellations, may lead to make changes in the original route plans. Since these events occur frequently when vehicles are already in route, a DRSP allows for the design of new routes, which include only those users who have not yet been picked up. This redesign process is performed in discrete time intervals. Formally speaking, the DRSP can be defined on a directed graph $G(N, E)$, where N is the set of nodes, and E is the set of edges linking these nodes, i.e., $E \subseteq N \times N = \{(i, j) \mid i \in N, j \in N, i \neq j\}$. Three subsets of nodes are considered, such that $N = I \cup O \cup A$. I is the subset of nodes where the users are located, O is the subset of drivers'/vehicles' origin nodes, and A is the subset of final destination nodes. Each pickup point $i \in I$ has a known fee f_i, which is paid by each user for being transported. Traversing each edge $(i, j) \in E$ has a deterministic cost c_{ij}. Routes are performed by a set K of vehicles. The capacity b_k of each vehicle $k \in K$ is known as well. Each pickup point $i \in I$ must be visited only once, and each vehicle $k \in K$ is assigned to only one route. Only a subset of nodes $J \subset I$ can be visited. Each route starts in an origin node $o \in O$, traverses a subset of nodes $H \subset J$, and finishes in a destination node $a \in A$. If d_h is the demand of each node $h \in H$, then $\sum_{h \in H} d_h \leq b_k$.

If t is the time interval set to recalculate the routes, then this process is always performed in the time $\tau = nt$, where $n \in \mathbb{N}$. This recalculation is performed iteratively until all vehicles have arrived to their respective destinations. Furthermore, if L_n is the subset of non-visited nodes in the period n, such that $L_n \subseteq J$, then the route recalculation is performed only for the nodes in both L_n and A. This assumption is helpful to respect the commitment of serving the originally selected customers, taken in the period $n = 0$. We assume that routes are only affected by traffic conditions, i.e., new orders and cancellations are not allowed in our addressed version of the problem. Hence, our problem consists in designing a set of $|K|$ routes that meet the aforementioned constraints, such that the total collected fee is maximized.

11.3 Methodological Approach

This section outlines our proposed methodology for addressing the DRSP. The approach is built upon a discrete-event heuristic (Fikar et al., 2016), which produces promising solutions by responding to events as they unfold over time. These events correspond to situations where the system must respond appropriately, such as adjusting routes in reaction to changes in traffic conditions. Discrete-event constructive heuristics utilize DES to manage time-dependent factors that emerge during the solution construction process. In our approach, the core concept is to perform a DES that tracks arrivals, departures, and route re-planning. This allows vehicles to adjust their routes based on traffic information provided by the open data server, aiming to minimize travel time to their final destination. This re-planning process is executed at each time interval t. As a result, events can be categorized into three types: vehicle delivery, vehicle arrival, and traffic update. Each event is linked to a specific vehicle and a trigger time. Additionally, the vehicle is associated with its current trip (traveling between two customers) and the route it is following.

The flowchart of our proposed solving approach is shown in Figure 11.1. Initially (at period $n = 0$), the algorithm generates a static plan without accounting for traffic conditions, meaning that all information used during this stage remains unchanged. Furthermore, the event list is initialized by adding one departure event for each available vehicle, along with a traffic update event. Departure events are scheduled to take place at period $n = 0$, while the traffic data event is set to occur at period $n = 1$. Each period n spans t time units, where t is defined as an input parameter. The main loop processes a list of events that occur throughout the execution of the routes, continuing until the list is empty. During each iteration, the algorithm selects the first event and performs actions based on its type. For a departure event, the vehicle departs from the current customer to the next destination in its trip. Consequently, a new arrival event is scheduled, taking into account the travel time and the prevailing traffic conditions. When an arrival event occurs, there are two possible outcomes: the vehicle either arrives at a customer location or reaches its final destination. If it arrives at a customer, the vehicle must pause to carry out a pick-up action. Following this, the vehicle's available capacity is adjusted to account for the passenger demand. Moreover, the algorithm schedules a new vehicle departure event from the customer. If the vehicle reaches the final destination, no further action is needed. For the traffic update event, which occurs at each period n, the algorithm re-plans the routes for the vehicles, taking into account the current traffic data, starting from the next stop until the final destination. Algorithm 5 presents the heuristic for solving the DRSP during a specific period n. This method is a two-stage heuristic algorithm designed to strike an effective balance between solution quality and computational effort. The input parameters consist of: a list of customers,

where each customer is defined by its location coordinates, passenger demand, and associated fee; a list of vehicles, with each vehicle specified by its origin and destination coordinates, as well as its seating capacity; and the α and β parameters, which are used for generating the savings list and for configuring the BR heuristic, respectively. In the first stage, the original problem is broken down into smaller sub-problems (clusters) based on the origin and destination points.

It is important to note that different sub-problems may share certain pick-up locations. In the second stage, each cluster is addressed by applying a savings-based heuristic introduced by Panadero, Juan, Bayliss, et al. (2020). The heuristic consists of the following steps: (i) creation of an initial dummy solution, where each pick-up point (node) is linked by a single route (vehicle) to the origin and final destination, and (ii) construction of an enhanced savings list of edges, where each savings value corresponds to an edge connecting locations such as the origin, destination, and pick-up points. The enhanced savings value is calculated as $s_{ij} = \alpha(c_{im} + c_{0j} - c_{ij}) + (1-\alpha)(\mu_i + \mu_j)$, where the input parameter α is within the range $(0, 1)$; c_{ij} represents the travel time between nodes i and j. Additionally, nodes 0 and m denote the origin and destination, respectively. Finally, μ_i and μ_j represent the fees assigned at each node. These savings values account for both the travel time and the total fee collected by visiting locations i and j. The main loop continues to iterate as long as the list is not empty. In each iteration, the edge at the top of the list is selected. The associated routes are then merged only if the resulting route does not exceed the vehicle's capacity; otherwise, the edge is discarded. This heuristic is deterministic since the merging process always picks the first element from the savings list. To introduce a broader range of solutions while preserving the core logic of the original heuristic, we extend it by incorporating a BR process. This is achieved by using the geometric probability distribution, Geom(β), with $\beta \in (0, 1)$, to introduce BR behavior. In Algorithm 6, the BR heuristic is executed up to a predefined limit of iterations or computational time, resulting in a MS approach (Martí, 2003). This process generates several feasible and promising solutions, from which the one yielding the highest collected fee is selected. The input parameters for Algorithm 6 are the same as those used in Algorithm 5.

Algorithm 5 Two-stage approach algorithm for solving a static RSP

input:
  customers: list of customers
  vehicles: list of vehicles
  α: parameter for computing the savings list
  β: parameter for the geometric distribution
end input
function `Biased-Randomized Algorithm`(customers,vehicles,β,α)
clusters ← computeClusters(customers,vehicles)
for $cluster_k$ in clusters **do**
    sol_k ← dummySolution($cluster_k$)
    savings ← genSavingsList($cluster_k$,α)
    while savings $\neq \emptyset$ **do**
        $edge_{ij}$ ← pick(savings,β)
        r_i, r_j ← getRoutes($edge_{ij}$)
        if checkMerging(r_i, r_j,vehicle($cluster_k$)) **then**
            $route_i$ ← merge($route_i$,$route_j$)
            sol_k ← replace(sol_k,$route_i$)
        end if
        savings ← remove(savings,$edge_{ij}$)
    end while
    sol ← add(sol,sol_k)
end for
return sol
output:
  sol: a solution
end output

Algorithm 6 Multi-Start Approach for solving RSP

input:
  customers: list of customers
  vehicles: list of vehicles
  α and β: parameters required by the BR algorithm heuristic
end input
function `Multi-Start`(customers,vehicles,β,α)
bestSol ← Heuristic(customers,vehicles)
while end not reached **do**
    sol ← BRA(customers,vehicles,β,α)
    if fee(sol) > fee(bestSol) **then**
        bestSol ← sol
    end if
end while
return bestSol
output:
  bestSol: the best solution
end output

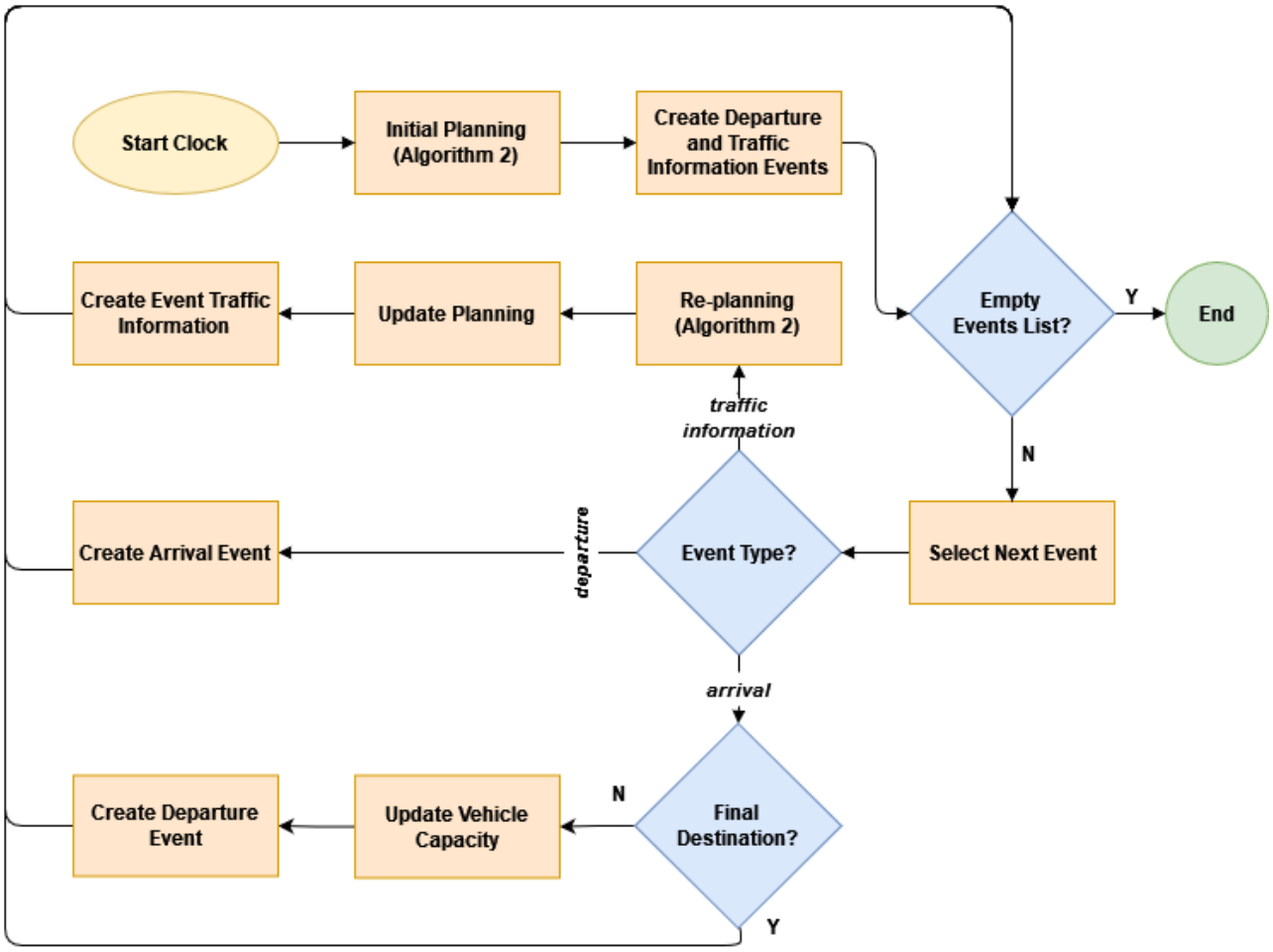


Figure 11.1: Discrete-Event Algorithm flowchart.

11.4 Computational Experiments

A series of numerical experiments were designed to test our approach. A total of 27 instances with different characteristics were tested. The instance name in Table 11.1 sets both the number of customers requesting a service and the number of available vehicles. For example, *drsp63x6-1* is the first instance in the list considering 63 potential customers and 6 vehicles. Hence, three groups of instances with different sizes are tested. Each potential customer's demand and location were generated randomly. Available vehicles are heterogeneous in each instance, with capacities varying between 4 and 8 users. The aggregated capacity of all vehicles is proportional to the total demand. Only for experimental purposes have the traffic conditions also been generated randomly for each edge $(i, j) \in E$ in each period n. These conditions are represented by a coefficient w_{ij}^n, which was generated according to a uniform probability distribution, such that $w_{ij}^n \sim U(0, 1)$. For real-world cases, w_{ij}^n can be computed after retrieving the corresponding traffic information from open data repositories. w_{ij}^n affects the cost c_{ij} incurred when traversing this edge. For instance, if c_{ij}^0 is the time spent for going from customer i to customer j in the period 0, i.e., before the vehicles' departure, then the simulated real time for traversing the edge (i, j) in the period n, affected by the traffic conditions, is computed according to Equation (11.1).

$$c_{ij}^n = c_{ij}^0(1 + w_{ij}^n) \tag{11.1}$$

Finally, the time interval that triggers the re-computation of routing plans is set in $t = 10$ time units. Notice that the number of times that routes are re-computed is instance-dependent, since these new computations are performed iteratively until all vehicles have arrived at their corresponding final destinations. The algorithm was implemented in Python 3.8. The experiments were carried out on an *i*7-8750 CPU at 2.20 GHz with 16 GB of RAM memory installed. The time limit for the BR process was set as 1 s (in order to keep the real-time condition).

11.5 Analysis of Results

Table 11.1 presents the results obtained from applying our approach to the considered instances. The restricted number of vehicles and their limited capacities make it impossible to visit all potential customers. Therefore, the maximum number of customers served is 23, 35, and 43 for each group of instances, respectively. The total fee collected from serving these customers is also displayed. Next, we use our algorithm to generate two distinct solutions: the best static solution (OBS) and the best dynamic solution (OBD). The OBS represents a solution generated at period 0 (the initial solution), which cannot be altered, meaning the vehicles follow the same static routes regardless of traffic conditions. On the other hand, the OBD is a solution that is recalculated every 10 time units, allowing it to adapt to the changing traffic conditions. The recalculation is carried out in the same manner as the initial solution, but it only considers the remaining customers to be served and the cost per edge as defined by Equation (11.1). The total collected fee remains the same for both the OBS and the OBD, as the customers chosen for service in the initial solution are retained in all recalculations for the OBD. In this case, only the routes may change. Lastly, the gaps in Table 11.1 represent the percentage variation between the costs obtained by the OBD and those obtained by the OBS. A negative gap indicates that the OBD achieves a lower cost than the OBS. The average gaps are consistently negative, regardless of the instance size. Only 4 out of 27 instances exhibit a positive gap, with the highest value being 2.43% for the instance *drsp63x6-6*. In contrast, the most significant negative gap is -35.61%, recorded for the instance *drsp43x4-5*. These findings highlight that, cost-wise, our dynamic approach consistently surpasses the scenario where the solution remains fixed and does not adapt to external fluctuations. Figure 11.2 displays a series of box plots that show the costs obtained by the OBS (in pink) and the OBD (in green). Each plot represents a group of instances, categorized by their size. The mean value of each data group is indicated by a crossed circle. The figure illustrates the expected increase in costs as the instance size expands. However, this increase is offset by the corresponding rise in the total collected fee, as demonstrated in Table 11.1. Additionally, Figure 11.2 visually highlights the cost savings achieved by using our dynamic approach as opposed to a static metaheuristic.

Table 11.1: Obtained results by our algorithm for a DRSP.

Instance	**Served Customers**	**Total Collected Fee**	**OBS Cost**	**OBD Cost**	**Gap**
drsp43x4-1	17	237	595.80	384.14	−35.53%
drsp43x4-2	18	295	587.88	492.58	−16.21%
drsp43x4-3	20	373	512.51	398.23	−22.30%
drsp43x4-4	22	394	502.89	402.97	−19.87%
drsp43x4-5	15	223	582.33	374.95	−35.61%
drsp43x4-6	22	347	528.00	528.70	0.13%
drsp43x4-7	16	261	497.84	367.13	−26.26%
drsp43x4-8	23	468	528.30	455.39	−13.80%
drsp43x4-9	18	313	650.22	418.83	−35.59%
Average	19.00	323.44	553.97	424.77	−22.78%
drsp63x6-1	33	676	685.87	635.04	−7.41%
drsp63x6-2	34	678	777.73	735.61	−5.42%
drsp63x6-3	31	494	811.37	726.05	−10.51%
drsp63x6-4	35	700	786.22	721.99	−8.17%
drsp63x6-5	30	515	714.26	665.46	−6.83%
drsp63x6-6	33	560	827.48	847.60	2.43%
drsp63x6-7	30	506	799.50	654.05	−18.19%
drsp63x6-8	28	383	873.49	791.88	−9.34%
drsp63x6-9	34	608	782.46	650.52	−16.86%
Average	32.00	568.89	784.26	714.24	−8.92%
drsp83x8-1	40	582	1114.75	1020.69	−8.44%
drsp83x8-2	38	722	1411.50	1040.39	−26.29%
drsp83x8-3	39	660	1310.40	1173.44	−10.45%
drsp83x8-4	43	740	1050.79	1051.51	0.07%
drsp83x8-5	34	516	1155.35	882.54	−23.61%
drsp83x8-6	39	661	1092.90	1019.06	−6.76%
drsp83x8-7	41	730	1124.13	1019.06	−9.35%
drsp83x8-8	40	726	1315.08	1340.66	1.95%
drsp83x8-9	39	668	1212.02	1052.88	−13.13%
Average	39.22	667.22	1198.55	1066.69	−10.67%

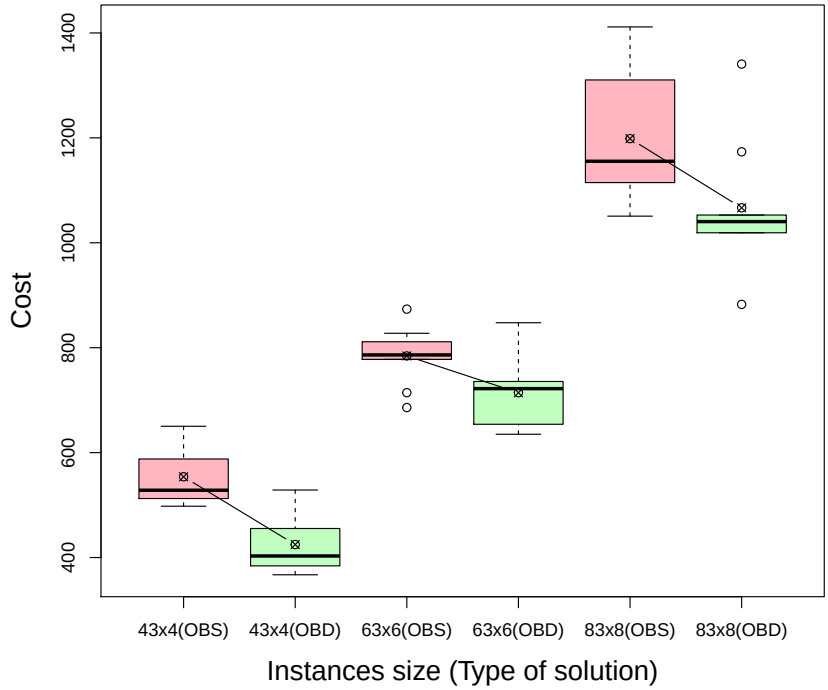
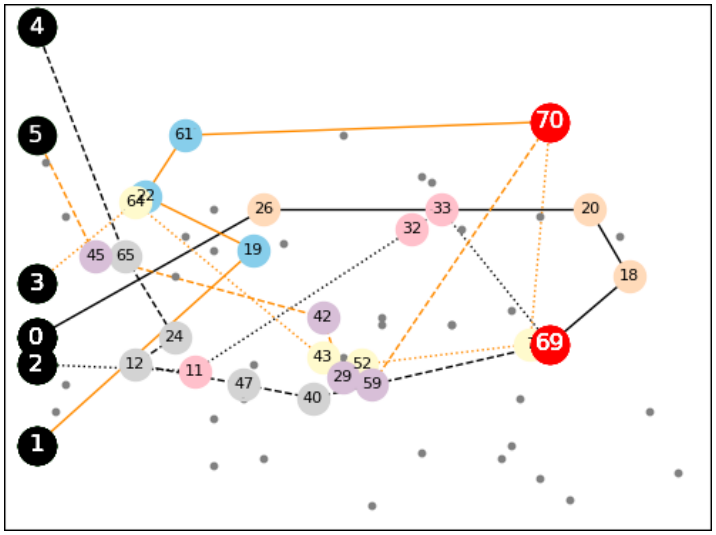


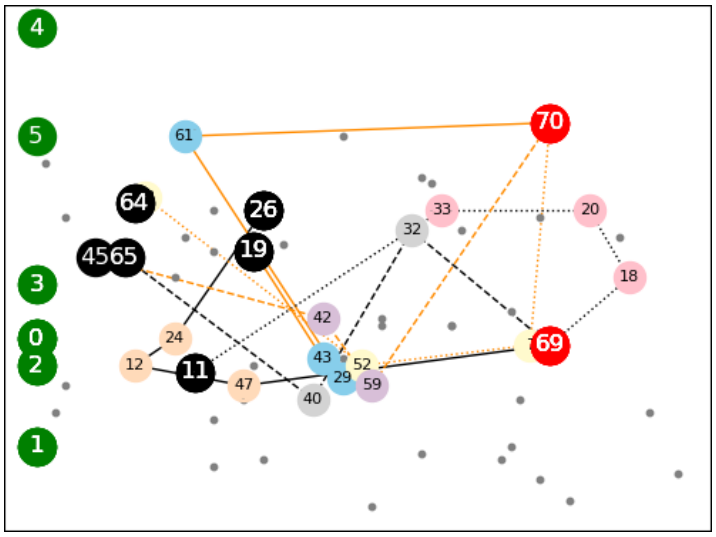
Figure 11.2: Costs obtained by each type of solution for different instance sizes.

Figure 11.3 illustrates an example of our algorithm's execution over two consecutive periods for the instance *drsp63x6-2*. The origin points are represented by large black nodes, while the destination points are marked by large red nodes. Medium-sized numbered nodes correspond to the served customers, and small unnumbered gray points indicate the non-served customers. The routes are illustrated by lines of different colors and patterns. Figure 11.3a displays the initial Best Found Solution (BFS), created during period $n = 0$. If this set of routes is not altered, the OBS is achieved, with a cost of 777.73 (Table 11.1). Alternatively, Figure 11.3b illustrates the BFS at period $n = 1$, where dynamic conditions are taken into account. Consequently, the original routes have been adjusted, resulting in the final OBD. At this point, each vehicle has already arrived at the location of its initial consumer. Our algorithm considers these places as new origin points; hence, the former origins are represented by green nodes in Figure 11.3b. It is important to note that the routes after the black nodes in this figure vary from those shown in Figure 11.3a. For instance, the black dashed route in Figure 11.3a follows the sequence *4-65-24-12-47-40-69*. In the period $n = 1$, the corresponding vehicle has already traveled from node *4* to node *65*. Nevertheless, the dynamic traffic conditions makes the original planned route change and the vehicle follows the new sequence *4-65-40-32-69*. The total customers selected in the period $n = 0$ remain the same, i.e., the commitment of serving them is respected, although the routes are different now. Given the limited space, the new re-computed routes for the rest of the periods are not displayed; however, this process is repeated until every vehicle has

arrived at nodes *69* and *70*, respectively. All these successive recalculations lead the attainment of the OBD, at a cost of 735.61 (Table 11.1).



(a) Period $n = 0$.



(b) Period $n = 1$.

Figure 11.3: Example of the BFS for the instance *drsp63x6-2* in the periods $n = 0$ and $n = 1$.

Chapter 12

Integration of Python with Discrete-Event Simulation

Simulation is a powerful tool for studying real-life systems with uncertainty, particularly through DES, which models time-dependent and complex systems. While commercial (AnyLogic, Simul8, FlexSim) and non-commercial (SimPy, Salabim) DES software provide accurate models, they often lack advanced analytical and optimization capabilities. Integrating DES with external programming languages like Python enhances these tools, allowing for more sophisticated analysis and greater flexibility in solving complex problems. In this chapter[1] we provide a step-by-step tutorial for simulation users to develop intelligent DES models by integrating them with Python.

[1]The contents of this part is based on the following work:

- **Peyman, Mohammad**, Pedro Copado, Javier Panadero, Angel A Juan, and Mohammad Dehghanimohammadabadi (2021). *"A tutorial on how to connect python with different simulation software to develop rich simheuristics"*. In: 2021 Winter Simulation Conference (WSC). IEEE, pp. 1–12.

12.1 Problem Description

Simulation packages are a perfect tool to model complex and dynamic systems with non-linear interactions. Several review papers and tutorials are written to enable users building simulation models using a simulation software or languages. Based on a subjective evaluation of parameters, Dias et al. (2016) made three clusters of simulation tools. The first cluster includes ProModel, FlexSim, Simul8, and WITNESS. The second cluster discusses ExtendSim, Simio, PlantSimulation, and AnyLogic. Finally, the third cluster covers SimProcess, AutoMod, Micro Saint, QUEST, Enterprise Dynamics, and Process Model. Arena appears in a different cluster. In another work, Schriber et al. (2012) provided several examples in AutoMod, SLX, ExtendSim, and Simio to guide user with simulation modeling. Besides, Dagkakis et al. (2016) reviewed an open-source DES with application in manufacturing, services, supply chain management, and logistics. In addition, authors evaluated and identified the best fitted simulation tools for different modeling needs. Later, Guimarães et al. (2018) developed a new method for selecting the right DES software to help and improve the efficiency of processes. In certain scenarios, simulation models need to be empowered with external environments to support advanced calculation, optimization, or evaluation. This highlights the importance of intelligent simulation modeling, where a DES platform is connected with a powerful programming language such as R, Python, or MATLAB. Due to the ongoing growth of ML techniques and optimization algorithms, this combination becomes even more necessary than ever. However, there is a lack of tutorials on how to efficiently combine simulation software with data science programming languages such as Python. Yuriy et al. (2008) connected AutoMod and Simul8 with an genetic algorithm engine to optimize equipment reliability assessment. Dehghanimohammadabadi and Keyser (2017) developed an intelligent simulation model by integrating MATLAB and Simio to perform multiple optimization analysis. In a similar integration, Dehghanimohammadabadi, Rezaeiahari, et al. (2017) created a simheuristic model for a patient scheduling optimization. Also, Ham (2018) introduced a new open-source and object-oriented package called Salabim. This package is developed in Python, and it includes an animation engine. Salabim has been used, for instance, to simulate a complex control system in logistics and production environments. Liu (2020) introduced Simulus, which supports event-driven and process-oriented simulation by utilizing Python packages and available modules.

Python is a popular language in the field of data science and AI. It is widely used to solve statistical and scientific problems. Combining simulation software with Python will provide enormous advantages for experimentation, optimization, results analysis, and professional data visualization. Additionally, it is an open-source language supplemented with multiple libraries for scientific computing, ML, optimization, data science, and big data (Raschka et al., 2019).

Therefore, this chapter aims to promote the development of *intelligent* simulation modeling by combining different simulation environments with Python. Some of our main goals are:

- To provide some insights on the use of an open-source simulation package such as SimPy.
- To show how it is possible to integrate commercial simulation packages (i.e., AnyLogic, Simula8, etc.) with Python.
- To show how the combined approaches can be used in practice to develop powerful simheuristics.

12.2 SimPy and Python

SimPy is an object-oriented and method-based open-source library. It is a powerful tool for developing DES models. Since it is built in the Python environment, interfacing SimPy models with other Python algorithms is seamless and trivial. Using SimPy, a user can create simulation models by including methods, messages, supplies, and vehicles as active components. Additionally, users can get access to other Python libraries to extend their modeling needs. Hence, SimPy is a non-commercial DES platform that can be used to simulate different models, such as production planning, hospital operations, and vehicle routing (Sanguesa et al., 2019). This section shows how to build a DES model using SimPy, and how to perform a simple execution. To install SimPy, the *pip install simpy* command has to be prompted.

This example models several EVs sharing a unique charging station. This model considers 3 types of EVs, each one with a different charging time (12, 24, and 36 hours), whilst the charging station provides only 2 charging points. The simulation works as follows: each vehicle starts to request a charging point, if any is available, then it gets parked and proceeds to be charged during its charging time. Otherwise, it waits until one becomes available. When an EVs releases the charging point, it completes the cycle carrying out a 5 hours trip. This cycle repeats over and over for a specified time horizon of 3 days. The Figure 12.1 shows the implementation of this model using Python-SimPy. This code defines a class (*ElectricVehicle*) with two methods, *run()* and *charging()*. SimPy is initialized by calling *simpy.Environment* to create an object environment *env*, which represents the simulation environment. Then, *env* is passed to the *ElectricVehicle* constructor to create an instance of a *car*. In this example, 3 cars are created. Each car starts its trip for 5 hours and stops at the charging station for recharging. There are only 2 charging stations, so one car needs to wait until another car gets charged. The waiting time defined in three different hours (*charging times = 12, 24, 36*). In addition, *simpy.Resource()* is used to limit the number of simultaneously used processes. As there is a

```python
class ElectricVehicle:
    car_id = 1
    def __init__(self,env,charging,station,dtrip):
        self.env=env
        self.car_id = ElectricVehicle.car_id
        self.charging_duration=charging
        self.charging_station = station
        self.action=env.process(self.run())
        self.duration_trip = dtrip
        ElectricVehicle.car_id += 1
    def run(self):
            while True:
                yield self.env.process(self.charging(self.charging_duration))
                print(f'car {self.car_id} starts trip at {self.env.now}')
                yield self.env.timeout(self.duration_trip)
    def charging(self,duration):
        with self.charging_station.request() as req:
            yield req
            print(f'car {self.car_id} is going to charge at {self.env.now} duration {self.charging_duration}')
            yield self.env.timeout(duration)
            print(f'car {self.car_id} is charged at {self.env.now}')
import simpy
nb_stations = 2
duration_trip = 5
nb_cars_types = 3
charging_times = [12,24,36]
time_horizon = 3*24
env=simpy.Environment()
charging_stations = simpy.Resource(env, capacity=nb_stations)
for i in range(nb_cars_types):
    ElectricVehicle(env,charging_times[i],charging_stations,duration_trip)
env.run(until=time_horizon) #3 days
```

Figure 12.1: Simpy model.

limitation in the number of charging stations, these can be modeled as a resource. Hence, EVs arrive at the station and place a request to get a full charge. After recharging, cars start their trips. Figure 12.2 shows the experimental results with driving and parking starting times. This model can be extended by including additional cars, charging stations, and key performance indicators, such as waiting times.

12.3 Calling Python from AnyLogic

The AnyLogic (AL) simulation platform is a modern simulation software based on the object-oriented paradigm. Borshchev (2013) introduced AL as an efficient software that can support different modeling methods, such as system dynamics, discrete-event, and agent-based. AL supports multiple types of simulation experiments. For instance, optimization and parameter variation. Besides, different application problems, such as transportation systems, supply chain management, industrial development, and business process evaluation can be modeled in AL Merkuryeva et al. (2010) and Muravev et al. (2021). A simulation model in AL includes a set of active objects that work simultaneously and interact with each other. The active object is the main structure of the model, which enables users to develop their customized modification. For more details about features and tutorials refer to this website How to learn AnyLogic.

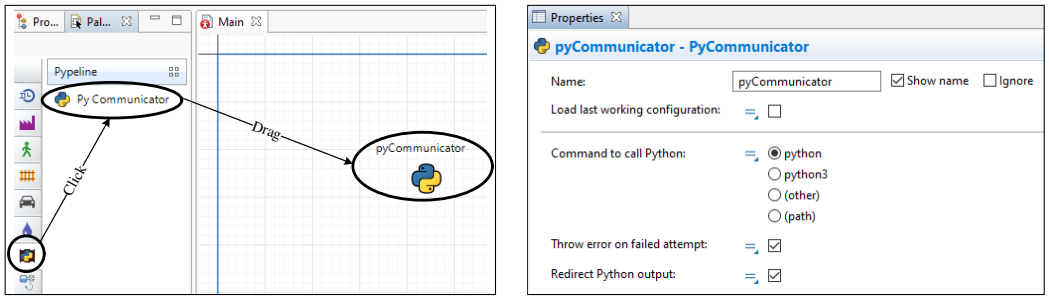
A key advantage of AL is its capability to integrate with Python packages. This enables the

```
car 1 is going to charge at 0 duration 12
car 2 is going to charge at 0 duration 24
car 1 is charged at 12
car 1 starts trip at 12
car 3 is going to charge at 12 duration 36
car 2 is charged at 24
car 2 starts trip at 24
car 1 is going to charge at 24 duration 12
car 1 is charged at 36
car 1 starts trip at 36
car 2 is going to charge at 36 duration 24
car 3 is charged at 48
car 3 starts trip at 48
car 1 is going to charge at 48 duration 12
car 2 is charged at 60
car 1 is charged at 60
car 2 starts trip at 60
car 1 starts trip at 60
car 3 is going to charge at 60 duration 36
car 2 is going to charge at 65 duration 24
```

Figure 12.2: SimPy result.

simulation model to interact with external codes written in Python. This connection is supported by a Pypeline library written in Java. Using this library, AL users can link the simulation model with any customized functions and algorithms developed in Python environment to support experimentation.

Pypeline (*pypeline.jar*) can be downloaded from github.com and added to the AL palette. After installation, the Pypeline library will be added to the AL environment (Figure 12.3a). At this point, the AL model is ready to connect with Python by adding an object called *pyCommunicator* to the main model. As shown in (Figure 12.3b), Pypeline is compatible with any Python installation, and it uses the same version as it is installed on the user's machine. Therefore, to complete the pyCommunicator settings, a user needs to select an appropriate Python version and add the Python path (the directory in which the Python is installed.)



(a) Pypeline icon (pyCommunicator) on AL palette.

(b) Linking Pypeline to Python local installation path.

Figure 12.3: AL Pypeline library.

Pypeline provides two methods of communication. The first function is *run*, which is the one-directional statement to send commands to Python. This function can be used in many instances such as import statements, variable assignments, and function calls. The second

function is *runResult*, which is two-directional. This function not only sends statements to Python, but it also can receive returns based on Python calculations. This functionality allows users to change the model configuration, conduct experiments, and even perform optimization from a Python environment. Both of these functions take a string as an input to represent execution codes. *runResults* returns a custom object, *Attempt*, which includes two outputs: *(i) isSuccessful*, a Boolean indicating whether the Python command is successful or not; and *(ii) getFeedback*, which shows the model error if it exists.

To illustrate how AL calls Python, a simple simulation model is created based on the the flowchart depicted in Figure 12.4. This model contains a *source* that is the main agent, a *queue* (a placeholder for agent to wait for the next process), and a *delay* that calls the Pypeline file. The delay time directly depends on the Python file execution time or could vary based on the computation size and complexity. There are two text elements in the model, *calls* and *result*, which represent the *number of Python calls* and the *returned value*, correspondingly. After finishing the process (e.g., the Python computation in this case), the agent leaves the system from the *sink*.

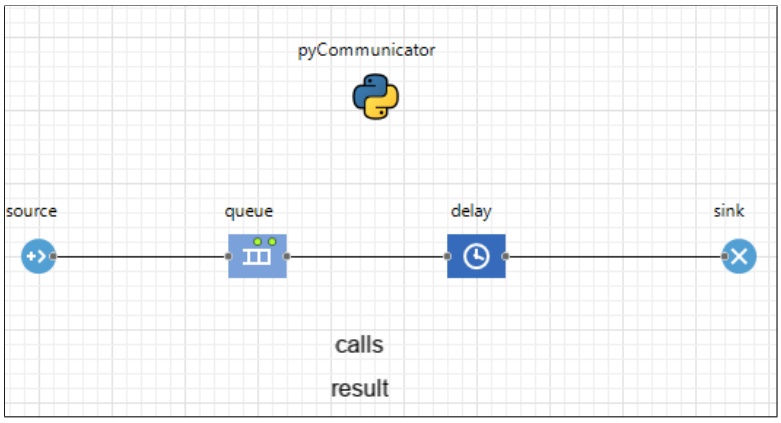


Figure 12.4: AL simple model.

The Python script that AL calls in this example is a continuous optimization problem to minimize the *basinfunction* function defined as follows: $f(x) = \sum_{i=1}^{n} a(x_i - h)^2 + k$, where $x \in [-0.5, 0.5]$ is the decision variable, with constant parameters $n = 2$, $a = 0.5$, $h = 2$, and $k = -5$. To solve this problem, a *randomSearch* algorithm is coded in Python (Figure 12.5).

This code includes a function called *Solver*. This function runs the *randomSearch* to minimize the *basinfunction* objective function. This algorithm creates a new solution (*newsol*) by randomly selecting a value for x in each iteration. Once the search is complete, the best ever found solution (*bestSol*) and its corresponding cost (*bestCost*) are returned. To run the Python code from the AL model, the model properties need to be edited properly. This change can be done as follows:

1. From the *main* tab, select the *main-agent type*.

```
import math, random, time
## Basin function code
def basinfunction(vector):
    a,h,k = 0.5, 2, -5
    # based on formulate we have some constant value a=0.5 ,k=-5, h=2
    sum = 0
    for item in vector:
        sum = sum + a * pow((item - h),2) + k
    return sum

def randomSearch(searchSpace, problemSize):
    #search space is min and max range of cordinate
    min = searchSpace[0] #min value in searchspace
    max =  searchSpace[1] #max value in searchspace
    inputValues = [] #list of values
    for i in range(0, problemSize):
        inputValues.append(min + (max - min) * random.random())
    return inputValues

##Solver Framework##
def Solver(maxIterations=1000,problemSize=2):
  bestSol = None
  searchSpace = [-5, 5]
  bestCost = float('inf')
  while maxIterations > 0 :
            newsol = randomSolution(searchSpace, problemSize)
        newCost = basinfunction(newsol)
        if newCost < bestCost:
                bestSol = newsol
              bestCost = newCost
        maxIterations -= 1
    return(bestCost)
```

Figure 12.5: Random search algorithm.

2. Modify the *Agent action* by changing the *on startup* settings (Figure 12.6):

   - Syntax 1: *pyCommunicator.run("import filename")*, which imports the Python file.
   - Syntax 2: *pyCommunicator.run("result, calls=0,0")*, which calls the Python function and receives its results.

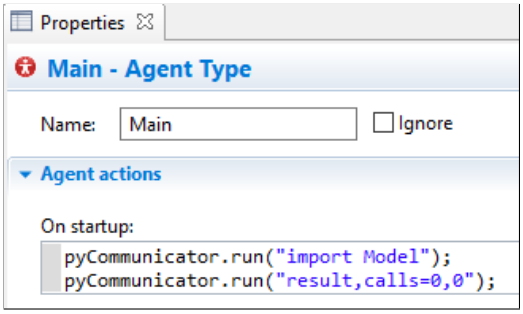


Figure 12.6: Call Python file.

3. From the *delay* object, modify the *on exit* event setting to enable the AL model to trigger the Python code (Figure 12.7). These changes are as follows:

   - Syntax 1: *pyCommunicator.run("calls += 1")* to increment the number of times the Python files are called.
   - Syntax 2: *pyCommunicator.run("results = round(Model.Solver(), 4)")* deploys the random search algorithm written in the *Solver* function. The number 4 determines how many decimal points of accuracy have to be included in the results.
   - Syntax 3: *Attempt attempt1 = pyCommunicator.runResults("calls")* to enable the AL model to trigger Python.
   - Syntax 4: *Attempt attempt2 = pyCommunicator.runResults("results")* to enable the AL model to receive Python results.

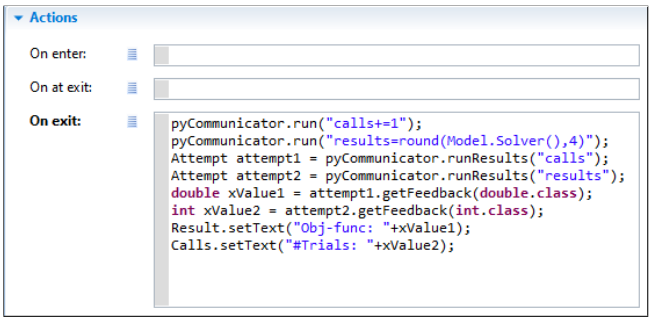


Figure 12.7: On exit properties of "delay".

- Syntax 5 to 8: The rest of the codes are for matching the datatype between Python and AL, as well as for displaying the results on the AL screen (Figure 12.8).

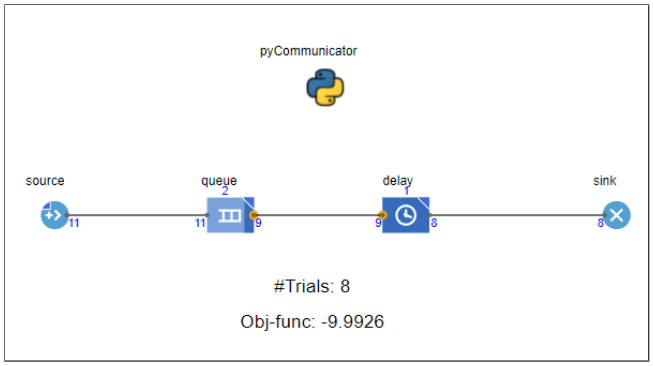


Figure 12.8: The structure of AL model integrated with Python.

12.4 Calling AnyLogic from Python

AL provides a cloud API for Python, which gives the capability of running simulation models in the cloud. Performing models in the cloud have the advantage that it may release computational and hardware requirements. To learn how to use this API, interested users can refer to AL cloud website. This section, shows how to take advantage of this API based on an existing Service System Demo model. This model contains a single *source*, a *service module*, a *checkpoint*, and one *sink*, as it is shown in Figure 12.9.

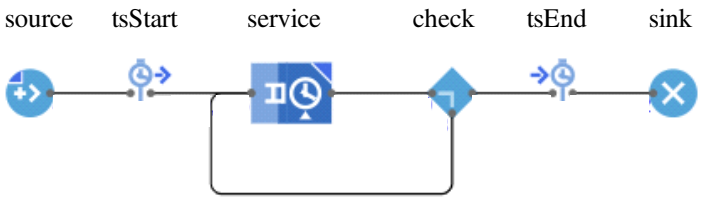


Figure 12.9: Service System Demo Model.

The model simulates a service process and allows a user to perform experiments by changing the model configuration (e.g., the server capacity). The model outputs are the average size of the service queue and the resource utilization. The following steps provide guidelines to develop AL models from cloud client in Python (Figure 12.10):

```
1  #install the AnyLogic cloud client library
2  pip install https://cloud.anylogic.com/files/api-8.5.0/clients/anylogiccloudclient-8.5.0-py3-none-any.whl
3
4  # Load anylogiccloudclient library
5  from anylogiccloudclient.client.cloud_client import CloudClient
6
7  #Create a CloudClient object, given the API key
8  client = CloudClient("e05a6efa-ea5f-4adf-b090-ae0ca7d16c20")
9
10 #Obtain latest model version of "Service System Demo" model
11 model = client.get_latest_model_version("Service System Demo")
12
13 #Create an Inputs object with the default input values
14 inputs = client.create_inputs_from_experiment(model, "Baseline")
15
16 #Change the "Server Capacity" parameter value
17 inputs.set_input("Server capacity", 10)
18
19 #Create a simulation object with the inputs
20 simulation = client.create_simulation(inputs)
21
22 #Obtain the simulation outputs
23 outputs = simulation.get_outputs_and_run_if_absent()
24
25 print("For Server Capacity = " + str(inputs.get_input("Server capacity")))
26 print("Mean queue size = " + str(outputs.value("Mean queue size|Mean queue size")))
27 print("Server utilization = " + str(outputs.value("Utilization|Server utilization")))
```

Figure 12.10: Calling the AL model via Python using the AL cloud client.

1. Install the AL cloud client library by using the *pip* package manager from the command line (line 2).

2. Import the AL cloud client library, and set the client object API key: **e05a6efa-ea5f-4adf-b090-ae0ca7d16c20** (lines 4 to 8).

3. Create a *model* object using the *client.get_latest_model_version()* function. This makes the interaction between Python and AL to be seamless for inputs, outputs, and dashboard configurations (line 11).

4. Establish the experimental settings *input* using *client.create_inputs_from_experiment()*. The *input* value can be user-defined (e.g., *Server capacity* is set to 10) (lines 14 to 17).

5. Develop the remote AL simulation model using the *client.create_simulation()* function and embedding the experimental inputs. (line 20).

6. Retrieve the simulation results by triggering the simulation experiments with *simulation.get_outputs_and_run_if_absent()* (line 23).

Figure 12.11 shows the experiment results of the aforementioned model with a server capacity of 10 units. These results show the mean service queue size of approximately 1 person, and the average Server utilization of 0.25.

```
For Server Capacity = 10
Mean queue size = 0.9999193295506064
Server utilization = 0.2501690832867859
```

Figure 12.11: Output from the Python example using the AL cloud API.

12.5 Calling Simul8 from Python

Simul8 is a simulation software that is used for simulating the discrete-entities processing at the discrete time. It is useful not only in industry, but also in academia. Simul8 visualizes the processes by using 2D animation, and it is suitable for DES. Also, it operates with 6 main parts, such as work item, entry point, storage bin, work center, work exit point, and resource (Elder, 2014). This section elaborates on how to interface Simul8 with Python based on the example provided in Simul8.com. It needs to be noted that, the 32-bit version of Python is recognizable by Simul8 as a COM application (you should use *pip install pywin32*). The following steps explain the syntax codes provided in Figure 12.12:

1. Import the required packages (lines 1 and 2).

```
1  import win32com
2  from win32com import client
3  from win32com.client import makepy
4  makepy.main()
5  class EventHandler:
6      def OnS8SimulationEndRun(self):
7          n=1
8          while (n <= S8.ResultsCount):
9              print(S8.Results(n))
10             n +=1
11 S8 = win32com.client.Dispatch("Simul8.S8Simulation")
12 S8.Open(r"C:\Program Files (x86)\SIMUL8_P_T\Examples\OptQuest\Call_Center_1.s8")
13 events = win32com.client.WithEvents(S8, EventHandler)
14 S8.RunSim(300)
15 S8.Quit()
```

Figure 12.12: Python code for running the model.

2. Import *makepy*, which is a specific file for *win32com* – it is required to create COM modules out of COM-enabled applications (line 3).

3. Enter into the *Makepy* file by running the Python code (line 4), and then select the *Simul8 Library* from the list (Figure 12.13).

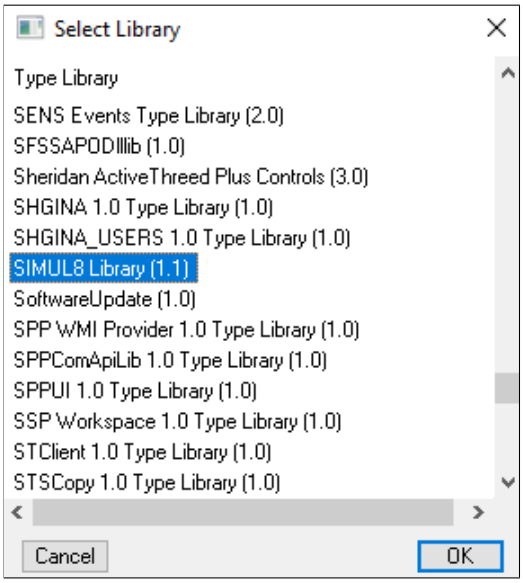


Figure 12.13: Simul8 library.

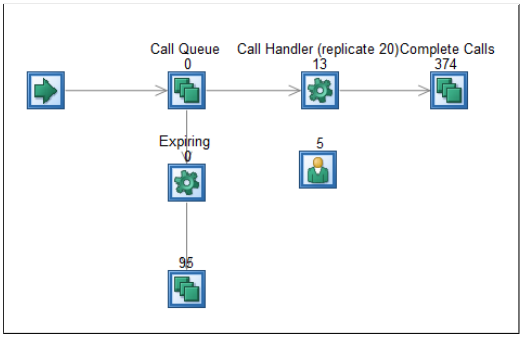
4. Create a class called *EventHandler*, which counts the number of times tha Simul8 is called, and prints the simulation results (lines 5 to 10).

5. Use the dispatch method to open the Simul8 software (line 11).

6. Open the developed simulation model (Figure 12.14a) (line 12).

7. Connect the *EventHolder* class with Simul8 (line 13).

8. Run the simulation experiment for 300 minutes (line 14) and then terminate the the execution (line 15). Finally, Figure 12.14b demonstrate the result.



(a) Simul8 example model.

```
Generating...
Importing module
18.0
18.0
18.0
13.0
18.0
85.0
85.0
4.0
374.0
374.0
0.0
```

(b) Result in Python.

Figure 12.14: Simul8 model and results.

Chapter 13

Flexsim-Python Integration for Discrete-Event Simulation Simheuristics

In chapter 12 we explained that connecting commercial DES packages to external software or programming languages is essential to advance simulation modeling capabilities. We showed how it is possible to integrate commercial simulation packages (i.e., AL, Simula8, etc.) with Python. This chapter[1] provides a step-by-step guideline to connect the FlexSim commercial simulator with the popular Python programming language via sockets. Using this type of connection, a simheuristic algorithm coded in Python aims at optimizing the product allocation in a warehouse, which has been previously modeled in the aforementioned simulator.

[1]The contents of this part is based on the following work:

- Leon, Jonas F, Paolo Marone, **Mohammad Peyman**, Yuda Li, Laura Calvet, Mohammad Dehghanimohammadabadi, and Angel A Juan (2022). *"A tutorial on combining Flexsim with Python for developing discrete-event simheuristics"*. In: 2022 Winter Simulation Conference (WSC). IEEE, pp. 1386–1400.

13.1 Problem Description

Python is an interpreted high-level general-purpose open-source programming language (Van Rossum et al., 2009). It has a diverse, wide, and international community of programmers. Calling simulation software from Python may represent limitless advantages for data processing and manipulation, experimentation, optimization, results analysis, and visualization. This is due to the multiple libraries for ML, optimization, data science, and big data already available for this language, as well as to the possibility of developing powerful metaheuristic algorithms over it. The DES platform selected for this section is FlexSim, which is an object-oriented software environment used to develop, model, simulate, visualize, and monitor dynamic-flow process activities and systems. It supports DES, as well as agent-based simulation and continuous simulation, enables 3D simulation modeling and analysis, and has diverse potential applications in many fields, among others: manufacturing, material handling, healthcare, and warehousing.

In this context, this chapter presents a two-fold goal: *(i)* to explain the integration between FlexSim and Python; and *(ii)* to describe an illustrative example by implementing a simheuristic algorithm designed to support decisions in warehouse management. Simheuristics represent a hybrid methodology that extends (meta)-heuristics through simulation to solve stochastic combinatorial optimization problems (Juan, Kelton, et al., 2018). Thus, this methodology enables us to deal with real-life uncertainty in a natural way by integrating simulation (in any of its variants) into a metaheuristic-driven framework. The combinatorial optimization problems studied in this work is the SLAP, which is an *NP-hard* problem of great importance in supply chain management. As described by Reyes et al. (2019), the SLAP is "an operational decision associated with the accommodation and picking process, influencing batch definition, classification, routing, and order sequencing". Typically, the objective functions are related to warehouse space utilization and the time required for order preparation and picking operations. Diverse restrictions such as available storage capacity, order-picking resource capacities, and dispatching policies may be considered as well.

13.2 Connection between FlexSim and Python

This section explains the technical details associated with the creation of a link between FlexSim and Python. In Figure 13.1 the general scheme of the connection and intercommunication process to be developed in this paper is presented.

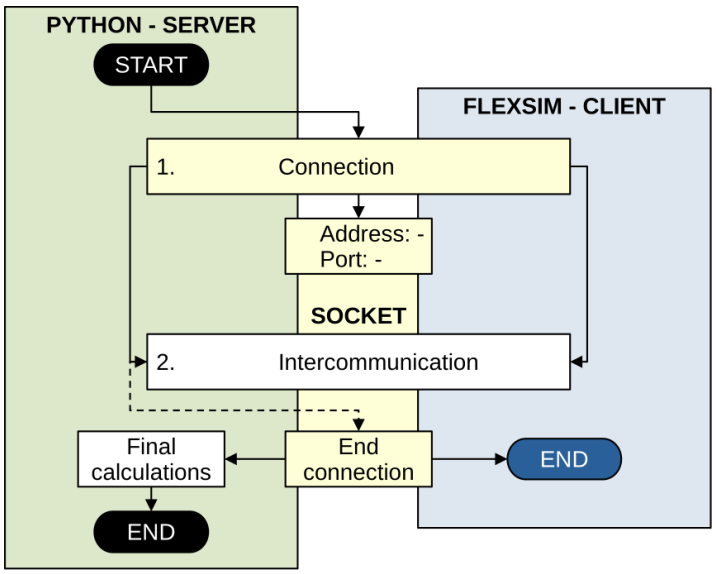


Figure 13.1: High-level flow diagram of the communication process.

13.2.1 General Considerations

There are multiple ways of connecting FlexSim and Python (and in general, any two pieces of software). This connection might depend upon the type of interaction intended or the function to be performed by each part. First of all, it is important to state that FlexSim is a commercial application (or program), and that Python is a programming language. Therefore, what is shown in this tutorial, is how to connect and communicate from a Python script to a single instance of FlexSim. A practical aspect to be defined beforehand is the roles for each part of the connection, namely, what piece of software will start the connection, what would be the protocol of communication, and how will the communication be terminated.

13.2.1.1 Communication

In this tutorial, it is assumed that the Python script behaves as the “server”, and FlexSim as the “client”. In reality, both are communicating on par with each other, but it is Python the one that starts the communication process, the one that listens for request/messages from FlexSim and provides an answer, and the one that eventually closes the simulation and the

communication channel. Also, in theory, multiple instances of FlexSim could be communicating with the Python "server". The communication is established through sockets. Sockets are a robust, standard and potentially secure way for applications to communicate with each other, allowing the extension of the simulation software capabilities (Rodrigues et al., 2005). The process starts with Python creating a socket with the specified address and port. By default, the "localhost" address and any port above port "1024" (e.g., "5005") can be used for creating a local connection. Afterwards, Python starts an instance of FlexSim, which in turn tries to establish a connection through the same socket (i.e., the same address and port).

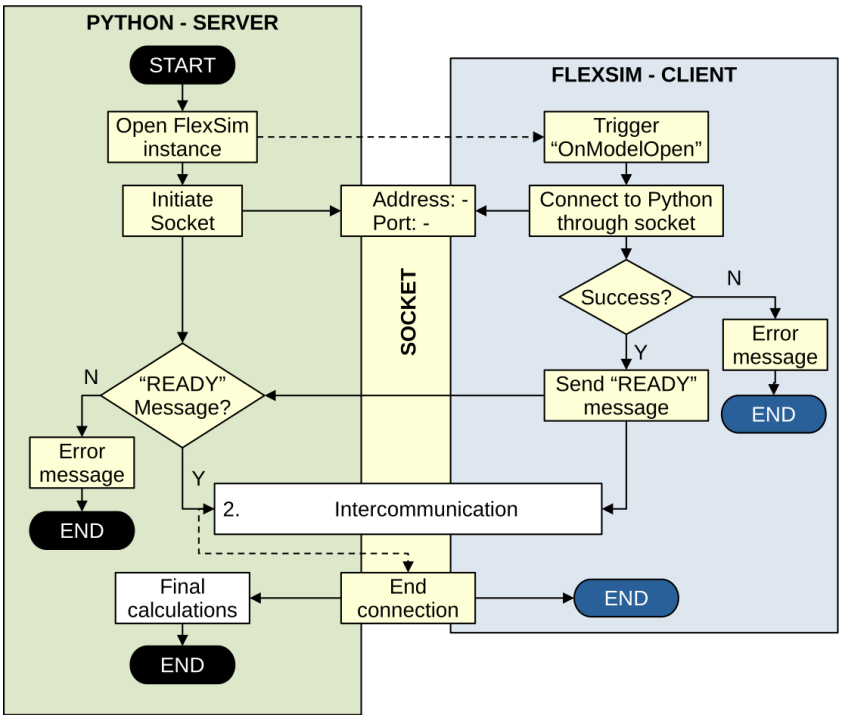


Figure 13.2: Detailed flow diagram of the connection process.

13.2.1.2 Server Side (Python)

The information provided in this section is based upon various examples available at https://www.flexsim.com. All the code shown herein is for illustrative purposes only, and has been simplified to a "minimal working example" size. The creation of the socket connection, which is the start of the connection process shown in Figure 13.2, is done by instantiating a Python *class*, which contains all the necessary functions for connection and communication. The simplified code, for declaring the Python class and the functions within it, is shown in Figure 13.3. The libraries for socket connection and for running sub-processes are imported at the beginning of the code. The functions defined within the class are listed below:

- *_socket_init*: opens the socket at the specified address and port, using the Python "socket" library.
- *_socket_end*: closes the socket at the specified address and port, using the Python "socket" library.
- *_launch_flexsim*: starts a FlexSim instance using the Python "subprocess" library, and initiates the socket by calling the *_socket_init* function.
- *_close_flexsim*: kills the FlexSim sub-process and shuts down the socket using the *_socket_end* function.
- *_socket_send*: sends a message to the client (FlexSim).
- *_socket_recv*: receives a message from the client (FlexSim).

```
import subprocess
import socket
class FlexSimConnection():
    def __init__(self, flexsimPath, modelPath, address='localhost', port=5005):
        self.flexsimPath = flexsimPath
        self.modelPath = modelPath
        self.address = address
        self.port = port
    def _launch_flexsim(self):
        args = [self.flexsimPath, self.modelPath] # option to add aditional arguments
        self.flexsimProcess = subprocess.Popen(args)
        self._socket_init(self.address, self.port)
    def _close_flexsim(self):
        self.flexsimProcess.kill()
        self._socket_end(self.address, self.port)
    def _socket_init(self, host, port):
        self.serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.serversocket.bind((host, port))
        self.serversocket.listen();
        (self.clientsocket, self.socketaddress) = self.serversocket.accept()
        message = self._socket_recv()
        if message != b"READY":
            raise RuntimeError("Did not receive READY! message")
    def _socket_end(self, host, port):
        self.serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.serversocket.bind((host, port))
        self.serversocket.close()
    def _socket_send(self, msg):
        totalsent = 0
        while totalsent < len(msg):
            sent = self.clientsocket.send(msg[totalsent:])
            if sent == 0:
                raise RuntimeError("Socket connection broken")
            totalsent = totalsent + sent
    def _socket_recv(self):
        chunks = []
        while 1:
            chunk = self.clientsocket.recv(2048)
            if chunk == b'':
                raise RuntimeError("Socket connection broken")
            if chunk[-1] == ord('!'):
                chunks.append(chunk[:-1])
                break;
            else:
                chunks.append(chunk)
        return b''.join(chunks)
```

Figure 13.3: Class for connecting to the FlexSim environment.

13.2.1.3 Client Side (FlexSim)

The information provided within this section and the settings required to set up the client side of the communication are well documented in the aforementioned FlexSim website. The code is written in *FlexScript*, which is the programming language that FlexSim uses internally. Alternatively, FlexSim allows employing C++ dedicated *DLLs* for the same purpose. The connection have to start right after the FlexSim instance is created, if the program is to be run automatically and without external intervention. For that, a (global) event trigger called "OnModelOpen" can be used within FlexSim. This trigger, which fires when the FlexSim instance is opened, executes the necessary code for the connection. A simplified example of this connection code that is to be written inside the "OnModelOpen" FlexSim trigger is provided in Figure 13.4.

```
connecttoserver("127.0.0.1",5005);
clientsend(getnodenum(node("/Tools/client", model())), "READY!");
// Request global parameters from server (e.g. simulation end time) or import tables
applicationcommand("reset");
applicationcommand("run");
```

Figure 13.4: Trigger code for "OnModelOpen".

The "connecttoserver" function, which appears in the first line of the "OnModelOpen" trigger code, is in charge of establishing the connection through the socket. Obviously, the address and port must be the same as used on the Python server side. The rest of the lines are mainly for confirming the connection by sending a "READY" message (alternative "shake-hand" protocols can be defined). Then, additional messages can be exchanged in order to obtain global simulation parameters. Finally, the simulation is reset and started. The code of that "connecttoserver" function is provided in Figure 13.5, and can be found in the FlexSim website.

13.2.1.4 Synchronous Communication

The connection method explained in Section 13.2 could already be used to run a single instance of a FlexSim simulation with some initial parameters. However, this only allows for a somewhat limited algorithm to be deployed. In this section the intercommunication process will be explained further to enrich the capabilities of the proposed method. In terms of the type of interaction between software, it is common in the simulation-optimization field that the communication process is in certain sense "asynchronous". This means that the optimization is performed in order to find a good candidate solution and after that, the simulation is run in order to evaluate or confirm the goodness of the proposed solution. Alternatively, the simulation could be run in order to obtain certain stochastic parameters that are then input into the analytical optimization model. In any case, the communication is a one-off event, which happens

```
treenode connection = assertsubnode(node("/Tools",model()), "client", DATATYPE_NUMBER);
string hostname = parstr(1);
int portnum = parval(2);
int client = 0;
// Initialize the socket before trying to create a connection. socketinit() return TRUE
// if the initialization is successful and reutrns FALSE if it is not successful.
if(socketinit()) {
    client = clientcreate(); // Start the process to use Windows sockets as a client.
}
// clientcreate() returns TRUE if the process could start properly and FALSE if it couldn't.
if(client) {
    // Make the connection to this machine using given hostname and port number.
    if(clientconnect(client,hostname,portnum)) {
        // Set the client node created earlier with the connectin value.
        setnodenum(connection,client);
        return 1; // Return a 1 to indicate the connection was made.
    }
    else {
        msg("Client Error", "Failed to connect to server.");
        if(!clientclose(client))
        msg("Socket Error", "Failed to close client socket.");
        return 0; // Return a 0 to indicate an issue.
    }
}
else // The process for socket connection could not start.
    msg("Socket Error", "Failed to create client socket.");
return client; // Return value in client variable (0) to indicate an error.
```

Figure 13.5: “connecttoserver” function.

before or after the simulation run (Figueira et al., 2014). For this type of connection, FlexSim already provides some options, like passing of initial parameters to a predefined simulation, or outputting simulation values to an external file. In contrast, in the intended application shown in this paper, the interaction will be continuous or “synchronous”, meaning that the simulation software and the Python script are exchanging messages (maybe thousands) as the DES evolves, with potential re-optimizations on every step. The intercommunication process is detailed in Figure 13.6.

13.2.1.5 Implementation

After the connection is successfully established, the FlexSim simulation starts and the Python script remains listening. Like within any DES software, as the FlexSim simulation proceeds different events will occur (e.g., in a warehouse context, a new order is received, a worker finishes her task, etc.), thus triggering other events or actions. The key idea of this type of “continuous” connection is that the action to be executed by FlexSim is to send a message to the Python script through the socket, and wait for the answer. In this regard, FlexSim allows for a “blocking” or “non blocking” behavior. This means that the FlexSim program execution could be frozen until something is received from the Python server. Since there could be potentially multiple messages sent during a small fraction of the simulation time, the blocking behavior option is to be used in this type of communication, in order to avoid misunderstanding between both software due to the lag between message and response. On the Python side, each message

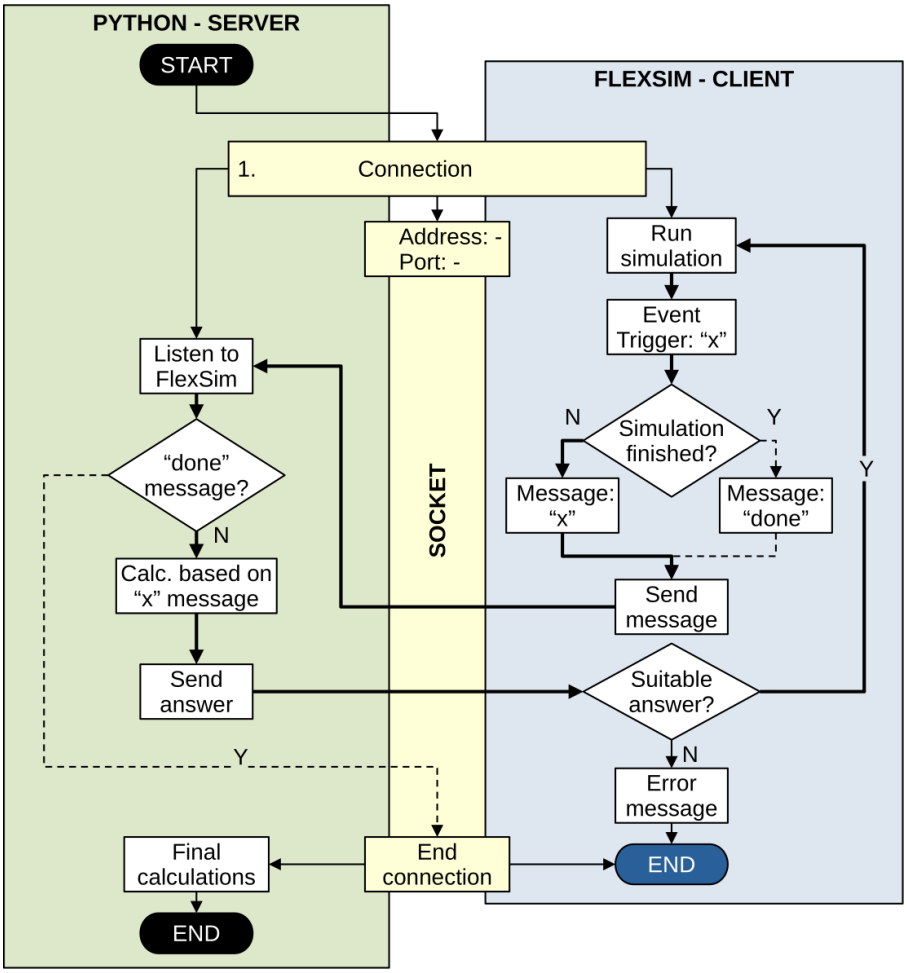


Figure 13.6: Detailed flow diagram of the intercommunication in the Simheuristic process.

received must be classified. This means that all possible events, that are defined in FlexSim as requiring input from Python, have to be mapped to an answer in the Python script. These Python answers are normally a calculation based on the status of the simulation. For that reason, it is important that the status is either contained in the message passed through the socket or through reading directly the information from an external table or a database. The answer to be sent by the Python script is always a string, and the "meaning" of that string is to be interpreted by FlexSim. This is what makes this communication protocol so versatile. In a sense, it could be understood as the basis for well researched intelligence enhancers for simulation, like simheuristics, reinforcement learning, or learnheuristics (Calvet et al., 2017). The Python *main* code, i.e., the one that controls the "runtime" execution (as opposed to the communication class definition), is shown in Figure 13.7. Once the *FlexSimConnection* class is instantiated (as *FS* in the code example), it can be used to open FlexSim and to send or receive messages. There can be some initial communication in order to establish some simulation parameters (e.g., for how long the simulation will run, or the number or resources to be employed). This parameter

setting needs to happen before the simulation starts running in FlexSim. The most important piece of code is the *while* loop, which keeps the Python server "listening" to the messages coming from FlexSim. Once a message is received, it is classified and replied using the *if* and *elif* logic.

```
if __name__ == "__main__":
    FS = FlexSimConnection(
        flexsimPath = "C:/Program Files/FlexSim 2022/program/flexsim.exe",
        modelPath = "C:/Users/.../simulation.fsm")
    FS._launch_flexsim()
    # Simulation initial set-up (optional):
    END_TIME = 1000 # example: time to end the simulation
    FS._socket_send(str(END_TIME).encode())
    # Running simulation:
    done = False
    while not done:
        try:
            # Receive message
            msg_recv = FS._socket_recv().decode('utf-8')
            if msg_recv == "x":
                ANSWER = calc_answer_x()
                FS._socket_send(str(ANSWER).encode())
            # Other message answer calculations in here...
            elif msg_recv == "done":
                done = True
        except:
            done = True
            FS._close_flexsim()
    FS._close_flexsim()
```

Figure 13.7: Main section of the Python script.

At specific events (e.g., every time that a product arrives to the warehouse), which are programmed beforehand in FlexSim, some information might be needed from the Python server script (e.g., where to place the new item optimally in the warehouse). For that, a message have to be sent, and an answer must be awaited. The FlexScript code shown in Figure 13.8 allows this event-driven intercommunication. The code can be adapted and used at almost every point where a decision have to be made, overriding the default FlexSim decision making. This is especially powerful when combined with the FlexSim *process flow tool* for defining the simulation logic.

```
clientsend(getnodenum(node("/Tools/client", model())), "x!");
string answer = clientreceive(getnodenum(node("/Tools/client",model())), NULL, 1024, 0);
// do something with the answer like assign it to a label
token.label = answer;
```

Figure 13.8: Sending and receiving messages via socket communication.

13.3 Problem Modeling

Warehouses are a crucial component in supply chain management. The improvement of efficiency in daily warehouse operations can provide significant benefits for companies. Typical activities in a warehouse include reception, storage, order-picking, and dispatch. Storage is defined as a

buffer of accumulated products to guarantee the quantities demanded in the shortest possible time. Product storage and retrieval are considered as the most crucial and resource-intensive activities in warehouses (Van den Berg et al., 1999). These processes imply product allocation decisions that affect the general performance of the storage systems. The storage location assignment problem is an operational decision problem that concerns the allocation of products into a storage space and the optimization of the storage space utilization. There are many solution approaches available in the literature, being one of the most common approaches based on policies and rules. The most representative policies include random storage, dedicated storage, and class-based storage. For a more comprehensive review of this problem, readers can refer to (Reyes et al., 2019). The SLAP that this paper tries to solve aims to locate, in a certain warehouse, N types of products in M spaces, where $M > N$ (i.e., there will be free spaces after all items are located). The goal of this problem consists in minimizing the traveled distance of all warehouse workers (both inbound and outbound). This minimization of the traveled distance increases the *throughput*, defined as the number of items dealt with per unit time, which is a measure of the efficiency of the warehouse. The size and shape of the warehouse has been chosen based on some toy examples from the literature (Xie et al., 2014). Figure 13.9 shows the scheme of the warehouse instance evaluated in this work. At this stage, the exact size of the warehouse is somehow irrelevant, since the main objective of the tutorial is to prove the advantage of implementing a simheuristic method that helps solving the SLAP.

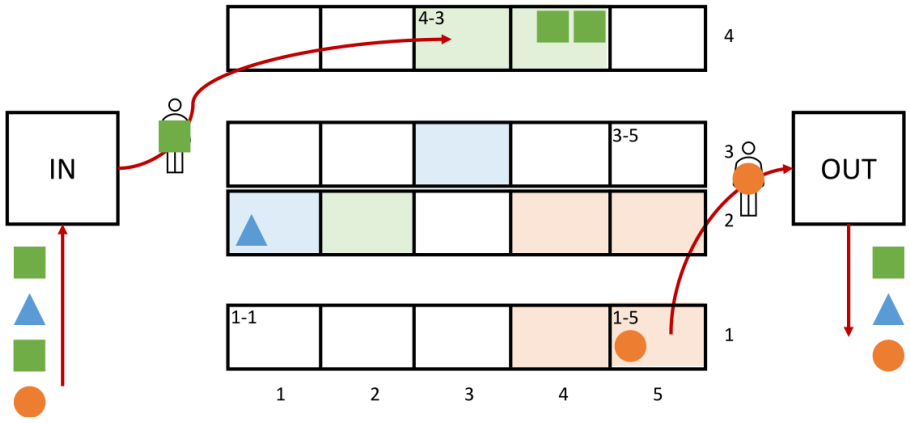


Figure 13.9: Problem description scheme.

As in real life, some types of product have higher arrival and pick-up frequency than others. Therefore, the products and the orders arrive at a certain rate (randomly), and according to a given probability distribution. As the products arrive to the inbound area, these are placed in their corresponding slot in the warehouse. Upon order arrival (which can be assumed to be a request for a certain type of product), a product needs to be retrieved from the warehouse

shelves. The problem, therefore, consists in assigning specific slots to the arriving products, such in a way that the overall efficiency of the process is optimized.

The two main policies in the literature that are proposed for tackling the SLAP are the *random assignment* and the dedicated or *fixed assignment* strategy. The former implies that the location for each type of product is decided randomly. The latter has a pre-assigned fixed location for each type of product. This pre-assignment is usually done based on the relative demand frequency for each product, trying to minimize the distance traveled by the workers. Each strategy has its own applicability and advantages depending on the warehouse configuration. For the warehouse instance investigated in this example, since the type of products N are less than the available slots M, the fixed location is expected to provide significantly better results. This is because the more demanded type of products can be placed closer to the exit reducing the workers' average traveled distance. As can be seen in Section 13.5, our simheuristic method was able to validate this expected behavior. In contrast with solving a "static" instance of the warehouse, different "dynamic" simulation-based policies can be tested using the intercommunication method presented in Section 13.2. The simulation part of the method represents a natural way for treating stochastic demand constraints, as well as allowing for a more realistic evaluation of the objective function (which takes into account the warehouse geometry, interaction between workers, sequence of operations, etc.). After the method is validated with the comparison between random and fixed policy, an *improved* storage location assignment policy was designed (see Section 13.4). This policy will try to beat the previous policies by incorporating information about the incoming warehouse orders "dynamically", i.e., during the simulation time.

13.4 Methodological Approach

The algorithm proposed to further improve the fixed allocation strategy was not an optimization based on the current status of the warehouse, but rather a heuristic action based upon the future orders to be fulfilled. In a certain sense, the list of orders that need to be prepared in the warehouse is indeed part of the current status of the warehouse, but neither the random or fixed strategies make use of this available information. For that reason, it is expected that incorporating this additional information will improve the warehouse efficiency. The improved heuristic for placing a new item in the warehouse is shown in Algorithm 7.

The logic followed can be explained as a sequence of actions (heuristic): *(i)* use the fixed location assignment as baseline; *(ii)* check if the next few orders (5 - 10) contain the item to place; *(iii)* check if there is available space closer to the exit than its default assigned place; and *(iv)* place the item in the best location. It is important to note that some simulation actions will

Algorithm 7 Improved SLAP

```
1: location ← fixed(status, item)
2: if item ∈ orders then
3:    loop: slot
4:    if out_distance(slot) < out_distance(location) then
5:       if has_space(slot) = true then
6:          location ← slot
7:       end if
8:    end if
9: end if
10: end
```

be performed by FlexSim using its own pre-programmed logic. This means that the proposed simheuristic controls only part of the simulation logic, and other parts are controlled by FlexSim (e.g., the first-in-first-out logic in some queues). For that reason, it can be expected that further improvements on the efficiency can be obtained if more actions/operations are mediated by the heuristic.

13.4.0.1 Intercommunication Sequence

The intercommunication sequence detailed in this section is based upon the more generic intercommunication process explained in Section 13.2 and depicted in Figure 13.2. Here, the more specific intercommunication protocol between Python and FlexSim will be explained. In particular, the communication-trigger events, its corresponding message and the calculated answer are listed below:

- *A new item arrives*: the message *"product"* is sent to Python. The type of product is calculated based on the probability associated with each type. FlexSim receives the item identification (ID) and the type of product to which the item belongs.
- *A new order arrives*: the message *"order"* is sent to Python. The type of product is calculated based on the probability associated with each type. FlexSim receives the type of product required in order to fulfill the order.
- *An item needs to be placed*: the message *"slot"* is sent to Python, followed by the associated type of product. The calculation is performed using the selected policy in the Python script. Three options are available: *1_random*, *2_fixed*, or *3_improved*. FlexSim receives the "address" where the item is to be placed according to its type.
- *A worker has finished her current job*: the message *"work"* is sent to Python. The next order is calculated based on a priority list and in the current status of the warehouse,

which is available in an external file. FlexSim receives the next order to be picked-up from the warehouse.

- *A stopping condition has been met*: the message *"done"* is sent to Python. The FlexSim application is closed and Python perform some final metrics calculation based on the status of the warehouse and the total distance covered by the workers.

Heuristic computations are fast, which allows to provide a prompt answer for the FlexSim simulation, which remains frozen in the meanwhile. In the previously described model, half an hour of simulation time is computed in approximately 10 seconds, with more than 100 messages passed back and forth.

13.5 Computational Experiments

Since the simulation is of stochastic nature, the results have to be treated statistically. In the example, the source of random variables is not in the arrival time for items and orders, but on the type of products that arrive and that are requested to be picked-up.

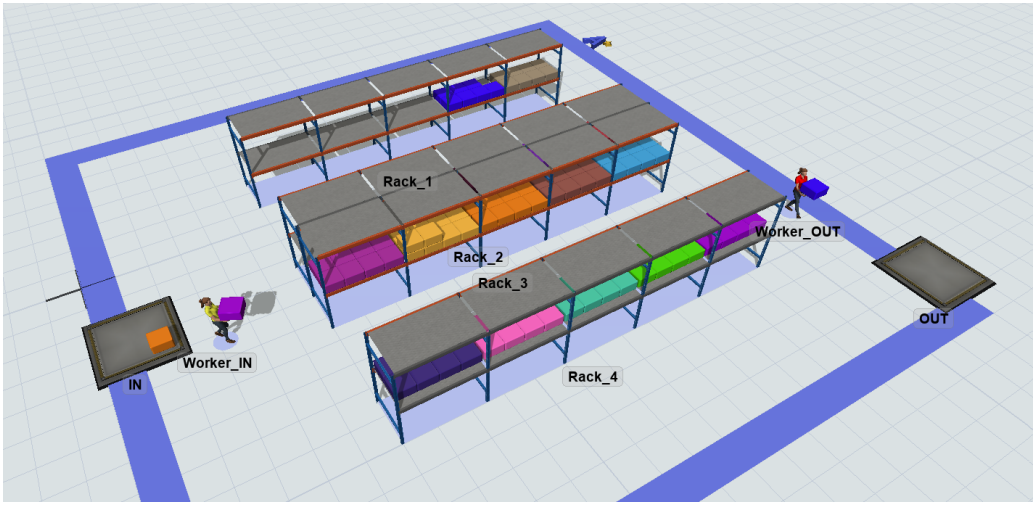


Figure 13.10: Screenshot from the FlexSim simulation.

13.6 Analysis of Results

The main results are summarized in Table 13.1 and in Figure 13.11. The metrics employed were the *throughput* (τ) and the *average distance* ($\bar{d}$). The throughput can be defined in different ways depending on the context, but for a warehouse it is usually measured as the number of items processed by unit of time. Specifically, the throughput in our example was defined as the

number of products that were output from the warehouse (n_{out}) divided by the total simulation time (t), as expressed in Equation (13.1):

$$\tau = \frac{n_{out}}{t} \tag{13.1}$$

Similarly, the average distance was defined as the number of meters "walked" by unit of output products. This is calculated by summing all the individual distances (d_i^w) covered in each trip i completed by each worker w, divided by the number of products that exit the warehouse, see Equation (13.2):

$$\bar{d} = \frac{\sum_i \sum_w d_i^w}{n_{out}} \tag{13.2}$$

Table 13.1: Results summary.

	τ		$\bar{d}$	
policy	mean	std. dev.	mean	std. dev.
1_random	227.8	6.9	62.7	1.9
2_fixed	280.2	9.9	51.0	1.8
3_improved	289.1	13.9	49.1	1.9

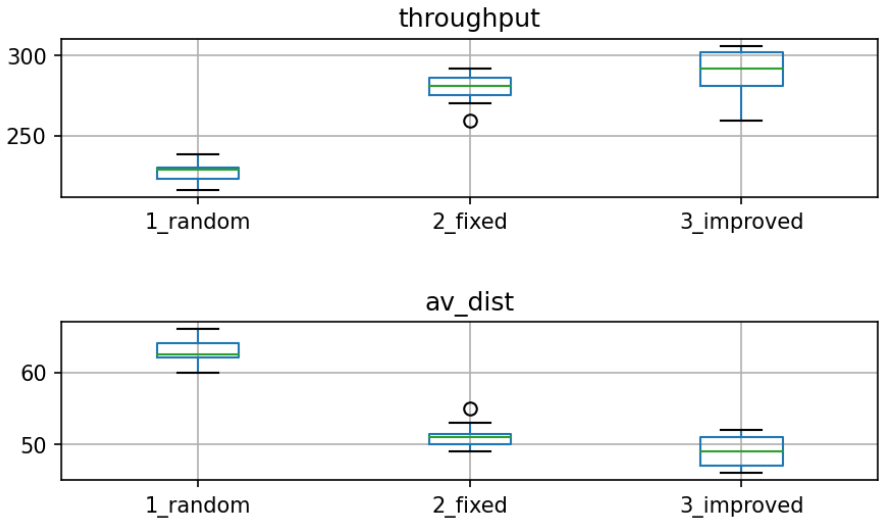


Figure 13.11: Simulation results comparing various SLAP strategies.

The results are the summary of 10 simheuristics runs for each policy (30 runs in total). It can be seen that implementing a dedicated or fixed assignment policy can indeed increase the efficiency of the warehouse. This is in line with the literature, and can be seen as a validation of the simheuristic model. The gains from introducing the improved placement logic (i.e., by

looking into the future orders) are much smaller in comparison, but they are still worth the implementation. Even modest improvements in warehouse efficiency can result in significant operational cost reduction, specially for large warehouses.

Chapter 14

Simulation of Heuristics for Automated Guided Vehicles Task Sequencing

AGVs are a key application of Industry 4.0, aimed at improving productivity while reducing labor costs, energy use, and emissions. However, implementing AGVs in specific industrial scenarios poses significant technological challenges. In this chapter[1] we explore the integration of Simul8 simulation software with heuristic algorithms, applied to a real-world case of task sequencing for AGVs. This problem involves dynamic queue formation and resource sharing, requiring advanced methods like heuristics and simulation. A heuristic procedure generates candidate task sequences, which are then evaluated using Simul8's DES. This approach provides high-quality solutions that can be visually assessed by stakeholders. Computational experiments validate the method and suggest future research directions, especially for stochastic environments.

[1]The contents of this part is based on the following work:

- Leon, Jonas F, **Mohammad Peyman**, Xabier A Martin, and Angel A Juan (2024). *"Simulation of Heuristics for Automated Guided Vehicle Task Sequencing with Resource Sharing and Dynamic Queues"*. In: Mathematics 12.2, p. 271.

14.1 Problem Description

Industry 4.0 offers significant opportunities to enhance productivity; however, it also introduces numerous technological challenges. These challenges primarily revolve around transitioning from human-supervised processes to automated, machine-controlled operations, which consequently result in issues related to the coordination and connectivity of distributed complex systems (Klingenberg et al., 2022). In this context, the adoption of AGVs has gained increasing relevance, primarily due to their ability to enhance productivity while simultaneously reducing labor costs, energy consumption, and emissions (Bechtsis et al., 2017). Moreover, AGVs serve as a paradigm for Industry 4.0, as their deployment necessitates the seamless integration of several key concepts and technologies, including adaptive robotics, embedded systems, communication and networking, cloud systems, simulation, virtualization, data analytics, and AI (Ustundag et al., 2018). Extensive research has been conducted on AGVs since their introduction more than 25 years ago (Vis, 2006). Despite this, several areas remain underdeveloped, and certain use cases continue to present technological and scientific challenges. The case discussed in this section is based on a real industrial application that proved challenging to model using existing AGVs problems described in the literature. The core of the problem lies in the dispatching, routing, and scheduling of tasks for multiple AGVs, although these terms are not entirely synonymous. According to Vivaldini et al. (2015), a distinction among these terms can be made as follows: (i) dispatching refers to the process of selecting and assigning tasks to vehicles; (ii) routing involves determining the specific paths that each vehicle must follow to complete the assigned tasks; and (iii) scheduling pertains to defining the arrival and departure times of the vehicles along their designated routes for each assigned task. Based on these definitions, addressing the scheduling problem assumes that the routing and dispatching problems have already been resolved. In this study, scheduling is viewed as a result of the sequence in which tasks are performed by the available resources at any given time. This implies that queue dynamics and resource sharing are the critical factors shaping the schedule, provided they are managed through a sufficiently representative model. The problem framed in this manner can be referred to as task sequencing, which involves determining the order in which the (already assigned and planned) tasks are executed. While the problem is trivial when resources are infinite and interactions between them are ignored, it becomes a significant combinatorial challenge when resources are limited and queue dynamics are taken into account. The coordination of multiple AGVs executing a given set of tasks, while accounting for interactions with other machines and resource-sharing constraints, constitutes a complex problem that can be modeled in various ways. For example, it can be formulated as a RTSP or a RCSP. Both the RTSP (Alatartsev et al., 2015) and the RCSP (Blazewicz et al., 1983) have been shown to be NP-hard.

Consequently, any version of the problem that can be regarded as a special case or a combination of these problems is also expected to be NP-hard. In other words, as the number of AGVs and workstations increases, capturing the interactions between individual AGVs and workstations using conventional mathematical methods becomes infeasible. Therefore, it is necessary to employ more advanced mathematical tools, such as heuristics, simulation, or a combination of both.

In this context, the application of algorithms and heuristics to obtain high-quality solutions within a reasonable timeframe is considered the preferred approach for addressing these types of problems, as opposed to exact methods that aim to find optimal solutions. The time required to determine the optimal solution for a problem with such an extensive solution space, combined with the gap between the optimal solution found and the true optimal solution (e.g., accounting for the randomness inherent in real-world systems), makes heuristics a practical and sensible choice (Pinedo et al., 2012). The methodology employed in this chapter involves identifying several algorithms commonly used for sorting non-uniform tasks and resources, and integrating them into a heuristic procedure designed to generate candidate solutions for the task sequencing problem. To enhance the fidelity of the proposed model, a DES model was developed using the Simul8 commercial software to assess the performance of the heuristic solutions. The use of simulation tools such as Simul8 offers several advantages in this context, including: (i) supporting the graphical validation of results by stakeholders, particularly technical personnel and decision-makers; and (ii) enabling the development of complex and detailed representations of real-world scenarios. Without the simulation component, evaluating the candidate solutions generated by the heuristic procedure becomes infeasible due to the dynamic nature of queues and the availability of shared resources, which involve temporal or sequential dependencies. Many existing approaches to scheduling or sequencing problems tend to oversimplify or neglect the complex interactions between available resources, often assuming that idealized solutions can be directly implemented in real-world settings. This limitation is further exacerbated in stochastic environments, which are typical of most real-world industrial applications. There is a pressing need for scalable methodologies that can handle these intricate interactions and adapt to more challenging scenarios involving random variability. This chapter addresses this gap by proposing a methodology that combines simulation with simple heuristics to identify high-quality solutions, particularly in cases where limited resource sharing results in queuing and interactions. The effectiveness of the proposed methodology is demonstrated through a series of numerical experiments, revealing it to be more efficient than the current NEH-based algorithm, which utilizes the state-of-the-art NEH heuristic.

14.2 Problem Modeling

The problem described in this section is inspired by a real industrial scenario at a manufacturing company in Spain. The company operates a production facility that utilizes multiple AGVs to collect and transport products between various designated locations within the plant. At the outset, the products awaiting processing are temporarily stored in a specific warehouse location referred to as the depot. Subsequently, these products are transported to various workstations, each assigned to a specific manufacturing process. After delivering the product, the AGVs returns to the starting point to transport the next product. The number of AGVs available at the starting point is fixed and shared across the different streams of work. Visiting a workstation entails both travel time (to and from) and processing time. It is important to note that each workstation can process only one product at a time. As a result, if an AGVs arrives at an occupied workstation, it must wait in the queue until the workstation becomes available. Figure 14.1 illustrates a schematic representation of the described industrial scenario. The company's primary goal is to determine the optimal sequence for executing the various tasks assigned to the AGVs in order to minimize the total production time, also known as the makespan.

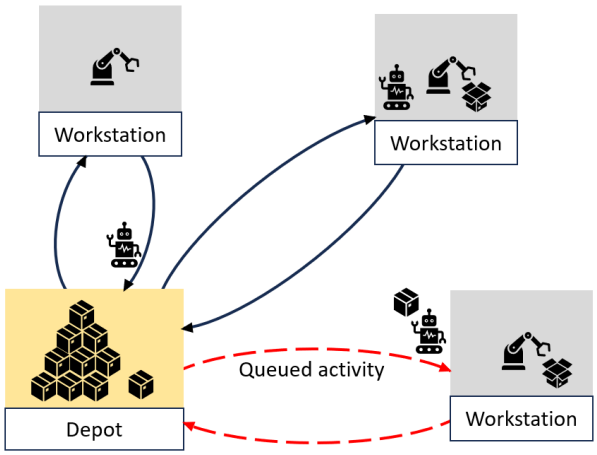


Figure 14.1: Schematic of the elements in the problem.

The allocation of tasks to workstations and AGVs is predetermined by the company, with each task already assigned to a specific location and AGVs. This implies that the products designated for processing, along with their respective travel and processing times, are defined in advance. Additionally, all products to be processed are assumed to be available at the start time. The problem resembles uniform machine scheduling but with a distinctive characteristic: the resources are both limited and shared across the tasks to be executed (i.e., each AGVs can handle only one task at a time). This characteristic gives rise to a variant of the resource-constrained

scheduling problem. Moreover, standard assumptions for such scheduling problems are applied, including the absence of preemption (tasks cannot be interrupted) and treating resources as renewable (each AGVs is available to perform a new task once it completes the current task).

14.2.1 Illustrative Example

Consider a scenario with 5 tasks distributed across 3 distinct workstations, where the assignment of tasks to workstations is predetermined. All three workstations have identical processing times. Specifically, task 1 is assigned to workstation 1, tasks 2 and 4 to workstation 2, and tasks 3 and 5 to workstation 3. In this setup, there are 2 available AGVs. AGVs A is responsible for tasks 1, 2, and 3, while AGVs B manages tasks 4 and 5. An illustration of this scenario is shown on the left side of Figure 14.2.

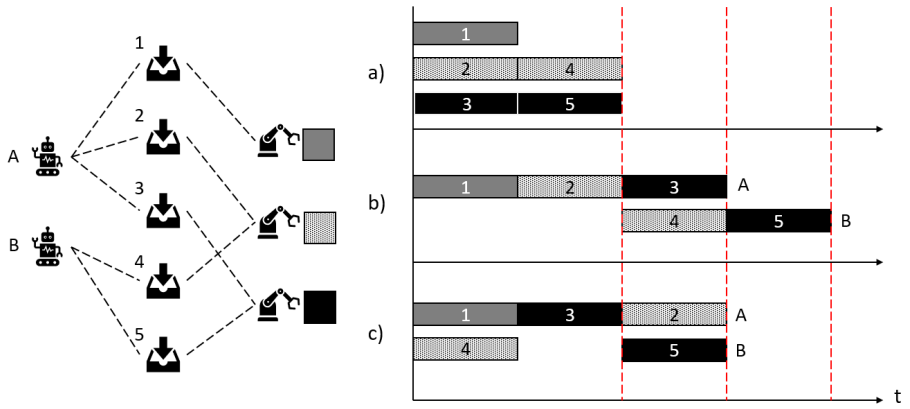


Figure 14.2: Illustrative example: (a) No resource constraints; (b) Active constraints, sequence {1, 2, 3, 4, 5}; (c) Active constraints, sequence {1, 4, 3, 2, 5}.

To simplify the problem, assume that travel time is negligible relative to the processing time at the workstations and is therefore disregarded. In an ideal scenario with unlimited resources and no AGVs constraints, tasks can begin execution as soon as the respective workstations are available. Under these conditions, scheduling becomes trivial, as the task sequence remains fixed and can be managed using a simple FIFO approach. The total makespan is dictated by the maximum number of tasks assigned to a single workstation, assuming all workstations have the same processing time. This scenario is illustrated in timeline a) of Figure 14.2, showing a total makespan equal to 2 times the workstation processing time. When resource sharing constraints are considered, as depicted in timelines b) and c) of Figure 14.2, the order of tasks becomes a key factor in determining the total makespan. In timeline b), if the task sequence {1, 2, 3, 4, 5} is preserved, tasks 1, 2, and 3 are executed consecutively as AGVs A becomes available. Task 4 can be executed simultaneously with task 3, as both AGVs B and the corresponding workstation are available. Task 5 is completed last, once AGVs B becomes available again. This results in a

total makespan of 4 times the workstation processing time, doubling the duration compared to the scenario with infinite resources. The resource constraints lead to task queuing until the assigned resource becomes available. By adhering strictly to the given sequence, opportunities for rearranging tasks to reduce the total makespan are overlooked, highlighting the potential for improvement. In timeline c) of Figure 14.2, the task execution order was adjusted to {1, 4, 3, 2, 5}. In this configuration, tasks 1 and 4 are performed simultaneously, as they are assigned to different AGVs and workstations. Subsequently, task 3 is executed. Although AGVs B becomes available at this point, the corresponding workstation is occupied, causing task 5 to wait. Task 5 is then executed concurrently with task 2. This reordering of tasks improves the makespan, reducing it to just 3 times the workstation processing time.

14.2.2 Generic Formulation for the Sequencing Problem with Shared Resources

In mathematical prespective let $M = \{m_1, m_2, \ldots, m_M\}$ be the set of workstations such that workstation $m_k \in M$ has an associated processing time p_k, which could be deterministic or, in the most generic version of the problem, stochastic. Let R be the set of available resources such that AGV $r_j \in R = \{r_1, r_2, \ldots, r_R\}$. The set of tasks can be defined as the binary relation T such that:

$$T \subset R \times M = \{(r_j, m_k) \mid r_j \in R, m_k \in M\}, \tag{14.1}$$

with task $t_i \in T = \{t_1, t_2, \ldots, t_T\}$. The goal of the RCSP is to produce the totally ordered set $S(T, \leq)$ such that the total makespan C_{max} is minimized. The total makespan can be defined as:

$$C_{max} = max\Big(E(t_i) + p_k\Big), \quad \forall t_i \in S, \tag{14.2}$$

where $E(t_i)$ denotes the start-time of a given task. The total makespan is therefore calculated as the maximum start-time plus its corresponding processing time p_k for all the tasks in S. The activities carried out are constrained by the available resources and workstations. The current activity, $C(S, t) \subset T$, for a given sequence S at time t, is a subset of the set of tasks T that has to comply with both the resource constraint:

$$\sum_{\forall t_i \in C(S,t)} \#r_j \leq 1, \tag{14.3}$$

and with the workstation constraint:

$$\sum_{\forall t_i \in C(S,t)} \#m_k \leq 1, \tag{14.4}$$

meaning that the number of times (represented by the symbol '#') that the resource r_j or the workstation m_k appears in the subset of current activities C is less or equal than 1. In other words, a resource or a workstation can only be used once at any given point in time t.

14.3 Methodological Approach

14.3.1 A two-step method for Sequencing Tasks

To address the challenge of determining the optimal task sequence for AGVs in a manufacturing facility—aimed at minimizing the total production time (makespan) while accounting for resource constraints and task allocation—a two-step approach combining heuristics and simulation was implemented. In the first step, the heuristic component focuses on generating high-quality candidate sequences by efficiently ordering tasks to minimize the total makespan. In the second step, the performance of the proposed sequence is assessed. It is crucial to emphasize that evaluating each sequence involves accounting for the complex and dynamic queuing behavior caused by AGVs idle time when traveling to occupied workstations. As demonstrated in the simplified example presented in Section 14.2.1, even minor adjustments to the task sequence can greatly influence task interferences. This complexity escalates as the number of tasks increases, making it nearly infeasible to address using traditional methods for large instances or in the presence of stochastic variables. Therefore, the proposed approach involves directly modeling the queuing phenomenon within Simul8. This simulation software is well-suited for handling complex queuing scenarios, enabling a comprehensive evaluation of interactions among shared resources. Additionally, Simul8 facilitates the straightforward development of a stochastic version of the problem by incorporating stochastic processing times in place of fixed ones. Figure 14.3 provides a visual representation of the outlined methodology.

The initial phase, depicted in Figure 14.3, involves assigning workstations and AGVs to tasks. This phase is treated as external to the core problem and is assumed to be provided as input. Task-specific details, such as travel and processing times, along with AGVs-to-workstation assignments, are encapsulated within the problem instances. Essentially, the company predefines both task-to-workstation and task-to-AGVs assignments. Subsequently, the task sequencing process begins as the first step of the methodology, employing heuristic algorithms to determine the optimal order of task execution. In the second step, the simulation component evaluates the optimized sequence, accounting for the dynamic queue formations that arise during the simulation as resources are shared among different task streams. Figure 14.4 illustrates the implementation of one of the problem instances in Simul8. It is important to emphasize that the simulation strictly adheres to the optimized sequence and automatically manages the availability

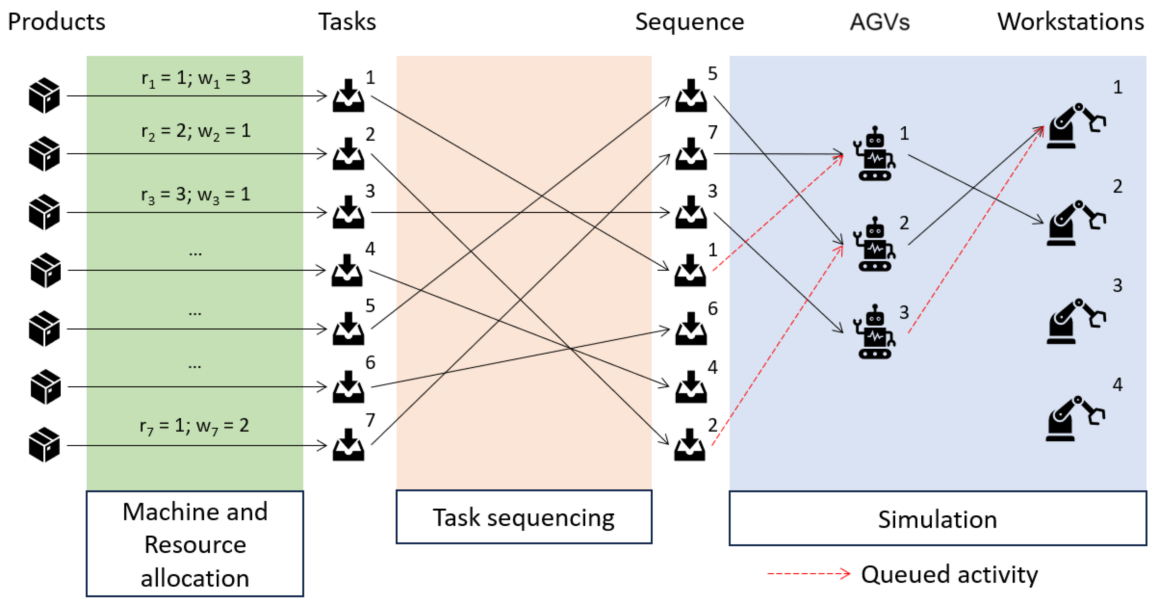


Figure 14.3: Schematic of the proposed methodology with task sequencing heuristic and Simul8 simulation of dynamic queues.

of shared AGVs based on the dynamically generated queues.

14.3.2 Algorithm combination and NEH-based benchmark

To derive efficient task sequences minimizing the total makespan, several well-known sequencing algorithms were combined in this study to generate a number of efficient sequencing heuristics for ordering the given tasks. The heuristic algorithms were devised using the information available within the instances, including the number of workstations, AGVs, tasks, travelling and processing times for each task, and task-specific workstation and AGVs allocations. The partial sequencing algorithms considered in this study were the following:

- Most Loaded Resource First (MLRF): This algorithm sorts AGVs, prioritizing the most loaded AGVs first based on either the total processing time the AGVs is engaged or the number of times it is required by a task. This study used total processing time for sorting.

- Less Loaded Resource First (LLRF): Similar to MLRF, LLRF sorts AGVs from lower workload to higher workload.

- Shortest Processing Time First (SPTF): This algorithm prioritizes tasks with the shortest processing times, sorting workstations from the least to the greatest processing time for tasks.

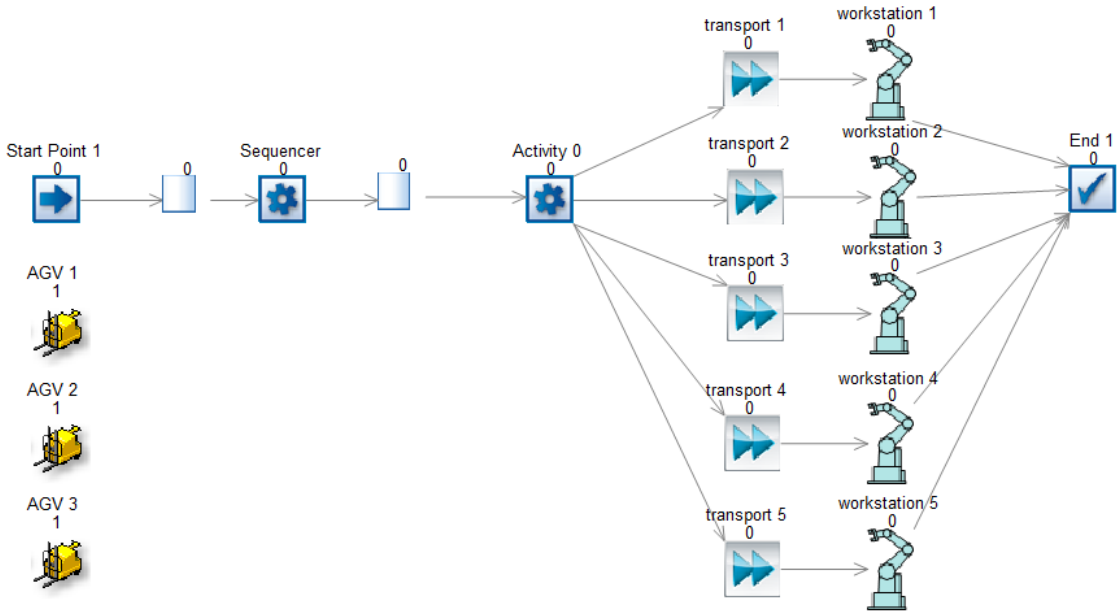


Figure 14.4: Screenshot of the Simul8 model: Instances with 3 AGVs and 5 workstations, managing queues and sequences for both automatically.

- Longest Processing Time First (LPTF): Contrary to SPTF, LPTF sorts workstations in reverse order, from lower to higher processing times.

These algorithms are referred to as partial sequencing algorithms because they arrange resources (AGVs or workstations) rather than tasks, which represent the specific combination of an AGVs and a workstation, along with their associated travel and processing times. These algorithms can be regarded as individual components that contribute to a comprehensive heuristic procedure designed to generate candidate solutions for the task sequencing problem while accounting for shared resources. The heuristic procedure begins by organizing the AGVs and workstations. This process utilizes the partial algorithms mentioned earlier, including those for sorting AGVs (MLRF and LLRF) and workstations (SPTF and LPTF). Combining these sorting algorithms results in four distinct heuristics: (i) MLRF/LPTF, (ii) MLRF/SPTF, (iii) LLRF/SPTF, and (iv) LLRF/LPTF.

Once the AGVs and workstations are organized, tasks are assigned to the final sequence using a round-robin method. This involves starting with the highest-ranked workstation from the sorted list and pairing it with the top-ranked AGVs from its respective list. If the pairing does not correspond to a task to be executed, the next AGVs in the list is considered. This process repeats until all tasks are included in the final candidate sequence. Algorithm 8 illustrates the MLRF/LPTF variant of the heuristic. To generate other versions of the heuristic, modifications are necessary in lines 7 to 8, while the remainder of the procedure remains unchanged. A

graphical representation of the heuristic process is provided in Figure 14.5.

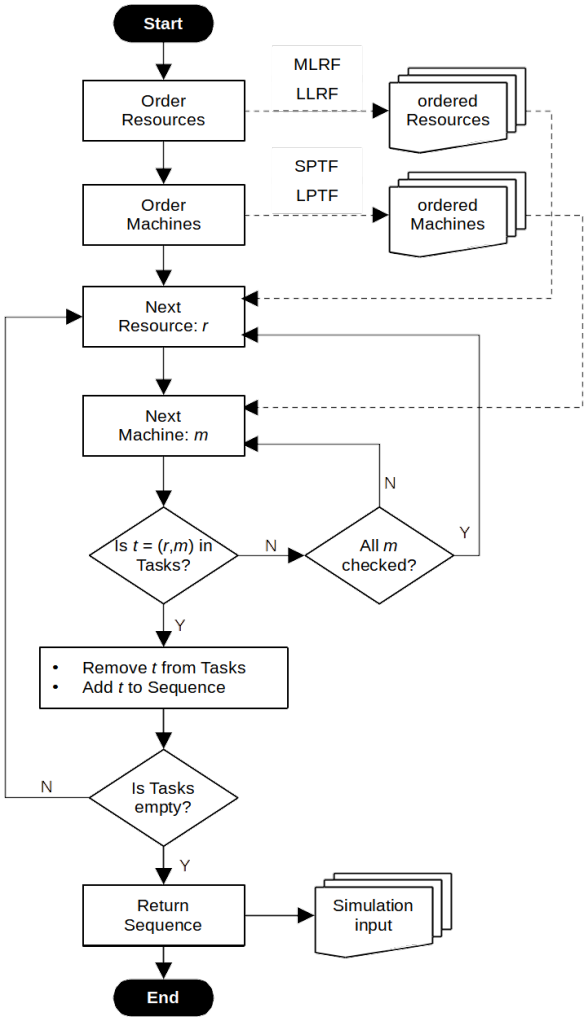


Figure 14.5: Schema of the heuristic for generating candidate solutions (task sequences as simulation inputs).

To evaluate the performance of our proposed solving methodology, we introduce a benchmark algorithm that employs the state-of-the-art NEH heuristic as a reference baseline for comparison (Nawaz et al., 1983). Algorithm 9 presents the main steps of the NEH-based algorithm. The scheduling process begins by dividing the list of tasks into multiple sub-problems, each based on the corresponding workstation-AGVs assignment. In other words, tasks assigned to the same workstation and AGVs are grouped together. Each sub-problem is independently solved using the well-established NEH heuristic, disregarding resource-sharing constraints. Once sub-sequences are generated for each sub-problem, the final task sequence is constructed by merging these sub-sequences. The sub-sequences are first arranged in descending order based on the number of tasks they contain, and tasks are sequentially selected from each sub-sequence and added to the final sequence until all tasks are incorporated.

Algorithm 8 MLRF/LPTF algorithm.

Input: Tasks = $\{t_i : i \leq T\}$, Resources = $\{r_j : j \leq R\}$, Machines = $\{m_k : k \leq M\}$
Output: Sequence
for each $t_i \in$ Tasks **do**
    $(r_j, m_k) \leftarrow t_i$
    r_j.load $\leftarrow \sum(m_k$.processTime) // Calculate total workload per resource
end for
Resources $\leftarrow$ `orderBy`(r_j.load, desc) // From max. to min.
Machines $\leftarrow$ `orderBy`(m_k.processTime, desc) // From max. to min.
$m \leftarrow$ `first`(Machines)
while Tasks $\neq \emptyset$ **do**
    $r \leftarrow$ `next`(Resources) // Using Round-Robin
    $t \leftarrow (r, m)$
    if $t \in$ Tasks **then**
        Sequence $\leftarrow$ `add`(t)
        Tasks $\leftarrow$ `remove`(t)
        $m \leftarrow$ `next`(Machines)
    else
        for each $m \in$ Machines **do**
            $m \leftarrow$ `next`(Machines) // Using Round-Robin
            $t \leftarrow (r, m)$
            if $t \in$ Tasks **then**
                Sequence $\leftarrow$ `add`(t)
                Tasks $\leftarrow$ `remove`(t)
            end if
        end for
    end if
end while
return Sequence

14.4 Computational Experiments

The computational experiments were performed on a Windows 10 operating system with an i7-4500U CPU at 2.40GHz and 6 GB of RAM. All algorithms were implemented using Python v3.8.10, and the simulation model was developed using Simul8 2022 (version 29.0). To assess the effectiveness of the heuristic algorithms designed for AGVs task scheduling, multiple problem instances were generated. Table 14.1 summarizes all instances utilized in this study along with their key attributes. Each instance is characterized by the number of workstations, the number of AGVs, the total tasks to be completed, the travel and processing times associated with each task, and the assignment of tasks to specific workstations and AGVs. Hereafter, both the AGVs travel time and the workstation processing time will be collectively referred to as "processing time" for simplicity.

In fact, two distinct types of instances were considered for each workstations-AGVs-tasks combination based on processing times. The first type is characterized by processing times generated from a uniform random distribution tightly centered around an average value, leading to instances with Low Variability in Processing Times (LVPT). Conversely, the second type

Algorithm 9 NEH-based algorithm.

```
1: Input: Tasks = {t_i : i ≤ T}, Resources = {r_j : j ≤ R}, Machines = {m_k : k ≤ M}
2: Output: Sequence
3: Combinations ← ∅
4: for t ∈ Tasks do
5:     Combinations(t.machine, t.resource) ← add(t)          // Split tasks into combinations.
6: end for
7: SubSequences ← ∅
8: for c ∈ Combinations do
9:     SortedTasks ← orderBy(c.totalTimes, desc)
10:    s ← HeuristicNEH(SortedTasks)
11:    SubSequences ← add(s)                                 // Solve Combinations using NEH.
12: end for
13: SortedSequences ← orderBy(SubSequences.length, desc)
14: while SortedSequences ≠ ∅ do
15:    for s ∈ SortedSequences do
16:       if s ≠ ∅ then
17:          t ← removeFirst(s)
18:          Sequence ← add(t)
19:       end if
20:    end for
21: end while
22:
23: return Sequence
```

features a wider range between the minimum and maximum processing times, resulting in instances with High Variability in Processing Times (HVPT). Similarly, two instance types were generated for assigning tasks to workstation-AGVs combinations. The first type ensures an evenly distributed task assignment, resulting in Low Variability in Task Assignment (LVTA) across all workstation-AGVs combinations. On the other hand, the second type creates an uneven distribution, where some workstation-AGVs combinations are assigned a disproportionately higher workload, leading to High Variability in Task Assignment (HVTA). Combining these characteristics yields four distinct instance types for evaluating the performance of the algorithms: (i) low variability in processing times and low variability in task assignment (LVPT/LVTA); (ii) high variability in processing times and low variability in task assignment (HVPT/LVTA); (iii) low variability in processing times and high variability in task assignment (LVPT/HVTA); and (iv) high variability in both processing times and task assignment (HVPT/HVTA).

Table 14.1: Instances used in computational experiments.

Instance	Workstations	AGVs	Tasks	Type
$m5r3t100\_01$	5	3	100	LVPT/LVTA
$m5r3t100\_02$	5	3	100	HVPT/LVTA
$m5r3t100\_03$	5	3	100	LVPT/HVTA
$m5r3t100\_04$	5	3	100	HVPT/HVTA
$m5r3t200\_01$	5	3	200	LVPT/LVTA
$m5r3t200\_02$	5	3	200	HVPT/LVTA
$m5r3t200\_03$	5	3	200	LVPT/HVTA
$m5r3t200\_04$	5	3	200	HVPT/HVTA
$m10r5t300\_01$	10	5	300	LVPT/LVTA
$m10r5t300\_02$	10	5	300	HVPT/LVTA
$m10r5t300\_03$	10	5	300	LVPT/HVTA
$m10r5t300\_04$	10	5	300	HVPT/HVTA
$m10r5t600\_01$	10	5	600	LVPT/LVTA
$m10r5t600\_02$	10	5	600	HVPT/LVTA
$m10r5t600\_03$	10	5	600	LVPT/HVTA
$m10r5t600\_04$	10	5	600	HVPT/HVTA
$m20r10t1200\_01$	20	10	1200	LVPT/LVTA
$m20r10t1200\_02$	20	10	1200	HVPT/LVTA
$m20r10t1200\_03$	20	10	1200	LVPT/HVTA
$m20r10t1200\_04$	20	10	1200	HVPT/HVTA

14.5 Analysis of Results

Table 14.2 presents the simulation results for each instance and the heuristic algorithms evaluated. The first column specifies the instances, while the subsequent columns display the total makespan required to complete all tasks in each instance. First, the results obtained from the NEH-based approach are presented, serving as the reference baseline for comparison. Subsequently, the remaining columns in the table showcase the results for the different combinations of sequencing algorithms. Additionally, the average makespan is calculated for each set of instances. Finally, the percentage gaps relative to the NEH-based approach are provided, as calculated by the following formula:

$$Gap[\%] = \frac{C^{i}_{max} - C^{NEH}_{max}}{C^{NEH}_{max}} \times 100, \tag{14.5}$$

where C_{max} is the total makespan for each one of the $i = \{$MLRF/LPTF, MLRF/SPTF, LLRF/SPTF, LLRF/LPTF$\}$ algorithms considered.

At this stage, it is important to highlight scenarios where resource sharing leads to inefficiencies in the task sequence. These inefficiencies arise when a task is ready to begin processing but

Table 14.2: Computational experiment results: Total makespan and percentage gap relative to the NEH strategy.

Instance	NEH	MLRF/LPTF	Gap [%]	MLRF/SPTF	Gap [%]	LLRF/SPTF	Gap [%]	LLRF/LPTF	Gap [%]
$m5r3t100\_01$	6594	5248	-20.41%	5377	-18.46%	5480	-16.89%	5310	-19.47%
$m5r3t100\_02$	6838	5674	-17.02%	6061	-11.36%	6414	-6.20%	5536	-19.04%
$m5r3t100\_03$	8644	8125	-6.00%	8115	-6.12%	8568	-0.88%	8277	-4.25%
$m5r3t100\_04$	8688	9121	4.98%	9115	4.91%	9333	7.42%	9043	4.09%
Average:	7691	7042	-9.61%	7167	-7.76%	7448.75	-4.14%	7041.50	-9.67%
$m5r3t200\_01$	13818	9752	-29.43%	10090	-26.98%	10405	-24.70%	9946	-28.02%
$m5r3t200\_02$	14761	11016	-25.37%	11509	-22.03%	11895	-19.42%	11350	-23.11%
$m5r3t200\_03$	16896	14528	-14.02%	14623	-13.45%	15048	-10.94%	14921	-11.69%
$m5r3t200\_04$	17236	15986	-7.25%	16410	-4.79%	16607	-3.65%	16402	-4.84%
Average:	15677.75	12820.50	-19.02%	13158	-16.81%	13488.75	-14.68%	13154.75	-16.91%
$m10r5t300\_01$	14966	11249	-24.84%	10613	-29.09%	11866	-20.71%	10918	-27.05%
$m10r5t300\_02$	14719	11393	-22.60%	11786	-19.93%	12243	-16.82%	12226	-16.94%
$m10r5t300\_03$	24557	22198	-9.61%	22438	-8.63%	22589	-8.01%	22453	-8.57%
$m10r5t300\_04$	23817	21690	-8.93%	21828	-8.35%	22546	-5.34%	21905	-8.03%
Average:	19514.75	16632.50	-16.49%	16666.25	-16.50%	17311	-12.72%	16875.50	-15.15%
$m10r5t600\_01$	36203	23095	-36.21%	22665	-37.39%	22094	-38.97%	24633	-31.96%
$m10r5t600\_02$	36759	26424	-28.12%	25766	-29.91%	25414	-30.86%	26826	-27.02%
$m10r5t600\_03$	53791	47427	-11.83%	47565	-11.57%	47684	-11.35%	47628	-11.46%
$m10r5t600\_04$	54801	49184	-10.25%	48980	-10.62%	49037	-10.52%	49149	-10.31%
Average:	45388.50	36532.50	-21.60%	36244	-22.37%	36057.25	-22.93%	37059	-20.19%
$m20r10t1200\_01$	52242	31338	-40.01%	31198	-40.28%	31503	-39.70%	32768	-37.28%
$m20r10t1200\_02$	54533	38646	-29.13%	38945	-28.58%	37442	-31.34%	37821	-30.65%
$m20r10t1200\_03$	75115	45027	-40.06%	45039	-40.04%	47707	-36.49%	47524	-36.73%
$m20r10t1200\_04$	78231	54280	-30.62%	55022	-29.67%	55259	-29.36%	54306	-30.58%
Average:	65030.25	42322.75	-34.95%	42551	-34.64%	42977.75	-34.22%	43104.75	-33.81%

is delayed due to one of the following reasons: (a) an AGVs is unavailable for transportation, or (b) the workstation it is assigned to is already occupied by a previous task. Consider a scenario where resources are either not shared among different AGVs-workstation pairs or are sufficiently abundant to accommodate sharing between workstations. In the context of this study, such situations are classified as inefficiency-free scenarios. In such cases, the NEH-based approach is expected to perform exceptionally well, minimizing the total makespan across various instance sizes and types. However, for the instances analyzed, the NEH-based approach results in the highest total makespan in almost all cases, as it lacks a specialized strategy for optimizing the overall makespan under resource-sharing constraints. It can be observed that all the approaches proposed in this study outperform the NEH-based approach, as evidenced by the lower makespan values and negative percentage gaps. Notably, the more negative the gap, the greater the improvement in makespan relative to the NEH-based strategy. This improvement is attributed to the optimized sequence, which reduces potential inefficiencies and, consequently, lowers the overall makespan. Beyond the expected increase in makespan as the instance size grows (since a higher number of tasks requires more time for completion), the performance of the algorithms tends to improve with larger instances. In scenarios with a greater number of AGVs, workstations, and tasks, the algorithm consistently produces better solutions compared to the baseline. This trend can be attributed to the limited room for improvement in smaller

instances, where changes in task sequencing have a minimal impact due to the reduced scope for optimization.

With respect to the different types of instances, it is worth noting that those exhibiting (HVTA) lead to an uneven utilization of system resources (AGVs and workstations), where certain resources are assigned significantly more tasks than others. As a result, fewer interactions occur between AGVs and workstations within the system, allowing the NEH-based algorithm to achieve solutions with acceptable performance. Conversely, instances with LVTA tend to generate a higher number of interactions between shared resources, where the proposed algorithm demonstrates superior performance, as reflected in the comparatively larger percentage gaps. An extreme example of this scenario is observed in the results for instance $m5r3t100\_04$, which exhibits high variability in both processing times and task assignment. Interestingly, the NEH-based approach outperforms the proposed algorithms in this case, achieving percentage gaps ranging from 4.09% to 7.42%. A detailed analysis of the instance within the simulation revealed that the majority of tasks were allocated to a specific workstation-AGVs combination. As a result, the number of interactions caused by resource sharing was significantly low. Consequently, the NEH heuristic was able to generate a high-quality sequence for that specific combination, effectively minimizing the makespan and ultimately producing a better task sequence than the proposed methods.

The identified trend suggests that the suggested algorithms demonstrate reduced effectiveness in situations characterized by significant variability in task allocation, where resources are distributed unevenly rather than being shared uniformly. Interestingly, this occurrence is not present in the majority of cases. This can be explained by the unequal increase in the quantity of tasks relative to the number of AGVs and workstations.

As the number of tasks increases, the variability in task assignment diminishes, resulting in a more balanced distribution of tasks across resources. As a result, interactions between resources tend to occur irrespective of the variability in task assignment. These findings highlight the promising scalability of the proposed methodology, particularly when managing a large number of tasks in environments with shared resources. Furthermore, the algorithm's performance appears to decline in instances characterized by HVPT. This can be attributed to the inefficiencies caused by exceptionally long processing times at certain workstations, which result in unavoidable queues and limit the effectiveness of sequence adjustments.

The boxplot shown in Figure 14.6 presents the average percentage gaps for the MLRF/LPTF, MLRF/SPTF, LLRF/SPTF, and LLRF/LPTF strategies relative to the NEH-based approach. The NEH-based approach was chosen as the reference baseline for evaluating other strategies, as it represents the state-of-the-art, particularly in scenarios without resource-sharing interactions. Therefore, the NEH-based heuristic serves as a valuable benchmark for assessing the relative

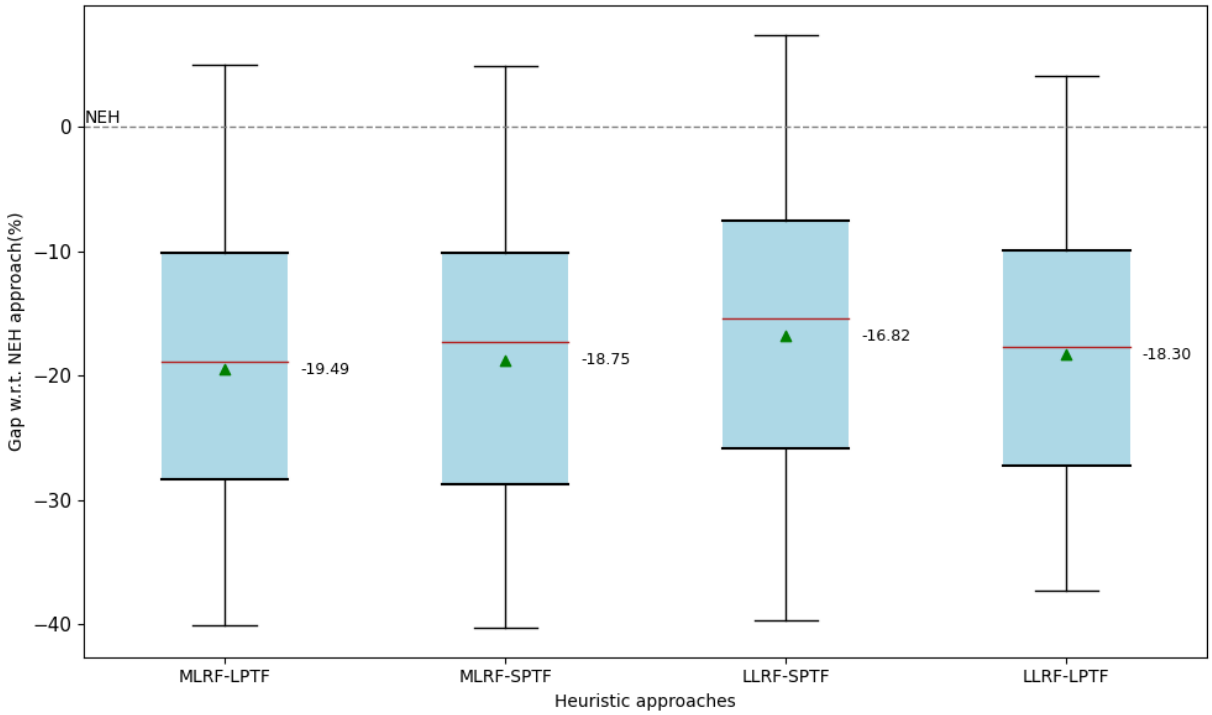


Figure 14.6: Boxplot of heuristic gaps relative to the NEH approach.

performance of alternative strategies. It is noteworthy that the average percentage gap consistently remains below zero, signifying an improvement in the average makespan compared to the NEH-based approach across all instances. The average percentage gaps range between -16.82% and -19.49%. The significant gap relative to the NEH-based approach suggests that the observed improvement may stem more from the round-robin sequencing method than from the specific combination of sorting algorithms for AGVs and workstations. This indicates that, in scenarios where resource sharing is a critical factor, balancing resource utilization takes precedence over optimizing the sequencing within a single stream of work. In the scenario analyzed, the combination of simple sequencing strategies outperforms the NEH algorithm when it is applied without adaptation in a context where resources are limited and must be shared across multiple streams of work.

Delving deeper, the strategies that prioritize the most loaded resource (MLRF/LPTF and MLRF/SPTF) appear to deliver results that are as good as or even better than those achieved by the other two approaches. This can be attributed to the structure of the benchmark instances, where the number of AGVs is consistently lower than the number of workstations. Therefore, the most critical resource to consider when generating an optimized task sequence is the processing time of the AGVs, with priority given to the most heavily loaded ones. Additionally, strategies that prioritize workstations with longer processing times (LPTF) also tend to perform effectively,

as they help prevent excessive workstation occupancy during the later stages of task processing. As a result, sequences prioritizing the LPTF strategy over SPTF consistently exhibit lower percentage gaps, highlighting its effectiveness across nearly all instances. The MLRF/LPTF algorithm emerges as the most effective sequencing heuristic for tackling the proposed problem. As previously discussed, its superiority primarily lies in its ability to strategically position the most heavily utilized AGVs and the most loaded workstations at the beginning of the sequence, thereby minimizing potential queue formations and inefficiencies later in the process.

Chapter 15

Discrete-Event Simheuristic for Storage Allocation Problem

As manufacturing and logistics systems become more complex, the combination of optimization and simulation emerges as a powerful tool for supporting informed managerial decision-making. A well-designed storage location assignment policy is critical for improving warehouse operations, which are central to the overall efficiency of supply chains. Existing traditional approaches to solving the SLAP often have limitations, such as overlooking the stochastic variability of processes or the interactions between various warehouse activities. This chapter[1] tackles these limitations by introducing a DES framework that delivers robust solutions, accounting for the dynamic nature of real-world warehouse environments. The approach incorporates the complexity of the problem by integrating both the order sequence and picking route into the solution process, and leverages commercial simulation software to minimize the influence of stochastic events on the solution's effectiveness.

[1]The contents of this part is based on the following work:

- Leon, Jonas F, Yuda Li, **Mohammad Peyman**, Laura Calvet, and Angel A Juan (2023). *"A discrete-event simheuristic for solving a realistic storage location assignment problem"*. In: Mathematics 11.7, p. 1577.

15.1 Problem Description

Warehouse managers often face challenging decisions, such as how to organize products, which items to reposition for better productivity, or whether resources are sufficient to meet upcoming orders. These decisions are complex due to the vagueness of terms like "best way", the need to consider various parameters, and the interconnection between different warehouse activities, such as product placement and worker movement. However, with well-defined questions and the right tools, science and technology can guide these decisions. As noted by Zhang, Wang, et al. (2019), warehouse operations can be improved through strategies like assigning items to optimal storage locations, determining efficient retrieval routes, and batching orders. While these elements aim to enhance overall warehouse efficiency, they are often studied separately in the literature. Breaking down complex problems is a common analytical approach, but isolating aspects like SLAP can impact other operations, such as picking routes, potentially reducing overall efficiency. This highlights a gap between research methods and real-world needs. This chapter aims to bridge that gap by developing a methodology that combines heuristics and simulation (simheuristics) to optimize product storage location assignments, while integrating other aspects of warehouse operations, such as routing and order fulfillment. Simulation offers several advantages: it allows the evaluation of complex, interconnected scenarios, generates data for stochastic analysis, and integrates multiple optimization problems into a single framework. However, it also has drawbacks, including increased computational time and the challenge of reproducibility due to the proprietary nature of many simulation tools. There are three original contributions in this chapter. The first aspect is the modeling of a realistic version of the SLAP (Reyes et al., 2019), where orders serve as the driving factor for optimization. As highlighted in the literature review, a common approach is to treat the SLAP in isolation and model it using mixed integer linear programming. However, this approach has limitations when applied to real-world warehouse conditions, such as the inclusion of stochastic variables or the consideration of interactions with other warehouse activities (e.g., retrieval routing strategies, order dispatching, etc.). In this context, a generic object-oriented formulation is proposed to accommodate stochastic conditions and integrate the SLAP within a broader warehouse optimization framework. The second contribution is the development of an integrated heuristic-simulation methodology, or simheuristic (Chica, Juan, Bayliss, et al., 2020), designed to solve the SLAP using the proposed model. This methodology relies heavily on the Python programming language (Van Rossum et al., 2009) and the well-established commercial simulation tool FlexSim (Nordgren, 2003). The third contribution is a thorough set of computational experiments, which includes a sensitivity analysis to assess the effectiveness of the approach adopted in this study, applied to a set of benchmark problem instances.

The related work discussed in chapter 3 focuses on the SLAP in isolation, which may not be the most effective strategy for optimizing the operations of a real-world warehouse, as the routing and order picking problems are closely interconnected with it (Amorim-Lopes et al., 2021). The formulation outlined in this section integrates a flexible routing and order sequence definition, while addressing a deterministic version of the SLAP. A fully deterministic solution to the SLAP can provide both fast and high-quality results. However, for these solutions to remain high-quality in real-life environments, they must assume the absence of stochastic variability in the problem conditions. In other words, they rely on significant simplifications that may not always align with the complexities of real-world scenarios. Finding solutions that remain robust under real-world warehouse conditions is crucial for decision-makers. To address this, a simheuristic framework has been developed to ensure that the SLAP solution obtained performs effectively within a realistic simulation context. Thus, the approach presented in this section aims to: *(i)* adopt an integrated approach that captures the complexity of the problem; and *(ii)* utilize simulation to minimize the impact of stochastic events on the quality of the SLAP solution in a real-world application. As far as the authors are aware, this is the first time a DES model developed using FlexSim has been seamlessly integrated into a simheuristic algorithm for optimizing the SLAP in a realistic setting.

15.2 Problem Modeling

Let the warehouse W in question contain M storage spaces designated for product storage. Typically, warehouse locations are designated with references to terms such as “racks”, “aisles”, “slots”, “positions”. For example, in a straightforward grid-patterned warehouse, locations can be pinpointed by utilizing rows (R) and columns (C). In general, there may be varying heights (H) and various subdivisions, which necessitates the use of a more comprehensive definition for storage location M represented as ($M = R \times C \times H \times ...$). Each position in the warehouse can be denoted as l_i, where: $l_i \in \mathrm{W} = \{l_1, l_2, \ldots, l_M\}$. The warehouse contains a variety of N different product types, denoted as $p_k \in \mathrm{P} = \{p_1, p_2, \ldots, p_N\}$, where N is less than or equal to M (indicating that the number of locations exceeds the number of product types). Consequently, a storage location assignment can be represented by the binary relation $\mathcal{R}$ defined as follows:

$$\mathcal{R} \subset \mathrm{W} \times \mathrm{P} = \{(l_i, p_k) \mid l_i \in \mathrm{W}, p_k \in \mathrm{P}\} \tag{15.1}$$

Consequently, the relation $\mathcal{R}$, which consists of ordered pairs in the form of location-product (l_i, p_k), is defined as a subset of the Cartesian product W $\times$ P, where W and P represent two distinct sets. With this definition, addressing the SLAP in its broadest sense involves identifying the binary relation $\mathcal{R}$ that meets a specific condition or a collection of conditions. It is worth

mentioning that further limitations may be applied to both the sets and the binary relation. Depending on the specific issue being addressed, the relationship may be classified as injective, surjective, or bijective. For example, it may be necessary for the relationship to encompass the entire domain of the definition of W (meaning every location should either have an assigned product or be vacant $\emptyset$), and there may still be some products that remain unassigned. In the examined scenario, the customer orders O are significant as the optimization process considers the expenses associated with retrieving these orders in a specific order. Each order o_m consists of a collection of products p_k, with their respective locations l_i defined by the relation $\mathcal{R}$. Throughout the analysis period, it is essential to assess all orders $o_m \in \mathrm{O} = \{o_1, o_2, \ldots, o_Z\}$ based on a cost function $\phi(\mathcal{R})$. This function ϕ may represent any metric or a combination of metrics considered significant by management. This could range from the usual distance traveled or time spent by warehouse staff to the energy usage of the AGVs, or even the incentives linked to visiting specific sites. It is essential to establish the sequence strategy, denoted as S, for navigating the various locations l_i necessary to fulfill an order o_m. After determining this strategy, a specific sequence $\mathrm{S}(o_m)$ can be generated for every order. It is crucial to develop the sequence strategy, referred to as S, for traversing the different locations l_i required to complete an order o_m. Once this strategy is defined, a particular sequence $\mathrm{S}(o_m)$ can be created for each order. These stochastic variables may influence various facets of the issue, but they are usually incorporated into the cost function, such as the velocity of the retrieval system and the likelihood of a product being out of stock. Thus, the SLAP can be articulated as determining $\mathcal{R}$ in a way that minimizes the total expected cost across all orders, while accounting for stochastic variability t and based on a predetermined strategy S.

$$opt\left(\sum_{m=1}^{Z} \phi(\mathcal{R}, t) \mid_{\mathrm{S}(o_m)}\right) \tag{15.2}$$

Figure 15.1 presents a summary of the formulation and illustrates the binary relation between locations and products, denoted as $\mathcal{R}$, which corresponds to physical locations within the warehouse alongside their respective products. Each order o_m specifies the products designated for collection, with the corresponding sequence required being represented as $\mathrm{S}(o_m)$. This establishes a stochastic operational cost d_{lp}^t for every product retrieval action, which adds to the cost function ϕ. As demonstrated in Figure 15.1, this relationship is linear. Ultimately, the overall cost, summed across all orders, serves as the objective function that needs optimization, as outlined in Equation 15.2.

This generic formulation, in its present form, is not useful for constructing directly a linear or integer programming model, which is a common strategy for solving the SLAP problem using exact methods. Instead, it helps to develop the structure of an object-oriented programming

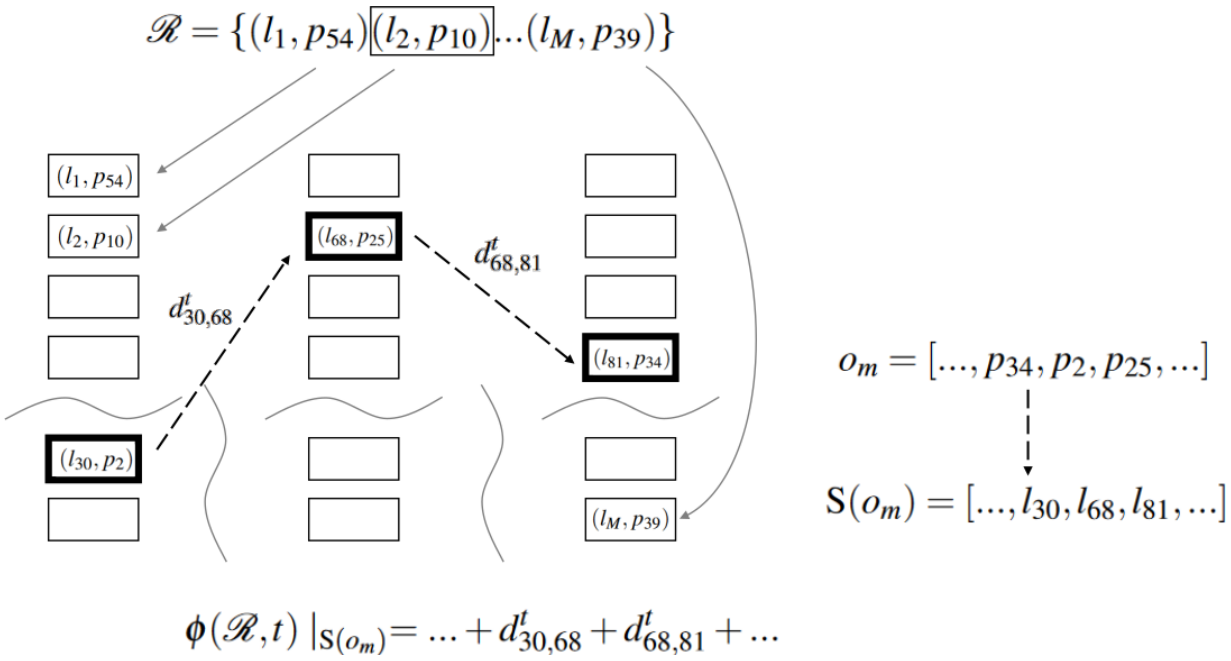


Figure 15.1: Components of the SLAP with an illustrative example instance.

code for studying the SLAP in a way that is compatible with a simulation model. There are some advantages in following this modeling approach which might gain traction in the future, as increased computation power becomes even more accessible. The necessary mathematical rigor is present in the definition of the model, in order to remove ambiguity and outline the hypothesis, but there is no need to overcomplicate the model numerically. The simplicity helps in at least two fronts: *(i)* the greater interpretability of the model, which is meant to aid managerial decision-making, and therefore, can be better explained in a natural and transparent manner; and *(ii)* the greater flexibility when it comes to the practical implementation and adaptation of the model, specially if it is intended to be connected to different dedicated commercial simulation tools available in the market, or employed in real use cases.

15.2.1 Additional Assumptions for the Considered SLAP Study

After establishing the generic static SLAP with orders, this subsection aims to elaborate on the specific problem at hand and clarify its underlying assumptions. Specifically, it addresses aspects such as the type of warehouse, the nature of the products, the orders involved, and the potential solutions that will be examined. The SLAP discussed in this section focuses on the arrangement of items within the designated "picking slots". Picking slots are specific locations allocated for the storage of products in unit or case formats, enabling employees to gather ("pick") the precise quantity of ordered items linked to each product. These picking slots are expected to be located either on the ground level or the first level, while the remaining height of

the rack is reserved for storage that does not involve picking. This indicates that the issue can be represented in two dimensions (2D). Consequently, only the 2D features of the warehouse were taken into account, specifically the width and depth of the aisles as well as measurements in the horizontal plane. The modeled warehouse pertains to a manual system, meaning it is not automated, and utilizes a voice-assisted single-order picking process. This indicates that employees navigate the warehouse with their activities directed by a warehouse management system, which instructs them on the subsequent products to gather. While there are more complex scenarios and picking strategies available, the choice for this particular warehouse design was influenced by its widespread prevalence globally. It is considered essential to address the simpler model (single-order picker-to-part) prior to advancing to more intricate picking methods. Figure 15.2 illustrates a simplified representation of the warehouse abstraction that informed our modeling of the warehouse.

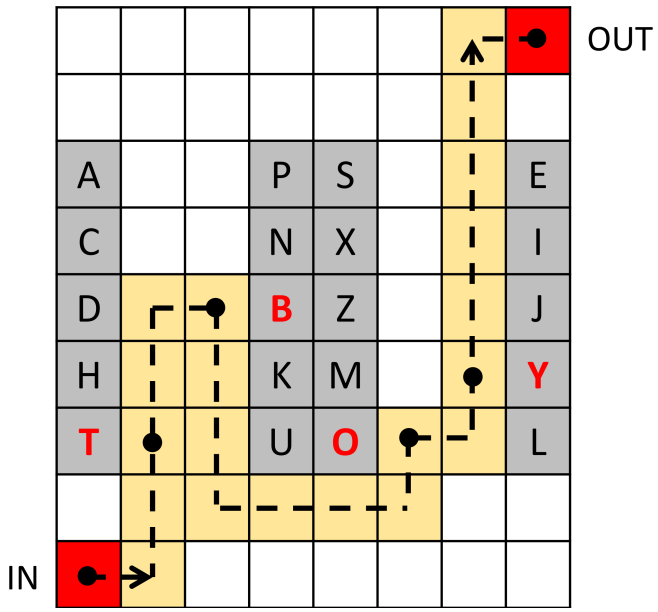


Figure 15.2: 2D schematic of SLAP components: space discretization, IN/OUT points, product-slot assignments, order products (red letters), and the picking route (IN, T, B, O, Y, OUT).

The locations designated for inbound and outbound activities (where products are received and orders are shipped) are considered permanent fixtures within the warehouse and will not vary over the course of the planning period. In this context, their position is considered a parametric decision that will impact the evaluation of the solution. Modifying the input and output points creates a different solution space for the SLAP, as altering the locations where completed orders are placed or where workers begin picking the next order affects the solution evaluation metrics (e.g., traveled distance). The impact of the inbound and outbound positions on the SLAP solution is often overlooked in most theoretical studies and is treated as a fixed

input in case studies within the literature. In contrast, this study considers the sensitivity of the solution to the positioning of the input and output points for the deterministic part of the proposed simheuristic. In this study, we assume that each type of product is assigned to a specific storage location, and each storage location is dedicated to a single product type. Additionally, the physical capacity of each storage location is ample enough to hold any quantity of items of the same type. Therefore, the replenishment of picking locations is not considered (i.e., there is always an adequate supply of products available for picking). Also, it is assumed that there is sufficient capacity for the picker to collect all products from an order in a single route. Finally, all products are assumed to have identical physical storage requirements (e.g., light, temperature), thereby eliminating the need for additional constraints, such as dedicated zones for specific products. Within the context of this study, orders are characterized as sequential arrangements of items to be collected from the warehouse, depicted in list format. Each list encompasses a designated quantity of product requests originating from a single client (i.e., one order for each client). It is presumed that these orders adhere to the Pareto principle, suggesting that approximately 80% of customer orders consist of merely 20% of the distinct products (warehouse references).

Given that daily product demand from customers fluctuates, the orders placed at the warehouse are considered the primary source of variability. No commercial agreements were assumed, and fixed product quantities based on a set schedule were not hypothesized.

15.3 Methodological Approach

Simheuristics belong to the category of simulation-optimization techniques and can be utilized in a diverse range of disciplines. They facilitate varying degrees of engagement between the simulation and optimization elements. In this research, the interaction between these two components was implemented through a commonly employed inter-process communication protocol known as "sockets". Figure 15.3 illustrates the overall architecture of the communication between Python and FlexSim. This connection enables both synchronous and adaptable interactions between the two software applications.

The simheuristic approach begins with addressing the deterministic variant of the problem through a metaheuristic technique coded in Python. Given that the warehouse configuration is established and the range of products it can hold is defined, the sole additional requirement is the collection of orders to be fulfilled. These orders may consist of actual requests entered into the Python program or can be synthesized based on projected demand for the time frame being examined. The metaheuristic will generate candidate solutions, which are then transmitted to FlexSim via the socket connection. The simulation model is set up in exactly the same

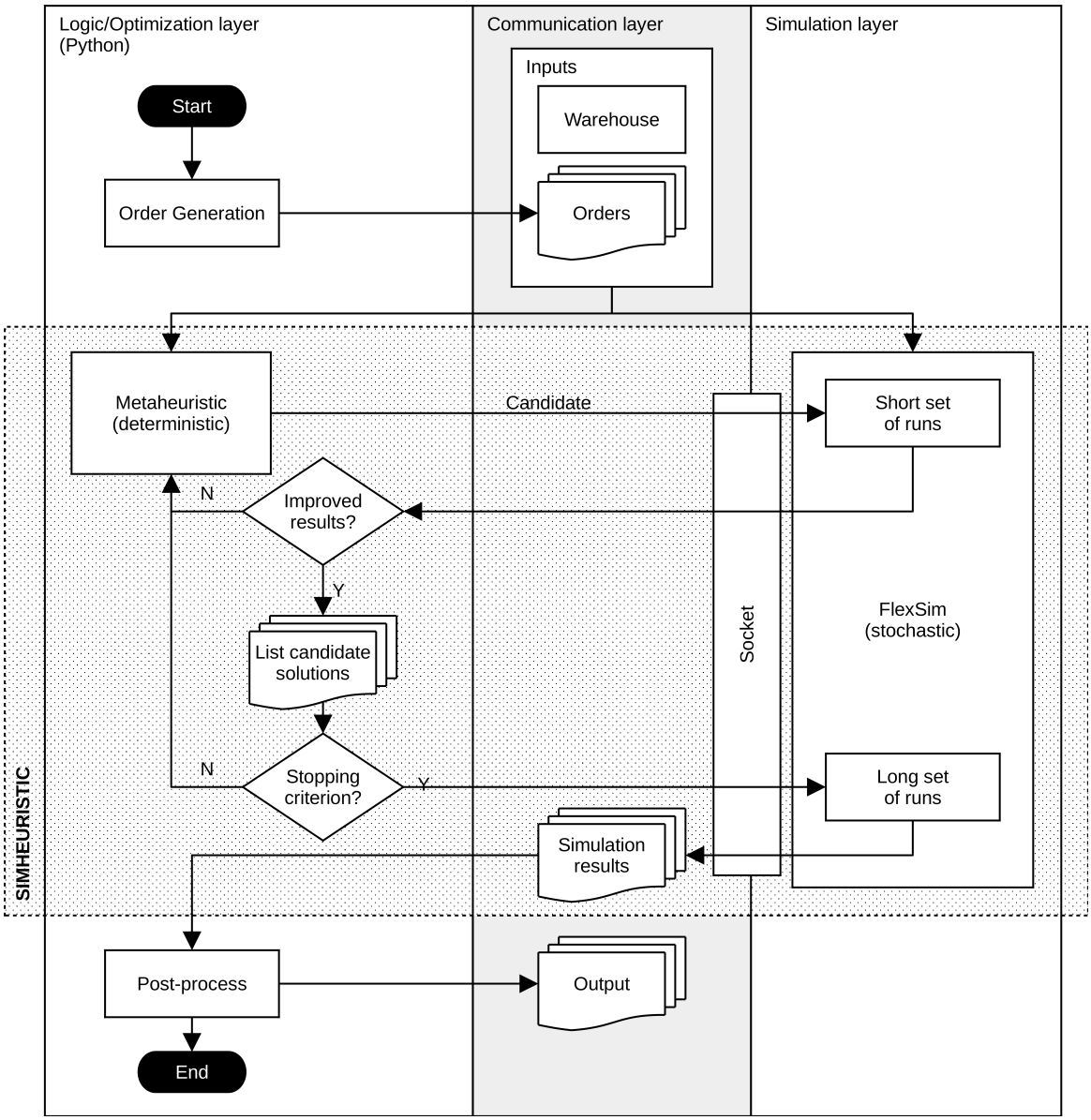


Figure 15.3: Explanation of the simheuristic framework logic.

way as the deterministic model (using the same orders, warehouse layout, etc.), meaning they share the same inputs. FlexSim then runs the simulation for each candidate storage location assignment, providing a realistic evaluation of the objective function (total traveled distance). This process is repeated a set number of times (a relatively small number of simulation runs or replications), with each run producing different results due to the stochastic nature of the model. The results are then averaged and compared, generating a list of candidate solutions, where the best-performing storage locations from both the deterministic and stochastic versions are retained. The elite candidate solutions are then reassessed in the stochastic FlexSim simulator with a greater number of replications. Subsequently, the results are analyzed. The final analysis, or post-processing, is conducted to evaluate the quality of each storage location assignment solution in terms of both its performance and variability. It is important to note that the performance in the deterministic phase serves merely as a proxy for the stochastic solution. Therefore, the true outcome of the simheuristic is determined by the performance of the solution in the stochastic simulator, following statistical treatment.

15.3.1 Order Generation

A Python module was developed to simulate the orders that would be received at the warehouse. The orders are generated using a pseudo-random number generator, which considers several parameters as the seed, including the date, warehouse size, number of products, and the number of orders (customers) to be served. This method of generating plausible orders ensures they can be reproduced by using the same parameters (i.e., the same seed). By creating identical orders, the instances can potentially be validated by other researchers. The order generator also incorporates the Pareto distribution to model the frequency with which each product is requested by customers. If such data is available, these generated orders can be substituted with actual orders from the warehouse under consideration. It is important to distinguish between the set of orders O (i.e., the lists of products to be retrieved for each customer) and the actual sequence of product retrieval $S(o_m)$. The distinction lies in the fact that the sequence $S(o_m)$, as defined in Subsection 15.3, contains more information than the order o_m. Multiple equivalent arrangements can be created by altering the order of the products; however, there exists only one optimized sequence. This sequence is obtained from a given order through the application of a heuristic method while concurrently assessing the solution.

15.3.2 Warehouse Navigation

The Python algorithm used to assess the quality of various solutions requires a consistent method for calculating distances within the warehouse. These distances can be determined

using basic metrics, such as Euclidean distance or Manhattan distance. However, to ensure that these distances are somewhat realistic, it is essential to take into account the obstacles present in the warehouse, particularly the racks. It's important to highlight that the distances analyzed within the simulation aspect of the simheuristic (FlexSim) are deemed authentic because they relate to a specific unit of measurement (such as meters), which aligns with the geometry of the warehouse. For the Python algorithm, it is essential to conduct a realistic comparison of distances or costs. One potential method for calculating distances that accounts for obstacles is the *A-star* (A^*) algorithm. This widely recognized pathfinding algorithm offers optimal results; however, it comes with a significant limitation: its time and space complexity grows exponentially. Consequently, applying the A^* algorithm in extensive warehouse scenarios may not be feasible. It is important to clarify that within the scope of this research, the term 'path' primarily denotes the sections linking two designated points within the warehouse.

The term 'route' refers to the ultimate pathway that connects the input and output locations while gathering items in accordance with the order sequence. In light of the objectives of this research, a dedicated algorithm has been introduced for warehouse navigation. This algorithm is named *W-star* (W^*), with W representing warehouse and the '$*$' symbol serving as a reminiscence to the name of the A^* navigation algorithm. The primary geometric premise regarding the warehouse's design is that it consists of a "single-block" structure, characterized by uniformly sized, undivided racks. Additionally, it is assumed that there are two square units of space between both sides of the aisle within the discretization grid. The initial and terminal points are considered valid locations for products, and the racks are arranged along the vertical axis of the two-dimensional warehouse layout. All this was demonstrated in Figure 15.2. The pseudocode for the W^* is shown in Algorithm 10. Unlike the A^* algorithm, this method's complexity does not rely on the length of the path. This is because alternative routes do not have to be retained with each movement of a new node. As a result, it enables the computation of larger warehouse scenarios within acceptable time frames.

It can be demonstrated that the suggested W^* navigation algorithm yields high-quality outcomes. The demonstration involves splitting the path into two segments and establishing that there is no superior alternative for each segment (refer to Figure 15.4). The initial aspect involves navigating around the racks. As crossing the racks is not feasible, this movement is inevitable and occurs in a straight horizontal line, representing the most direct route between two points positioned directly above or below the racks. The subsequent aspect pertains to movement within the corridors. Although there are various methods for this type of movement, the distance covered is minimized by the chosen strategy (similar to others, but it cannot be reduced further since diagonal movements are prohibited).

Algorithm 10 W* algorithm

```
1: Inputs: warehouse, start, end
2: Output: path
3: for all direction in ⟨up⟩, ⟨down⟩ do
4:     while horizontal distance is not 0 do
5:         if moving horizontally towards end is allowed then
6:             move 1 step horizontally towards end
7:         else
8:             move 1 step vertically according to direction
9:         end if
10:    end while
11:    while vertical distance is not 0 do
12:        move vertically towards end
13:    end while
14: end for
15: select shortest path between ⟨up⟩ and ⟨down⟩
```

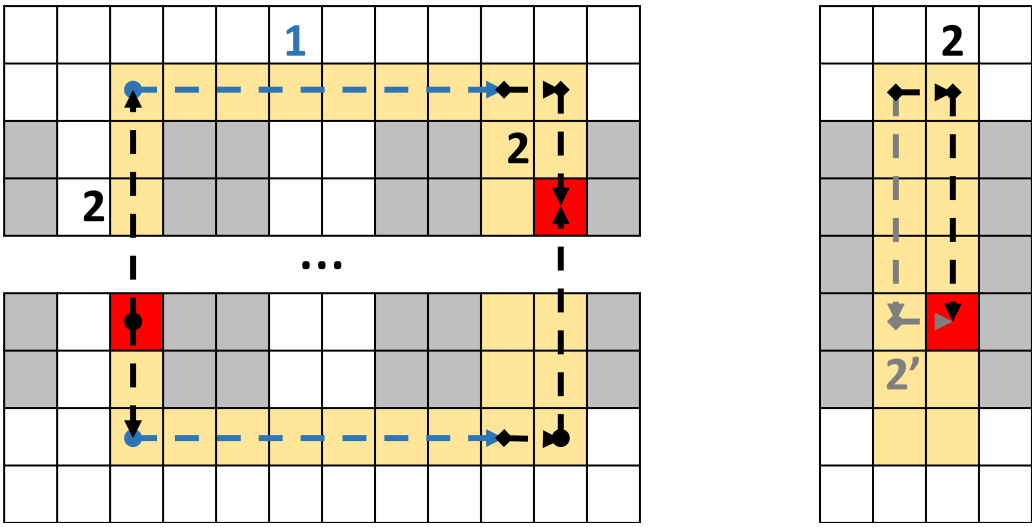


Figure 15.4: The movement around the racks (1) is the shortest. The movement within the aisle (2) is also minimal, though alternative paths (2') may exist with equal or greater length.

15.3.3 Picking Operation and Solution Evaluation

The process of picking or retrieving products is utilized to assess the efficiency of how storage locations are assigned, functioning as a means to evaluate the objective function, specifically the overall distance traveled. In practice, the picking operation is comprised of two primary elements: *(i)* the workers executing the tasks, which involves their number, positioning, and characteristics like speed, agility, and various other skills; and *(ii)* the methodology applied throughout the picking process, encompassing routing, permissible directions and zones, as well as the order of operations to be undertaken. In the case study discussed in this chapter, the picking operation was crafted to ensure a strong alignment between the optimization aspect of the Python algorithm and the simulation model created in FlexSim. As expected, the simulation's picking operation is significantly more intricate; it takes into account elements such as congestion and the minor adjustments needed to access racks and retrieve items, among other considerations. Conversely, the optimization component had to embrace various simplifications. Concerning the picking strategy, employees will consistently gather items according to the predefined sequence $\mathrm{S}(o_m)$. This necessitates that the order lists be restructured to guarantee that a reliable and efficient approach is adhered to during the execution of the picking operation. In practical applications, as long as the strategy is maintained consistently, it is possible to compare two solutions. The literature has introduced several picking and routing strategies, including S-shape, largest gap, return, among others. These heuristics are frequently utilized in real-world scenarios, often surpassing optimal routing techniques. The selection of a particular strategy is influenced by the warehouse layout and item distribution (Koster et al., 1998). In this research, a specific order picking routing was not established. Rather, a simple collection of guidelines was utilized to formulate the ultimate strategy (Algorithm 11). The guidelines for creating the picking route begin at the starting point and determine the closest product location, employing various strategies while maintaining a uniform sequence. When a product is located, it becomes established in the retrieval order, and the algorithm advances from that point. If a product is not located, the search moves on to follow the subsequent rule. The rules for searching are outlined as follows (in sequence): *(i)* Starting from the present location, examine the spot directly behind in the same aisle; *(ii)* From the current position, look for the nearest product within that aisle; and *(iii)* From the existing location, seek out the nearest product in a "radial" direction. Figure 15.5 illustrates the strategy used for constructing this picking route.

It is essential to recognize that this approach is both consistent (deterministic) and logical, often leading to a reduction in the overall route length. However, it does not ensure optimal results. In reality, this method operates as a local greedy search process, which may result in actions that a warehouse manager might consider unsuitable in certain arrangements of storage

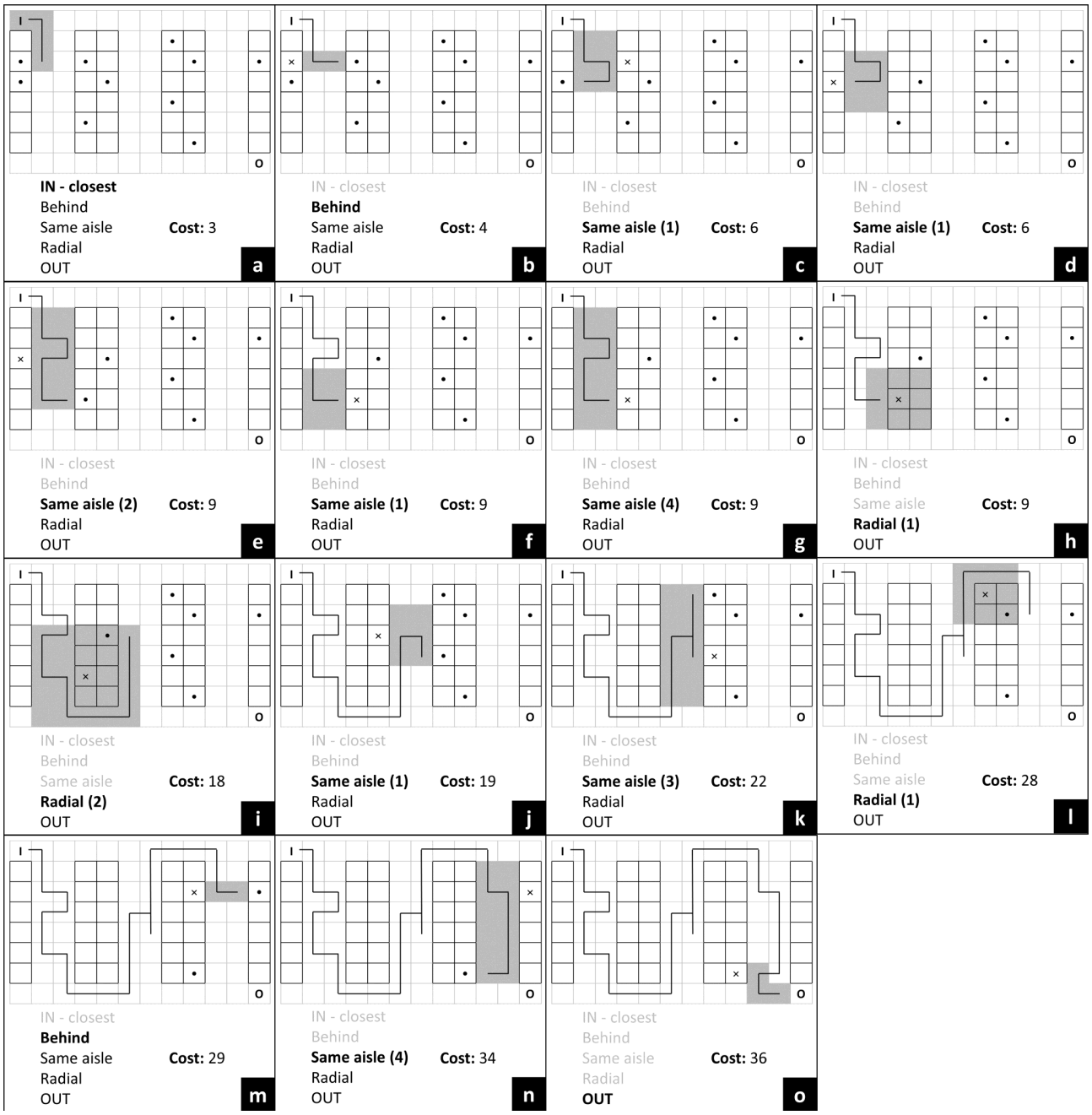


Figure 15.5: Graphic representation of the sequence used to construct the picking route.

Algorithm 11 Evaluate Deterministic Solution

```
1: Inputs: SLA, Orders
2: Outputs: Cost, Sequence
3: cost ← 0
4: for all order ∈ orders do
5:    item ← empty
6:    while order ≠ empty do
7:       if item = empty then
8:          next_item ← find_closest(IN)
9:          cost ← cost+ distance(IN, next_item)
10:      else
11:         step ← 0
12:         while next_item = empty do
13:            step ← step + 1
14:            if behind(item) ≠ empty then
15:               next_item ← behind(item)
16:            else if length(item, step) ≤ aisle_length then
17:               next_item ← search_in_aisle(step)
18:            else
19:               next_item ← search_round(step)
20:            end if
21:         end while
22:         cost ← cost+ distance(item, next_item)
23:      end if
24:      remove item from order
25:      sequence ← add(item)
26:      item ← next_item
27:   end while
28:   cost ← cost+ distance(item, OUT)
29: end for
30: return cost, sequence
```

locations. The benefit of adopting this strategy lies in significantly lowering computational complexity, as it eliminates the need for calculating distance matrices or executing multiple loops throughout the entire warehouse. The ultimate configuration of the route closely mirrors the S-shape picking method, as it aims to finish one entire aisle before proceeding to another. However, based on the distribution of orders or the locations of the input and output points, different patterns of movement may emerge. In essence, the algorithm offers a degree of customization: during the phase when it identifies products within the same aisle, it can be prompted to initiate the "radial" search sooner if the subsequent product down the aisle is located at a considerable distance. This is particularly significant for widely spaced storage areas in large warehouses and will yield results akin to a mid-point strategy. It is essential to note that, in addition to assessing the cost of each SLAP solution in a deterministic manner, Algorithm 11 also outlines the order in which warehouse workers should retrieve items for every order during the FlexSim simulation (and possibly in real-world scenarios). The algorithm used for route-construction has potential for further enhancement. These enhancements, which are beyond the current study's focus, may consist of: *(i)* enabling the search pattern to anticipate slightly ahead to prevent abrupt turns

within the aisle (for instance, refer to movements i, j and k in Figure 15.5, where items could have been collected without making turns inside the aisle), while still limiting how far ahead is considered to avoid a significant increase in computational time; and *(ii)* incorporating an automatic modification of the search pattern's shape rather than maintaining a strictly linear or radial form, allowing it to adjust and accommodate various warehouse layouts and storage location assignments.

15.3.4 A Simheuristic for Solving the Storage Location Assignment Problem

The key element in finding a satisfactory solution to the SLAP is the strategy used to generate high-quality storage location assignments. The strategies employed to generate different storage location assignments enable the exploration of the solution space. In this study, both the random strategy and the Picking Frequency Based Storage Strategy (PFS) were adopted as benchmark storage methods. In the random strategy, items were assigned to storage locations randomly. In the latter approach, items with higher picking frequencies were placed in storage locations nearer to the input and output points. These two methods are among the most commonly used strategies by warehouse managers when addressing the SLAP, representing two extremes of a spectrum: random assignment and dedicated assignment. The class-based assignment strategy falls in between these two approaches. The PFS is a deterministic and greedy heuristic employed for the allocation of items to storage spaces. The process unfolds in a clear sequence: (1) items are organized based on their picking frequency from the given set of orders; (2) storage locations are arranged according to their proximity to both input and output points, incorporating both distances; (3) items and locations are paired in a strict manner, whereby the item with the greatest picking frequency is assigned to the location nearest to both input and output points, continuing this way until all items have been allocated. The simheuristic introduced in this chapter aims to improve benchmark solutions through a BR algorithm. This algorithm incorporates an element of randomness into a greedy heuristic, which helps maintain certain features of its foundational logic while allowing for the investigation of potential solutions near the greedy outcome. In particular, the greedy PFS algorithm builds solutions by choosing items based on their highest picking frequency in a sequential manner, whereas the BR algorithm modifies the item selection process by applying a skewed probability distribution. A geometric probability distribution characterized by the parameter $\beta \in (0, 1)$ was specifically utilized to integrate BR behavior into the greedy PFS. In implementation, the sequence of items arranged by their selection frequency is restructured, with the assumption that the likelihood of an item remaining at position x is defined by:

$$P(\mathrm{X}{=}x) = (1 - \beta)^x \tag{15.3}$$

After completing the fine-tuning process, the parameter β was chosen randomly from a range of 0.5 to 0.7. These values typically provide an effective compromise between achieving uniform randomization of the list ($\beta = 1$) and maintaining the original order of the list ($\beta = 0$) for various optimization problems. The pseudocode presented in Algorithm 12 demonstrates the application of the BR method to this particular SLAP. It adheres to the same principles as the

previously outlined PFS algorithm but incorporates an extra step for randomization. Moreover, this randomization can be utilized to either create a completely arbitrary assignment of storage locations (by adjusting $\beta = 1$) or to revert to the purely greedy approach of the PFS algorithm (by setting $\beta = 0$).

Algorithm 12 Biased Randomized (BR) Heuristic

```
1: Inputs: warehouse, orders, β
2: Output: SLA
3: items_list ← order_items_by_freq(orders)
4: locations_list ← order_locations_by_distance(warehouse)
5: while items_list ≠ empty do
6:     items_list ← randomize(items_list, β)
7:     item ← first(items_list)
8:     location ← first(locations_list)
9:     sla ← add(item, location)
10:    remove item from items_list
11:    remove location from locations_list
12: end while
13: return sla
```

The pseudocode in Algorithm 13 demonstrates how the simheuristic operates by integrating the BR algorithm with the FlexSim DES model. The overall framework is also illustrated in Figure 15.3. The core idea behind the simheuristic is to generate a set of elite candidate solutions that will perform effectively in both the deterministic and stochastic versions of the problem. To achieve this, after an initialization phase where the PFS greedy algorithm is used as a baseline, several improvement iterations are carried out. In each iteration, the BR algorithm is invoked to generate a solution. The solution is first assessed according to its deterministic performance. A stochastic evaluation is conducted only if there is a decrease in the solution cost. This stochastic assessment involves executing a brief simulation phase, during which FlexSim is invoked multiple times, and the resulting stochastic costs from each invocation are averaged. Once the enhancement cycles are concluded and the roster of top solutions is established, an "intensive" assessment phase is initiated. This evaluation stage aims to provide a precise estimate of the stochastic costs associated with each top solution, facilitating the identification of the option with the lowest average cost to be designated as the final result. The essential elements required for the simheuristic include: the count of iterations needed to compute solutions using the BR algorithm, the β parameter to randomize product listings, the layout of the warehouse instance, the orders that need processing, as well as the number of replications designated for a rapid simulation phase and those intended for a more thorough simulation phase.

Algorithm 13 Simheuristic

Inputs: *iterations*, *warehouse*, *orders*, β
Output: *best_solution*
initial_SLA ← BRA(*warehouse*, *orders*, β=0)
best_solution.determ_cost ← evaluate_determ_solution(*initial_SLA*, *orders*)
best_solution.stoch_cost ← Simulation(*best_solution*, *short*)
best_sol_list ← add(*best_solution*)
for $i = 1$ **to** *iterations* **do**
    new_SLA ← BRA(*warehouse*, *orders*, β)
    new_solution.determ_cost ← evaluate_determ_solution(*new_SLA*, *orders*)
    if *new_solution.determ_cost* < *best_solution.determ_cost* **then**
        new_solution.stoch_cost ← Simulation(*new_solution*, *short*)
        if *new_solution.stoch_cost* < *best_solution.stoch_cost* **then**
            best_solution ← *new_solution*
            best_sol_list ← update(*best_solution*)
        end if
    end if
end for
for all *solution* **in** *best_sol_list* **do**
    solution.stoch_cost ← Simulation(*solution*, *long*)
end for
best_solution ← best(*best_sol_list*)
return *best_solution*

15.4 Computational Experiments

This section outlines a series of computational experiments conducted to assess the proposed method. The performance of the simheuristic is compared with three benchmark methods: the greedy PFS, the BR algorithm, and random strategies. The total traveled distance required to complete all order picking operations is used as the primary performance metric for comparison. All algorithms were implemented in Python v3.8.10 and executed on a Windows 10 operating system with an i7-4500U CPU (2.40GHz) and 6 GB of RAM. FlexSim version 22.1.2 was utilized for the simulation model.

15.4.1 Experiment setting

Two warehouse configurations were utilized for the numerical analysis. Configuration A comprises 15 racks, each featuring 10 slots, enabling the storage of a maximum of 150 distinct products. In contrast, configuration B includes 42 racks with 18 slots each, facilitating the accommodation of up to 756 unique products. The dimensions of configuration B are modeled after those presented in Lee, Chung, et al. (2020). Customer orders were generated through an independent Python script that uses a consistent random seed for every calendar day to guarantee reproducibility. In essence, the script can produce identical orders for a specific date and consistent parameters. As detailed in Subsection 15.3.1, these orders adhere to the Pareto distribution, indicating

that around 20% of the products will account for almost 80% of customer orders. Figure 15.6 demonstrates the FlexSim execution of instance A. For the evaluation of the number of runs, every solution produced by various methods is assessed in the stochastic FlexSim environment through 10 replications, each utilizing a distinct random seed to ascertain its corresponding stochastic cost. To facilitate an equitable comparison between the BR algorithm and the suggested simheuristic approach, both methodologies were permitted to iterate 50 times in order to identify the optimal solution.

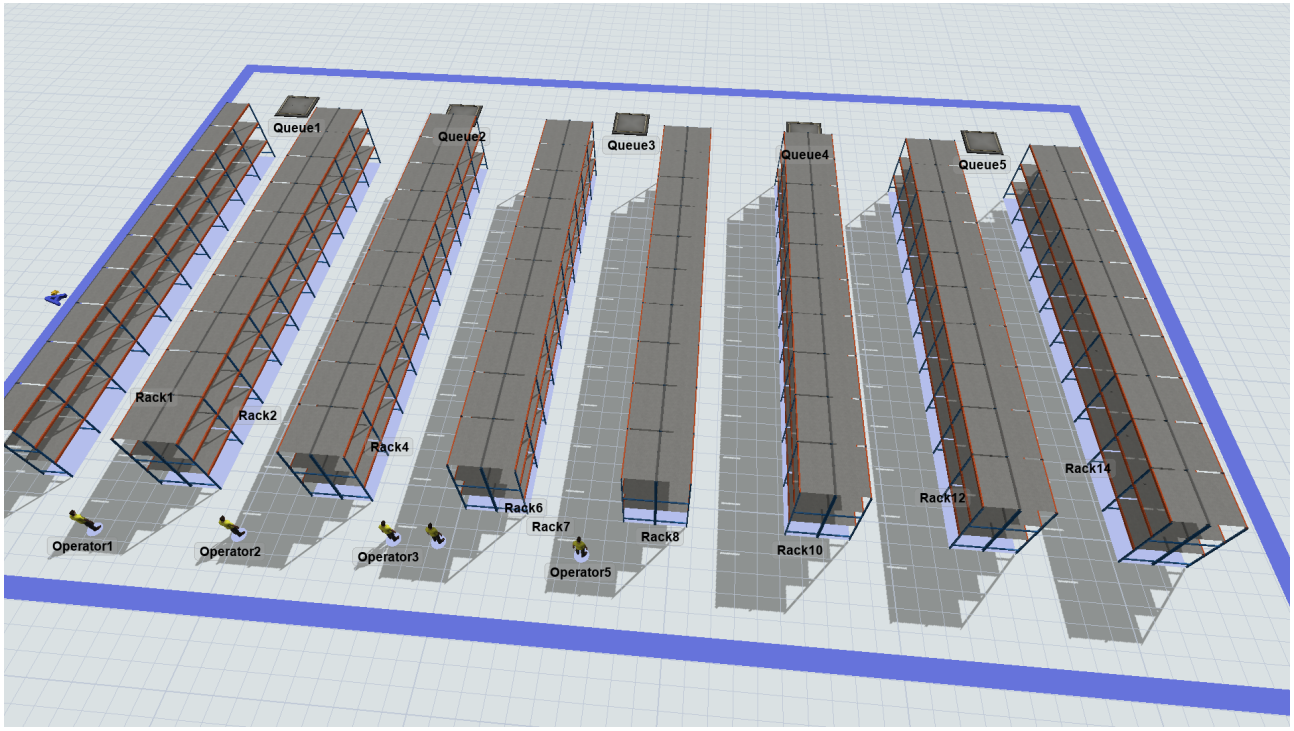


Figure 15.6: FlexSim screenshot of instance A ($15x10$).

15.4.2 FlexSim modeling

The operation of the FlexSim stochastic model reflects all the assumptions established in the Python deterministic model, albeit with certain essential variations. These variations contribute to a more realistic portrayal in the FlexSim simulation compared to its deterministic counterpart. For example, navigation within the simulation utilizes FlexSim's proprietary A^* algorithm, which assesses minimum distances with greater accuracy by considering all existing obstacles, particularly other workers. Additionally, another distinction between the two models pertains to the origins and destinations of the picking routes. In the Python warehouse model, there is one designated input point and one output point. Pickers commence the collection of products at the input point and deposit the gathered items at the output point. Conversely, in the FlexSim

warehouse model, several output points are located on one side of the warehouse, simulating a dispatch area found in an actual warehouse. Here, the location where pickers begin retrieving items for the next order corresponds to where they completed their previous order. This scenario accurately reflects real-life operations in many voice-assisted picking systems, where workers do not have to relocate to a designated area to obtain the next order list. Additionally, the number of employees and output locations can differ based on the specific situation at hand. In practical applications, the workforce available for order picking may fluctuate depending on various factors such as holidays or sick leave, which could introduce another layer of randomness into the model. In the current scenario, it was assumed that variations in worker numbers, both across different instances and within the same instance, have minimal impact. This assumption stems from the fact that a majority of the distance covered involves movement between item locations, with relevance only arising if aisle congestion reaches critical levels. Additionally, as warehouse instances expand in size, the influence of multiple output points in the simulation may become substantial, thereby increasing the disparity between the simulation results and those predicted by the deterministic model.

15.5 Analysis of Results

Given the stochastic nature of the problem in real-life scenarios, the results obtained were analyzed statistically by varying the set of orders. Warehouse instance A was tested using 10 different sets of orders, while instance B was tested with only 3 different sets of orders due to its higher computational cost. This approach prevents improvements being made to the storage location assignment for a specific day or set of orders, thereby increasing the confidence in the generated results. To further test the robustness of the simheuristic method, 9 different combinations of input and output points were considered for the deterministic part of the simheuristic. Specifically, all possible combinations of left (L), center (C), and right (R) positions for the input and output points were analyzed, as shown in Tables 15.1 and 15.2. These tables display the traveled distance for each type of solution, for every combination of input and output points, averaged across the set of orders considered for each instance. The gap between the simheuristic solution and each benchmark solution is calculated as the percentage difference, which is used to assess their performance. As the goal of the optimization is to minimize the traveled distances, a negative gap signifies that the simheuristic algorithm achieved better performance than the benchmark solution in that case.

As shown in both tables, the simheuristic method consistently outperforms the other methods, with the traveled distance being lower in nearly all cases for the simheuristic approach. However, in instance B, while the performance difference is still present and generally favors

Table 15.1: Comparison of methods and gap to the simheuristic solution for instance A.

Warehouse instance A							
Input/Output	Random [1]	Greedy [2]	BR [3]	Simheuristic [4]	Gap [1] - [4]	Gap [2] - [4]	Gap [3] - [4]
C / C	4163	4102	4058	3998	-4.0%	-2.5%	-1.5%
C / L	4171	3938	3905	3827	-8.2%	-2.8%	-2.0%
C / R	4128	4129	4074	3997	-3.2%	-3.2%	-1.9%
L / C	3985	3775	3710	3675	-7.8%	-2.6%	-0.9%
L / L	3978	3682	3658	3593	-9.7%	-2.4%	-1.8%
L / R	4003	3946	3936	3882	-3.0%	-1.6%	-1.4%
R / C	4167	4122	4076	4050	-2.8%	-1.7%	-0.6%
R / L	4191	4215	4197	4111	-1.9%	-2.5%	-2.0%
R / R	4214	4081	4059	4002	-5.0%	-1.9%	-1.4%
Average	4111	3999	3964	3904	-5.1%	-2.4%	-1.5%

Table 15.2: Comparison of methods and gap to the simheuristic solution for instance B.

Warehouse instance B							
Input/Output	Random [1]	Greedy [2]	BR [3]	Simheuristic [4]	Gap [1] - [4]	Gap [2] - [4]	Gap [3] - [4]
C / C	30013	29497	29272	29317	-2.3%	-0.6%	0.2%
C / L	29996	28718	28741	28502	-5.0%	-0.8%	-0.8%
C / R	30333	30088	29816	29710	-2.1%	-1.3%	-0.4%
L / C	29242	27923	27799	27799	-4.9%	-0.4%	0.0%
L / L	29320	27414	27322	27106	-7.6%	-1.1%	-0.8%
L / R	29629	28647	28434	28330	-4.4%	-1.1%	-0.4%
R / C	30049	29964	29768	29814	-0.8%	-0.5%	0.2%
R / L	30113	29638	29385	29263	-2.8%	-1.3%	-0.4%
R / R	30038	29979	30016	29763	-0.9%	-0.7%	-0.8%
Average	29859	29096	28950	28845	-3.4%	-0.9%	-0.4%

the simheuristic, it is much smaller. This is visually represented in both Figures 15.7 and 15.8. The influence of other parameters, such as the number of orders to be picked, was also analyzed, but no significant impact was observed. The scenario in which the number of products per order can be modified was also examined, but only the gap between the random method and the other methods was impacted. This suggests that for shorter orders (fewer items to collect), the random policy could be a viable option, as it yields comparable results and is much easier to implement. However, for orders with more items to collect, the greedy, BR algorithm, and simheuristic methods demonstrated a clear advantage. Nevertheless, the relative improvement between these three methods remained consistent, with the simheuristic consistently performing the best among them.

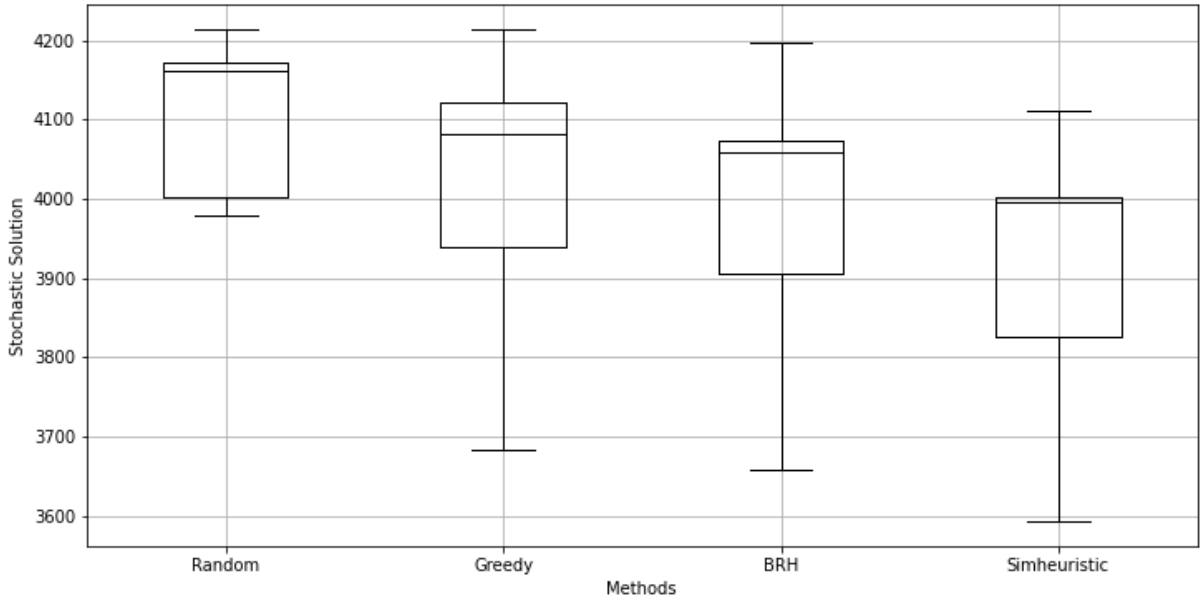


Figure 15.7: Average distance of stochastic versions across methods for instance A.

It was observed that the results of the random strategy were less scattered compared to the other methods. Additionally, the evaluations of the other methods exhibited a similar degree of variability. One possible explanation is that the random solution performs consistently, regardless of the stochastic variability present in the FlexSim simulator, making it less sensitive to such factors. Conversely, solutions that are more optimal in the deterministic version of the problem may exhibit greater variability in their results when assessed in a stochastic environment. For example, a deterministic solution might depend on workers consistently starting their picking routes near one side of the warehouse. However, in the stochastic model, products may occasionally need to be dropped off at the opposite end of the warehouse, negating the advantage of the deterministic solution. Although the simheuristic has a similar level of variability, it tends to be biased towards minimizing traveled distance. The results also

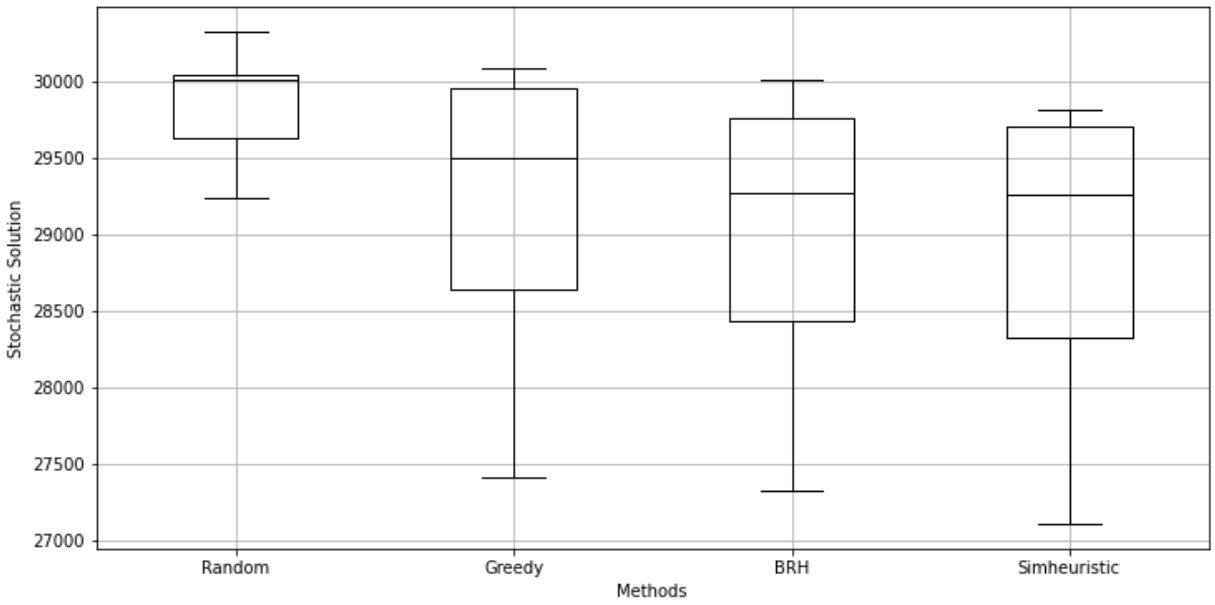


Figure 15.8: Average distance of stochastic versions across methods for instance B.

indicate that the simheuristic's ability to enhance the BR algorithm's performance is reduced in warehouse instance B (the larger instance) compared to warehouse A. This may be primarily attributed to the additional input and output points present in the FlexSim model, as opposed to the 3 points (left, center, right) considered in the deterministic model. Nevertheless, this is a direct consequence of the warehouse size; in most real-world scenarios, the larger the warehouse, the more extensive the outbound area. It is rare to find studies in the literature that account for this reality: in a real warehouse, the outbound area typically resembles a continuous section of the warehouse rather than a single point. This study assumes that the FlexSim stochastic environment provides a more realistic representation of the warehouse, while the deterministic model serves as a tool for efficiently exploring the optimization space. It is evident that as the differences between the deterministic and stochastic models increase, the correlation between them will weaken, thereby reducing the effectiveness of the method. Finally, the dependence of optimization results on the specific set of orders analyzed has noteworthy implications. On one hand, if this dependency were interpreted rigidly, it would be impractical, as it would require reconfiguring the warehouse for every new set of orders received. On the other hand, a more flexible interpretation of this dependency could transform it into a highly valuable managerial tool. For example, the set of orders to be analyzed could be prepared in advance to reflect historical demand in the warehouse or to evaluate various "what-if" scenarios. In such cases, the methodology proposed in this study offers a robust storage location assignment that minimizes warehouse travel distance for the given set of orders. Consequently, decision-makers can compare the warehouse's existing configuration with the solutions generated by the simheuristic and take

appropriate actions. In other words, it enables the exploration of stochastically robust solutions tailored to a specific order profile of interest.

Chapter 16

Discrete-Event Simheuristic for Team Orienteering Problem

This chapter[1] introduces a innovative methodology aimed at enhancing urban mobility within smart cities, focusing on a sophisticated variant of the TOP. The research uses real-world datasets obtained from transportation hubs in Barcelona, sourced from Open Data Barcelona. The study centers on optimizing routes for autonomous delivery drones operating on a university campus, tasked with transporting items to specific drop-off locations. Conventional optimization methods often overlook the stochastic variability present in urban traffic patterns and the intricate interdependencies among diverse transportation systems. To overcome these limitations, this chapter utilizes advanced commercial simulation tools capable of accurately modeling the dynamic and unpredictable nature of urban settings. This simulation framework serves as a reliable basis for assessing heuristic-based solutions. The effectiveness and real-world applicability of the proposed simheuristic approach are validated through extensive computational experiments conducted in a practical urban scenario. By integrating simulation with optimization, this work bridges the divide between theoretical route planning and real-world implementation in urban environments. The findings underscore the importance of simulation in enhancing the robustness and relevance of urban mobility strategies, addressing the shortcomings of traditional optimization methods.

[1]The contents of this part is based on the following work:

- **Peyman, Mohammad**, Xabier A Martin, Javier Panadero, and Angel A Juan (2025).“A discrete-event simheuristic for enhancing urban mobility”. In: Simulation Modelling Practice and Theory 140, p. 103084.

16.1 Problem Description

Urban transportation management poses several complex challenges, including questions such as: "What strategies can be implemented to minimize traffic congestion?", "How can routing systems dynamically adjust to unexpected traffic fluctuations?", and "Is the existing infrastructure capable of meeting future transportation demands?" These challenges stem from differing interpretations of what constitutes "effectiveness", frequently neglected factors such as fluctuating traffic patterns, and the intricate relationships between various elements of the transportation network. Recent advancements in analytical methodologies, including the integration of heuristic algorithms with DES, provide innovative approaches to tackle these challenges (Upadhyay et al., 2024). Our research centers on a specific variant of the TOP, a challenging optimization problem in which vehicles seek to maximize rewards by visiting designated locations while adhering to predefined constraints (Chao et al., 1996). This methodology is essential for addressing the uncertain and dynamic nature of urban traffic, as well as for optimizing resource allocation within transportation networks (Kirac et al., 2023; Hammami, 2024). The inherent uncertainties in urban transportation systems, such as unpredictable travel times caused by traffic congestion or adverse weather conditions, necessitate the use of probabilistic approaches. These methods often involve modeling variations in travel times through random variables. For example, Wu et al. (2024) investigates stochastic systems by leveraging simulation techniques. Within this framework, the stochastic TOP integrates uncertainties, including variable travel durations and dynamic reward structures, to better reflect real-world conditions. As discussed in Chapter 3, the existing body of literature examines both deterministic and stochastic methodologies for addressing the TOP. A majority of these studies utilize MCS to capture the uncertainties inherent in real-world situations. However, this review highlights a notable gap in the use of DES to tackle specific challenges associated with the problem. DES provides a scalable and flexible framework that has the potential to substantially improve modeling and solution approaches for the TOP, particularly in more intricate and dynamic environments. Consequently, the incorporation of DES into the TOP presents a promising avenue for future research. This integration holds the potential to unlock new insights and solutions for addressing complex variants of the TOP. In this context, this section proposes a novel methodology that integrates heuristic algorithms with DES. Specifically, it utilizes two distinct optimization techniques for numerical comparison, alongside FlexSim, a widely recognized commercial simulation tool, to accurately model the dynamic and unpredictable characteristics of urban environments Sanchez et al. (2002).Thus, the key contributions of this research are as follows: *(i)* the creation of a discrete-event model that accurately captures the realistic complexities of urban mobility; and *(ii)* the proposal of a combined heuristic-simulation framework that leverages DES to address a

practical variant of the TOP. To the best of our knowledge, the fusion of DES with heuristic optimization represents a promising and underexplored research direction, offering significant potential for advancing the understanding and resolution of intricate TOP variants.

16.2 Problem Modeling

This matter stems from a case study that investigates the implementation of autonomous delivery drones in regulated environments, including expansive campuses, industrial areas, and gigafactories. In particular, it highlights the difficulties encountered by last-mile delivery systems in a university context. Current university campuses are increasingly demanding prompt and dependable transportation of items such as packages, food, and laboratory supplies. Autonomous delivery drones offer a novel approach by effectively navigating the campus to deliver goods straight to designated drop-off points. However, operational constraints emerge from the presence of shared routes or limited access points in certain areas of the campus, which allows only one robot to operate within that space at any moment. This measure is implemented to prevent overcrowding and ensure safety. Furthermore, delays are experienced at drop-off locations because when a robot is serving a node in a particular area, other drones aiming for different nodes within the same zone must remain in a specified staging area. This arrangement is essential since the shared pathways or drop-off sites can only support one robot simultaneously. Operational policies are implemented to avoid the simultaneous presence of multiple drones in a single location, as this could disrupt pedestrian flow and create safety hazards. To achieve this, it is essential to effectively manage the routing of autonomous delivery drones, ensuring compliance with the limitations set by shared resources and operational protocols. Figure 16.1 presents a schematic overview of the problem at hand. A group of vehicles (V1, V2) sets off from a starting depot, navigates through selected nodes in the network to gather rewards, and eventually reaches a destination depot. The nodes within the network are categorized into three separate interconnected zones, with each node in a given zone having access to shared resources that are limited, including charging stations, fueling locations, or service docks. To avoid resource conflicts and maximize efficiency, the system permits only one vehicle to service an interconnected node at any moment. In particular, a single vehicle can operate within the interconnected zone at a time, while other vehicles seeking to attend to different interconnected nodes must remain on standby until the first vehicle has finished its task. If V1 is attending to node 13 in Zone Close, and V2 reaches node 10, V2 is required to wait until V1 completes its service at that node. This period of waiting is included in the overall route travel time. After V1 leaves the node, V2 can then begin its service, contributing its own service duration to the total travel time for the route.

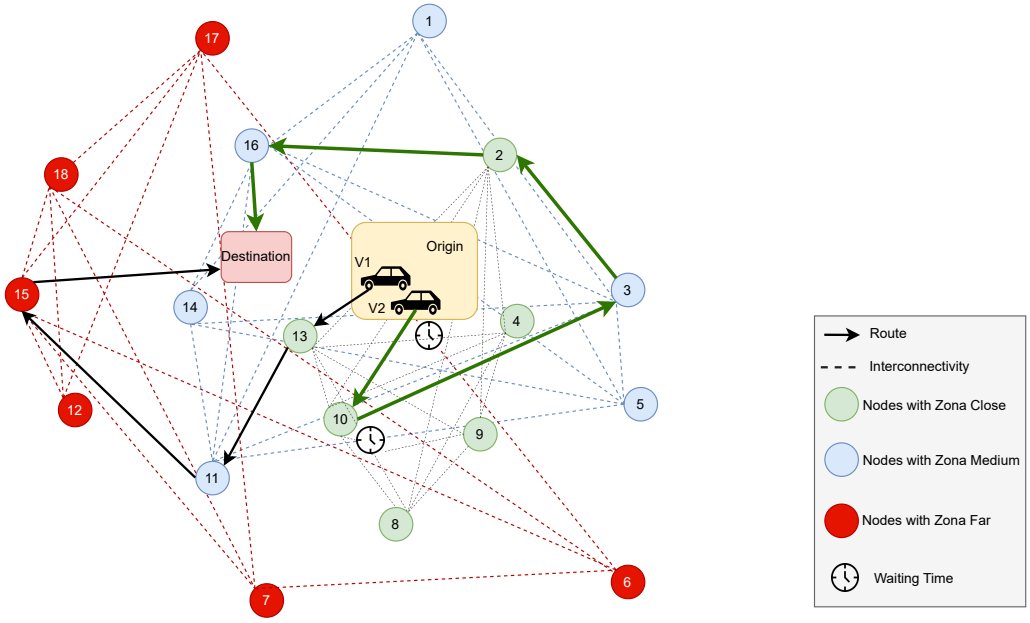


Figure 16.1: Schema of the described problem with interconnected nodes.

16.3 Methodological Approach

A simheuristic method is proposed that integrates a heuristic optimization element with DES. Two distinct approaches have been formulated for the heuristic component: a BR heuristic utilizing the MS framework (BR-MS) and a BR heuristic employing ILS (BR-ILS), both of which are applied for numerical evaluations. The BR-MS provides an easier implementation, making it a practical option for quickly generating numerous high-quality solutions (Martí et al., 2013). Its straightforwardness and effectiveness are ideally suited for situations that demand swift exploration of the solution space. However, the BR-MS method does not incorporate the iterative refinement process found in more sophisticated techniques, which may restrict its capacity to enhance solutions, especially in intricate optimization scenarios. Conversely, the BR-ILS utilizes an iterative strategy that switches between perturbation and localized search centered on a foundational solution. This methodology enables it to attain more polished solutions and effectively balance exploration with exploitation (Lourenço et al., 2019). Despite its benefits, the BR-ILS presents a greater level of complexity in implementation because it necessitates the establishment of its perturbation and local search strategies. Additionally, it demands more extensive computational resources relative to the BR-MS, particularly in large-scale scenarios or when stringent stopping conditions are enforced. Furthermore, commercial simulation software such as FlexSim effectively represents the intricacies associated with real-world optimization challenges.

Figure 16.2 illustrates a schematic representation of our simheuristic framework. This

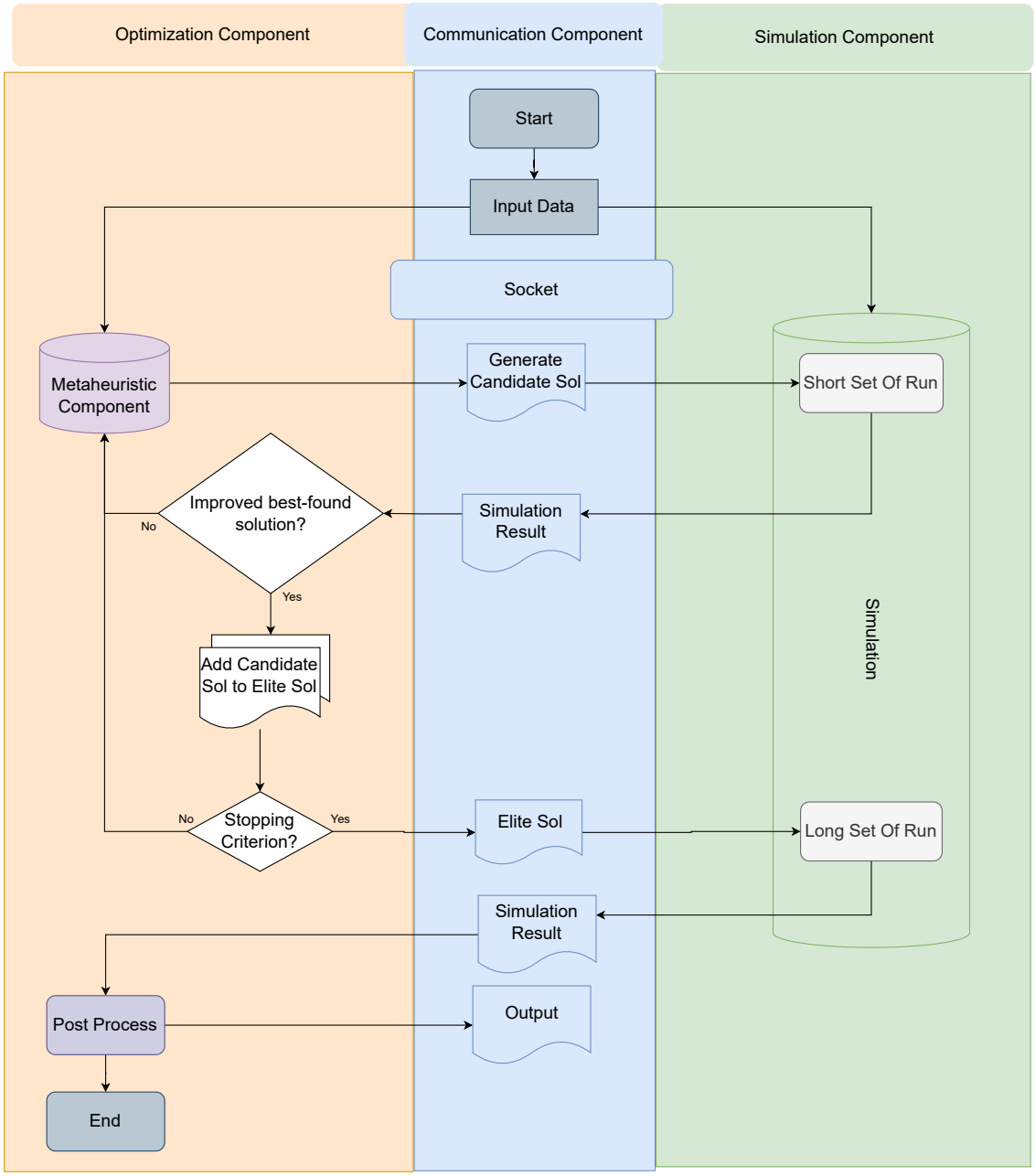


Figure 16.2: Description of the discrete events simheuristic framework logic.

process initiates with the algorithm accepting an input instance that comprises information such as the nodes to be visited, the upper limit on the number of vehicles, and the maximum allowable travel time. Initially, the stochastic inputs are transformed into deterministic forms by substituting random variables with their expected values. Following this, candidate solutions are generated in an iterative manner until a predefined stopping criterion is achieved, utilizing a BR metaheuristic component (either BR-MS or BR-ILS). As discussed in Part II, BR is a

method for transforming a deterministic procedure into a probabilistic algorithm. Essentially, it incorporates controlled randomness into the deterministic process, progressively diverging from the initial methodology while preserving its fundamental logic. The level of randomness introduced is governed by a geometric distribution characterized by the parameter $\beta \in (0, 1)$. A lower value of β (approaching 0) yields a selection that is more random, whereas a higher value of β (approaching 1) results in a selection that is more deterministic or greedy. The potential solutions for the deterministic issue produced by the optimization component are subsequently forwarded to the simulation phase to assess their effectiveness in uncertain conditions. In particular, a limited number of simulation runs utilizing FlexSim are performed to estimate the anticipated reward associated with these promising solutions. The most favorable solutions are then incorporated into a collection of elite options. Once the stopping criterion is satisfied, a more thorough assessment of the elite solutions is conducted under uncertain conditions across a greater number of simulation iterations to achieve a more accurate evaluation of these top solutions. Ultimately, from this collection of elite solutions, the algorithm identifies and returns the solution that offers the greatest expected reward. It is important to note that an effective integration between the BR component and the commercial simulation software is crucial for enabling a responsive simheuristic framework. Therefore, we utilize an effective inter-process communication method referred to as "sockets" to facilitate smooth and adaptable interactions. This approach is especially advantageous for our research as it enhances the optimization-simulation cycle, which is crucial for addressing the outlined issue within brief computing durations.

Algorithm 14 details the primary elements of the constructive heuristic introduced by Panadero, Juan, Bayliss, et al. (2020) for producing viable solutions to the TOP. The process initiates by creating an initial "dummy" solution, which includes as many routes as there are reachable nodes within the maximum travel time. Each route originates at the depot, visits a selected node, and concludes at the destination depot. Subsequently, the enriched savings s_{ij} for each node pair i and j are computed using Equation 16.1 as follows:

$$s_{i,j} = \alpha(t_{1,j} + t_{i,n} - t_{i,j}) + (1 - \alpha)(u_i + u_j) \tag{16.1}$$

The parameter α, which lies in the range of $(0, 1)$, serves to balance the savings in travel time against the total reward. This parameter is adjusted based on empirical observations during the generation of the initial solution. Specifically, an appropriate value for α is established through a grid search tuning method. In this process, several evenly distributed values of α within the interval $(0, 1)$ are created, and solutions are derived using a constructive heuristic for each selected value. Subsequently, we identify the starting solution and α value that provide the maximum reward. The enriched savings are then arranged in descending order. Following this,

an iterative process of merging routes commences, during which we combine routes according to the enriched savings. In each iteration, the next saving is chosen using a BR approach, and if the criteria are met, the corresponding routes are acquired and merged. This procedure continues until the savings list is exhausted. Ultimately, from all the routes produced in the preceding steps, the optimal routes matching the number of vehicles are chosen to maximize the overall reward.

Algorithm 14 Savings-based heuristic

```
1: sol ← genDummySolution(inputs)
2: savingsList ← genSortedSavingsList(inputs, α)
3: while savingsList ≠ ∅ do
4:     ijEdge ← selectNext(savingsList)
5:     iRoute ← getOriginRoute(ijEdge)
6:     jRoute ← getEndRoute(ijEdge)
7:     newRoute ← mergeRoutes(iRoute, jRoute)
8:     isMergePossible ← checkMergeConditions(newRoute)
9:     if isMergePossible = true then
10:        deleteRoute(sol, iRoute)
11:        deleteRoute(sol, jRoute)
12:        addRoute(sol, newRoute)
13:    end if
14:    deleteEdge(savingsList, ijEdge)
15: end while
16: sortRoutesByProfit(sol)
17: deleteRoutesByProfit(sol, v_max)
18: return sol
```

Algorithm 15 delineates the primary elements of the BR-MS optimization framework. The procedure commences with the creation of an initial solution through the specified constructive heuristic. This initial solution is subjected to a limited series of simulation runs to assess its performance in uncertain conditions, establishing it as the current best option. Additionally, an elite solutions pool is initialized and updated with this initial solution. Subsequently, candidate solutions are produced until a predetermined stopping condition is achieved. At each iteration, a new candidate solution is produced utilizing a BR variant of the constructive heuristic. This approach enables the generation of multiple solutions that adhere to the heuristic framework, allowing for exploration across various regions of the solution space. When a newly created solution exceeds the performance of the existing best solution, it undergoes a limited number of simulation trials to assess its anticipated reward. Should this new solution achieve a higher expected reward than the current best, it will replace it, and the elite solutions pool will be refreshed. After reaching the stopping criterion, an extended series of simulation runs is conducted on the elite solutions retained in the pool, ultimately yielding the best solution.

Algorithm 15 BR-MS Simheuristic Algorithm

```
1: initSol, α ← genInitSol(instance)
2: simulation(initSol, nShort)
3: bestSol ← initSol
4: eliteSols ← ∅
5: update(eliteSols, initSol)
6: elapsed ← 0
7: startTime ← currentTime()
8: while elapsed ≤ maxTime do
9:     newSol ← BR-Heuristic(instance, α, β)
10:    if reward(newSol) > reward(bestSol) then
11:        simulation(newSol, nShort)
12:        if expReward(newSol) > expReward(bestSol) then
13:            bestSol ← newSol
14:        end if
15:        update(eliteSols, newSol)
16:    end if
17:    elapsed ← currentTime() − startTime
18: end while
19: for sol ∈ eliteSols do
20:    simulation(sol, nLong)
21: end for
22: sort(eliteSols)
23: return first(eliteSols)
```

Algorithm 16 illustrates the key features of the BR-ILS component. Initially, a viable starting solution is created through a constructive heuristic. Subsequently, a local search method is utilized to enhance this solution, incorporating a 2-opt operator along with both remove-based and insert-based operators, as outlined in the local search framework proposed by (Panadero, Juan, Bayliss, et al., 2020). The enhanced initial solution undergoes a limited number of simulation runs to evaluate its reward amid uncertainty. Subsequently, this initial solution is designated as both the base solution and the current optimal solution for the BR-ILS. In addition, it is incorporated into a collection of elite solutions for further assessment under uncertain conditions. Following this, the base solution is modified to create new solutions, and a local search operator is utilized to refine these solutions until a predetermined stopping criterion is fulfilled. The perturbation phase utilizes a process of destruction and reconstruction, where certain paths of the solution are partially eliminated and then rebuilt using the BR constructive heuristic. Following this, a local search is performed to enhance the solution. Should the new solution yield better rewards than the original, a limited number of simulation runs will be conducted to assess its performance in uncertain conditions. If the new approach yields a higher expected reward than the existing solution, it takes the place of the base solution. Likewise, if this solution enhances the current best option, it is designated as the new best solution, leading to an update in the elite solutions pool. During the search for solutions, non-improving options may still be accepted through a credit system (Talbi, 2009). Once the stopping criterion is

satisfied, an increased number of simulation runs is conducted on the top solutions retained in the pool, from which the optimal solution is selected and returned.

Algorithm 16 BR-ILS Simheuristic Algorithm

```
1: initSol, α ← genInitSol(instance)
2: initSol ← localSearch(initSol)
3: simulation(initSol, nShort)
4: baseSol ← initSol
5: bestSol ← initSol
6: eliteSols ← ∅
7: update(eliteSols, initSol)
8: credit ← 0
9: elapsed ← currentTime()
10: while elapsed ≤ maxTime do
11:     perturbedSol ← perturbation(baseSol, α, β)
12:     newSol ← localSearch(perturbedSol)
13:     Δ ← reward(baseSol) − reward(newSol)
14:     if Δ < 0 then
15:         simulation(newSol, nShort)
16:         Δ ← expReward(baseSol) − expReward(newSol)
17:         if Δ < 0 then
18:             credit ← −Δ
19:             baseSol ← newSol
20:             if expReward(newSol) < expReward(bestSol) then
21:                 bestSol ← newSol
22:             end if
23:             update(eliteSols, newSol)
24:         end if
25:     end if
26:     if 0 < Δ ≤ credit then
27:         credit ← 0
28:         baseSol ← newSol
29:     end if
30:     elapsed ← currentTime() +startTime
31: end while
32: for sol ∈ eliteSols do
33:     simulation(sol, nLong)
34: end for
35: sort(eliteSols)
36: return first(eliteSols)
```

The simulation model recreates the specific problem scenario, maintaining identical nodes, the highest allowable number of vehicles, and the utmost travel duration. Essentially, it mirrors the problem case intended for resolution within the software framework. When the simulation process is initiated, it generates a solution that must be assessed in the context of uncertainty, along with the count of simulation replications. At every replication of the simulation, the developed model is initiated, allowing vehicles to move from the starting depot to the destination depot while stopping at the specified nodes as per the solution. Nevertheless, if a vehicle needs to stop at a node that is linked to another node being serviced by a different vehicle, it must remain on hold until the second vehicle completes its service at that node. This prolongs the

travel duration for the initial vehicle, as it must pause until the connected area is available for servicing. Upon reaching the destination depot, the anticipated travel times and accrued rewards are calculated. Specifically, the expected travel durations and accrued rewards are calculated by summing the expected travel times and rewards from each route included in the solution. Should the anticipated route travel duration surpass the permissible maximum, a penalty will be applied to that route. This penalty leads to a decrease in the rewards gathered from the nodes visited, proportional to how much the actual travel time exceeds the allowed limit (i.e., the difference between the actual travel time and the maximum permitted duration). Next, the expected travel times and total rewards are consolidated into variables initialized to zero at the beginning of the simulation process. After completing all simulation replications, the average expected travel time and total reward for the solution are determined by calculating the mean of the accumulated expected travel times and collected rewards from all replications, respectively.

16.4 Computational Experiments

The computational experiments were carried out on a Windows 11 operating system, featuring an i7-1165G7 CPU running at 2.80GHz and equipped with 16 GB of RAM. The BR-MS and BR-ILS methodologies were executed using Python 3.11, while FlexSim version 2023.0.8 was employed for the simulation model. To fine-tune the algorithm parameters, a randomly chosen set of problem instances was utilized. The stopping condition for both the BR-MS and BR-ILS was established at 2500 iterations. Furthermore, the β parameter of the geometric distribution was randomly selected from the range of (0.1, 0.3). Due to the absence of publicly available instances for this specific variant of the TOP, we adapted the standard instances for the deterministic TOP as introduced by (Chao et al., 1996). This recognized benchmark suite consists of 320 instances categorized into seven primary groups. Each instance follows the labeling format px.y.z, where x represents the group number, y denotes the count of vehicles, which ranges from 2 to 4 based on the specific instance, and z signifies the instance number. These instances have been extensively utilized in prior studies to evaluate the effectiveness of algorithms in addressing the deterministic TOP. In our research, we performed simulation modeling and numerical experiments by randomly choosing 24 instances from two distinct sets. The first set, designated as p3.y.z, consists of instances featuring 33 nodes and incorporates either 2, 3, or 4 vehicles. The second set, identified as p6.y.z, contains instances with 64 nodes and similarly involves 2, 3, or 4 vehicles. For the simulation modeling conducted in FlexSim, we established a network comprising interconnected nodes by analyzing the previously detailed instances, which led to the formation of the resulting nodes, their interconnections, and the

vehicles. To develop the various interconnected zones, we calculated the distance from each node to the origin depot. Subsequently, we organized the nodes based on their distance from the depot, in ascending order. Following this, we organize the nodes in ascending order based on their distance from the origin depot. Subsequently, we categorize the customer nodes into three distinct zones according to their proximity to the origin depot: near, central, and distant. The nodes within each zone are treated as interconnected, facilitating a more systematic method for route planning and ensuring the reproducibility of the outcomes. Additionally, the service time for each vehicle was represented using an exponential distribution characterized by a location parameter (γ) and a scale parameter (β). The location parameter was set to 0, while the scale parameter was adjusted to align with various benchmark network configurations.

16.4.1 Case Study: Application to Autonomous Delivery Drones in a University Campus

To create a model of the campus environment, we utilized data sourced from Open Data Barcelona, accessible in an open repository. This dataset includes details on the locations of stations and the distances between them, akin to the paths for delivery points on a campus. We identified a total of 34 nodes, which signify delivery points such as academic buildings, dormitories, libraries, and administrative offices. These nodes were categorized into three separate zones based on their geographical positioning and proximity. Nodes that are situated near the origin depot are designated as "Zone Close", those at a moderate distance from both the origin and destination depots are termed "Zone Medium", and nodes located nearer to the destination depot (for instance, an off-campus distribution center) are classified as "Zone Far". Within each zone, the nodes are linked through common pathways or restricted areas, leading to the requirement that only a single robot can attend to any node in the zone at any given time. The campus delivery network was represented using FlexSim, a commercial DES software selected for its proficiency in accurately modeling discrete events and resource limitations, effectively capturing the system's complexities and random characteristics. To replicate vehicle operations within this environment, we designed vehicles with a capacity of one unit, which corresponds to the precise and compact nature of campus deliveries. Additionally, vehicles can move across the campus at a maximum speed of one unit per time step. The service duration for each node was modeled using an exponential distribution.

Specifically, it is characterized by a location parameter ($\gamma = 0$) and a scale parameter ($\beta = 0.008$). This setup guarantees that the service times remain non-negative and adhere to a plausible variability pattern. In Figure 16.3, the FlexSim simulation environment is depicted, featuring a comprehensive visual representation of the system on the left side. The components

illustrate the spatial arrangement and movement of entities in the simulated environment. On the right, the logical flow created to model the case study is depicted. This logic, realized through FlexSim's "ProcessFlow" module, outlines the order of operations, which encompasses token creation, task distributions, and transitions through queues. It also establishes parameters such as movement between queues, duration of service, and the completion of the simulation process. The visual representation and the logical flow collaborate to emulate the system's behavior, facilitating a realistic and dynamic simulation of the specified case scenario.

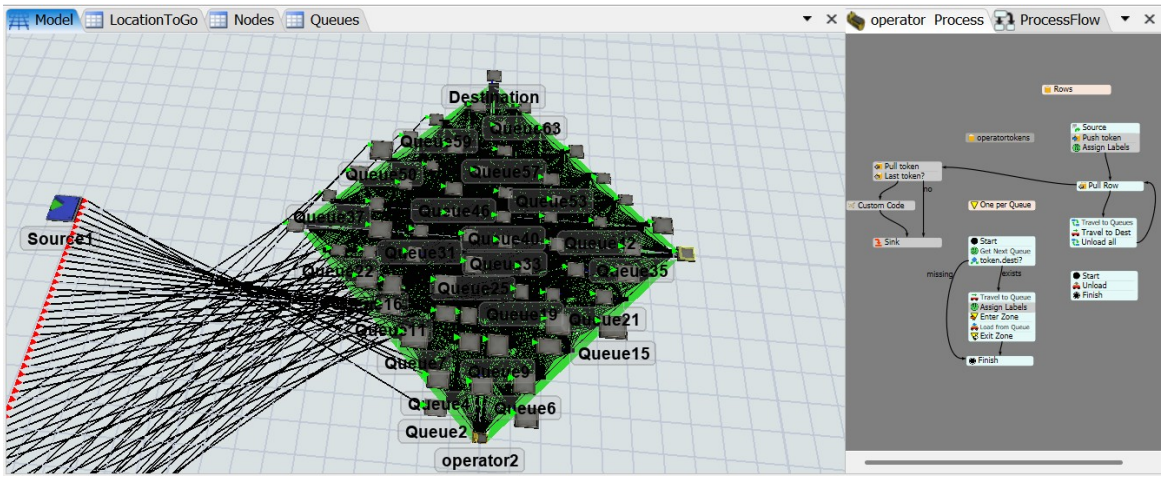


Figure 16.3: FlexSim environment screenshot with system model and ProcessFlow logic.

16.5 Analysis of Results

The computational outcomes for the BR-MS and BR-ILS methods are displayed in Tables 16.1 and 16.2, respectively. The initial column lists the problem instances, whereas the following columns provide the results for both deterministic and stochastic evaluation conditions. The *BKS* column displays the best-known solutions (BKS) for the deterministic version of the problem, as documented in existing literature. The *OBD-R* column shows the total reward accumulated from the best-found solutions within the deterministic scenario. The *OBD-S-R* column reflects the reward collected when the deterministic solutions are assessed in a stochastic environment, thereby illustrating their effectiveness amid uncertainty. Lastly, the *OBS-R* column indicates the reward gathered for the best-found solutions in the stochastic scenario. Finally, the *GAP%* columns calculate the percentage differences of the solutions in relation to the BKS, offering a view into how effective the best solutions are for each scenario.

Upon reviewing Table 16.1, it is evident that the BR-MS method yields high-quality outcomes for the deterministic iteration of the problem. Notably, the *OBD-R* demonstrates an average gap of 9.33% in relation to the BKS. In comparison, the average gap for the *OBS-R* stands at 20.98%,

Table 16.1: Computational results for the BR-MS approach.

Instance	BKS (1)	OBD-R (2)	OBD-S-R (3)	OBS-R (4)	GAP% (1-2)	GAP% (1-3)	GAP% (1-4)
p6.2.k	1032.00	876.00	770.43	821.69	15.12	25.35	20.38
p6.2.l	1116.00	864.00	824.60	835.86	22.58	26.11	25.10
p6.2.n	1260.00	1002.00	793.71	874.30	20.48	37.01	30.61
p6.3.i	642.00	618.00	491.73	501.77	3.74	23.41	21.84
p6.3.j	828.00	816.00	654.50	654.50	1.45	20.95	20.95
p6.3.k	894.00	864.00	704.73	707.00	3.36	21.17	20.92
p6.3.m	1080.00	960.00	793.90	849.45	11.11	26.49	21.35
p6.3.n	1170.00	972.00	797.95	809.97	16.92	31.80	30.77
p6.4.n	1068.00	1020.00	804.04	804.04	4.49	24.72	24.72
p6.4.l	696.00	570.00	503.96	503.96	18.10	27.57	27.57
p6.4.k	528.00	486.00	411.65	420.14	7.95	22.04	20.43
p6.4.j	366.00	360.00	310.65	310.65	1.64	15.12	15.12
p3.2.s	800.00	800.00	695.20	700.15	0.00	13.10	12.48
p3.2.n	660.00	620.00	523.94	533.87	6.06	20.62	19.11
p3.3.t	760.00	700.00	634.33	653.10	7.89	16.54	14.07
p3.3.p	640.00	610.00	523.80	531.86	4.69	18.16	16.90
p3.3.m	520.00	440.00	395.04	403.19	15.38	24.03	22.46
p3.3.n	570.00	560.00	484.31	493.48	1.75	15.03	13.43
p3.3.s	720.00	670.00	591.31	599.09	6.94	17.87	16.79
p3.3.q	680.00	610.00	537.03	537.03	10.29	21.02	21.02
p3.3.r	710.00	710.00	598.93	598.93	0.00	15.64	15.64
p3.4.t	670.00	670.00	599.19	599.19	0.00	10.57	10.57
p3.4.s	670.00	620.00	543.42	543.42	7.46	18.89	18.89
p3.4.r	600.00	520.00	468.48	468.48	13.33	21.92	21.92
Average	789.81	705.75	602.37	614.80	9.33	22.31	20.98

Table 16.2: Computational results for the BR-ILS approach.

Instance	BKS (1)	OBD-R (2)	OBD-S-R (3)	OBS-R (4)	Gap% (1-2)	Gap% (1-3)	Gap% (1-4)
p6.2.k	1032.00	990.00	809.77	822.30	4.07	21.53	20.32
p6.2.l	1116.00	1050.00	857.52	881.70	5.91	23.16	20.99
p6.2.n	1260.00	1074.00	911.36	929.18	14.76	27.67	26.26
p6.3.i	642.00	636.00	505.57	508.17	0.93	21.25	20.85
p6.3.j	828.00	816.00	652.35	661.83	1.45	21.21	20.07
p6.3.k	894.00	864.00	704.20	721.36	3.36	21.23	19.31
p6.3.m	1080.00	1038.00	859.56	876.77	3.89	20.41	18.82
p6.3.n	1170.00	1080.00	884.54	902.10	7.69	24.40	22.90
p6.4.n	1068.00	1062.00	797.93	808.34	0.56	25.29	24.31
p6.4.l	969.00	648.00	528.99	539.17	33.13	45.41	44.36
p6.4.k	528.00	504.00	430.06	433.46	4.55	18.55	17.90
p6.4.j	366.00	366.00	314.26	315.75	0.00	14.14	13.73
p3.2.s	800.00	800.00	700.42	711.19	0.00	12.45	11.10
p3.2.n	660.00	620.00	542.90	557.90	6.06	17.74	15.47
p3.3.t	760.00	700.00	660.61	680.58	7.89	13.08	10.45
p3.3.p	640.00	610.00	548.08	561.12	4.69	14.36	12.32
p3.3.m	520.00	480.00	421.97	438.66	7.69	18.85	15.64
p3.3.n	570.00	570.00	505.61	507.93	0.00	11.30	10.89
p3.3.s	720.00	680.00	611.03	643.50	5.56	15.13	10.63
p3.3.q	680.00	630.00	559.17	577.03	7.35	17.77	15.14
p3.3.r	710.00	710.00	610.82	619.62	0.00	13.97	12.73
p3.4.t	670.00	670.00	609.65	625.07	0.00	9.01	6.71
p3.4.s	670.00	670.00	580.53	590.34	0.00	13.35	11.89
p3.4.r	600.00	570.00	498.02	516.04	5.00	17.00	13.99
Average	789.71	743.25	629.37	642.88	5.19	19.09	17.37

consistently remaining closer to the optimal BKS than the *OBD-S-R*, which has an average gap of 22.31%. This indicates that while the best solutions identified in the deterministic context are efficient, they become suboptimal when uncertainty is factored in, as evidenced by the lower reward obtained from the *OBD-S-R* compared to that of the *OBS-R*. Upon reviewing Table 16.2, it is observed that the BR-ILS method yields high-quality solutions for the deterministic variant of the problem. Notably, it consistently exhibits superior percentage gaps compared to the BR-MS method, achieving an average gap of 5.19% in relation to the BKS. Furthermore, the average gap for the *OBS-R* is 17.37%, which remains closer to the optimal BKS than the *OBD-S-R*, which has an average gap of 19.09%. Both average gaps exceed those produced by the BR-MS method. This highlights the benefits of employing a simheuristic approach for effectively addressing uncertainty, as well as the significance of simulation in assessing viable solutions during the quest for optimal outcomes in uncertain conditions.

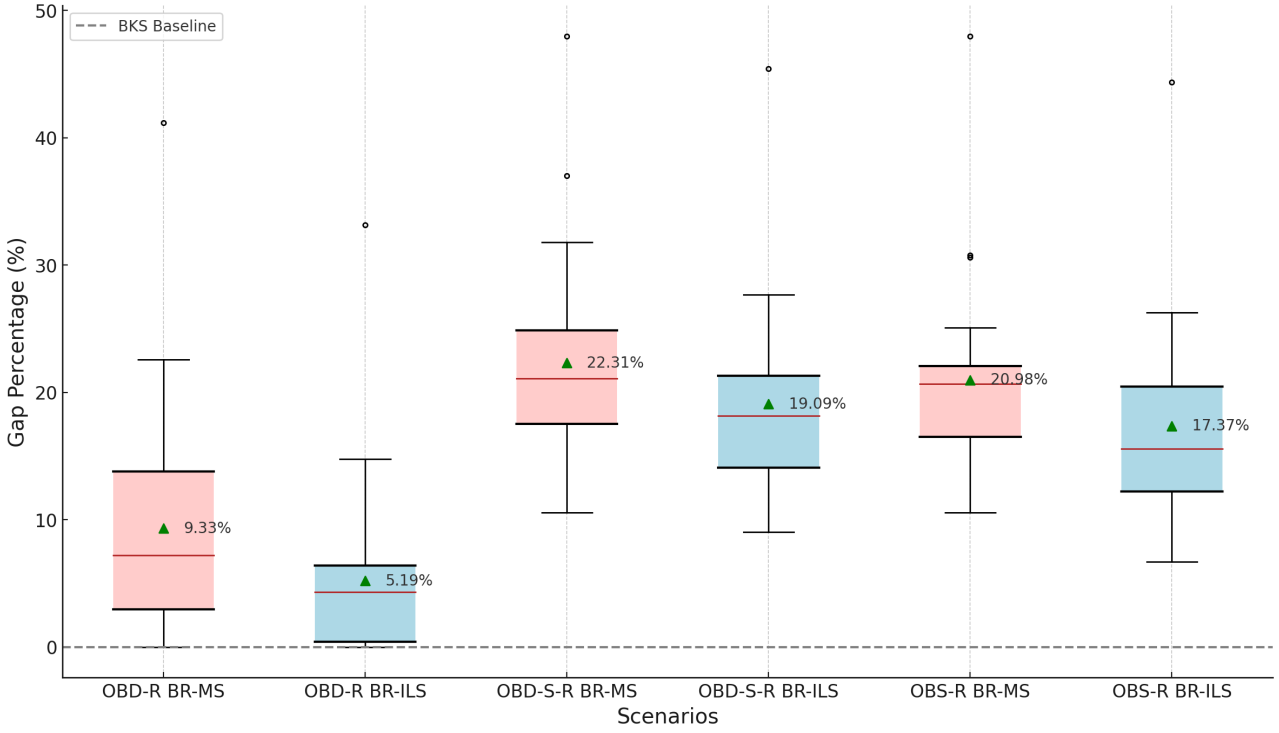


Figure 16.4: Boxplot comparison for the obtained results.

Figure 16.4 presents a summary of the results derived from Tables 16.1 to 16.2. In the boxplot, the horizontal axis illustrates the solutions produced by the BR-MS and BR-ILS methods across various scenarios, while the vertical axis depicts the percentage differences compared to the BKS. Observe that in the deterministic scenario, the BR-ILS method attains a notably lower average gap percentage of 5.19%, in contrast to 9.33% for BR-MS. This suggests that BR-ILS is more proficient in generating deterministic solutions that consistently approach

the BKS. Additionally, the variation in performance gaps for BR-ILS in this context is reduced, indicating a more uniform performance across different problem instances. When assessing these solutions within a stochastic context (*OBD-S-R*), the performance discrepancies between the two methods become more pronounced. Nevertheless, BR-ILS consistently demonstrates superior performance over BR-MS, exhibiting an average gap percentage of 19.09% in contrast to 22.31%. In terms of the outcomes derived from the simheuristic framework (*OBS-R*), BR-ILS once again registers a lower average gap percentage of 17.37%, compared to 20.98% for BR-MS. This indicates that BR-ILS possesses enhanced robustness in adjusting to stochastic environments. Notably, BR-ILS shows reduced variability in its performance gaps across all scenarios when compared to BR-MS, resulting in more stable and dependable solutions across various problem instances.

In the case study, Tables 16.3 and 16.4 display the outcomes of the BR-MS and BR-ILS methods across three different instances involving 2, 3, or 4 vehicles. Each table includes results for both deterministic and stochastic scenarios, along with the percentage gaps (*Gap%*) indicating the difference between the optimal solutions identified in a deterministic context when assessed in a stochastic environment (*OBD-S-R*) and the optimal solutions found in the stochastic scenario (*OBS-S-R*).

Table 16.3: Case study computational results for the BR-MS approach.

Instance	**OBD-D-R**	**OBD-S-R (1)**	**OBS-S-R (2)**	**Gap% (1-2)**
BCN-2v	419.00	274.24	288.03	-5.03
BCN-3v	401.00	220.52	243.30	-10.33
BCN-4v	357.00	91.96	133.18	-44.82
Average	392.33	195.57	221.50	-20.06

Table 16.4: Computational results of the case study for BR-ILS approach.

Instance	**OBD-D-R**	**OBD-S-R (1)**	**OBS-S-R (2)**	**Gap% (1-2)**
BCN-2v	442.00	190.83	291.57	-52.79
BCN-3v	414.00	164.34	247.81	-50.79
BCN-4v	359.00	58.61	196.16	-234.70
Average	405.00	137.93	245.18	-112.76

If we examine Table 16.3, it is evident that the best solutions in the deterministic scenario yield rewards of 419.00, 401.00, and 357.00 across various instances. In contrast, when these solutions are assessed in stochastic conditions, there is a noticeable decline in the rewards, which fall to 274.24, 220.52, and 91.96. Nevertheless, the top solutions for the stochastic

scenario (*OBS-S-R*) consistently produce superior rewards of 288.03, 243.30, and 133.18 for the corresponding instances. If we examine Table 16.4, it reveals that the top deterministic solutions from the BR-ILS yield higher rewards of 442.00, 414.00, and 359.00 in comparison to the BR-MS method. When assessed under stochastic conditions, the performance of the deterministic solutions (*OBD-S-R*) declines to 190.83, 164.34, and 58.61. In contrast, the leading stochastic solutions (*OBS-S-R*) exhibit a notable enhancement, achieving rewards of 291.57, 247.81, and 196.16, respectively. This underscores the value of utilizing simulation as a means to enhance robustness and adaptability in uncertain real-world situations.

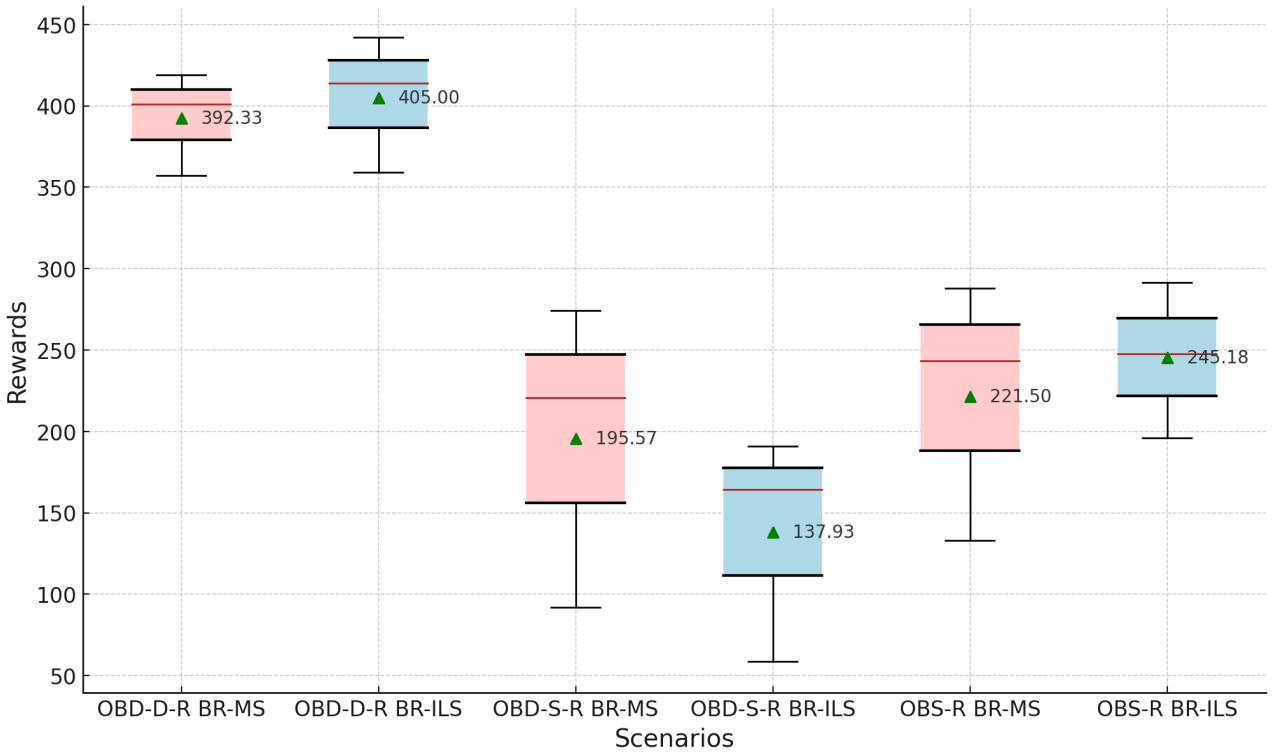


Figure 16.5: Boxplot comparison for the collected Reward results.

Figure 16.5 illustrates a comparative assessment of the rewards achieved by the BR-MS and BR-ILS methods across various scenarios presented in Tables 16.3 through 16.4. In the boxplot, the horizontal axis represents the solutions produced by the BR-MS and BR-ILS methods, while the vertical axis indicates the corresponding rewards earned. It is evident that the BR-ILS method secures greater rewards in the deterministic scenario; however, it incurs higher penalties in uncertain conditions. In the stochastic scenario, both methods demonstrate significant enhancements in rewards when compared to the *OBD-S-R*. The BR-ILS method reliably yields greater benefits in various instances within stochastic environments. This highlights the effectiveness of the BR-ILS strategy in managing uncertainty and optimizing under stochastic scenarios. Additionally, this chapter introduces a simheuristic framework designed to

tackle a specific variant of the TOP that incorporates the challenges of urban vehicle routing. To our knowledge, prior research has primarily concentrated on the deterministic TOP or on stochastic variants employing MCS to represent the uncertainties present in real-world situations. The suggested simheuristic approach, which integrates BR-MS or BR-ILS techniques with DES, has proven to effectively generate high-quality solutions in an efficient manner. Notably, the BR-ILS method surpasses the BR-MS approach, yielding solutions that achieve greater collected rewards in both deterministic and stochastic scenarios. Moreover, FlexSim is pivotal as it precisely simulates the intricate and variable characteristics of urban settings, offering a realistic framework for solution assessment.

Part V

Sim-Learnheuristics

Chapter 17

Sim-Learnheuristic Framework

This chapter introduces the concept of sim-learnheuristic, an innovative approach that combines simulation with the learnheuristics framework to address complex, hybrid scenarios involving uncertainty and dynamic conditions. Learnheuristics, which integrates ML techniques with heuristic optimization, has been widely recognized for its ability to enhance problem-solving in dynamic decision-making environments by learning from historical data and adapting the heuristic search process. However, in many real-world applications, uncertainty and rapidly changing conditions require more robust methods. Sim-learnheuristic builds on the foundation of learnheuristics by incorporating simulation, making it more effective in situations where traditional deterministic methods fall short. This chapter provides a detailed overview of the sim-learnheuristic methodology, explaining how simulation enhances the adaptability of heuristic optimization in uncertain environments. All of our published research has contributed to the development of this framework.

The combination of ML and metaheuristics in literature is on the rise, showcasing a diverse array of applications. This trend indicates a significant shift towards incorporating AI technologies into diverse areas. The combination of ML algorithms and metaheuristic approaches are now more commonly used to address intricate challenges across various sectors such as healthcare, finance, transportation, and beyond. Typically, this combination can be categorized into two approaches (Calvet et al., 2017): using metaheuristics to enhance ML and utilizing ML to improve metaheuristics. In the former approach, applications span across various tasks such as classification (including feature selection, feature extraction, and parameter fine-tuning), regression (for complex regression analysis), clustering, and rule mining. The latter approach involves both local-level and high-level hybridizations. Local-level hybridizations encompass fine-tuning metaheuristic parameters, initialization, evaluation, population management, operators, and local search. Global hybridizations, on the other hand, focus on reducing search space, algorithm selection, employing hyperheuristics, cooperative strategies, or developing novel types

of metaheuristics. In contrast to these approaches, the learnheuristic framework integrates both ML and metaheuristics, offering a solution tailored to address specific optimization problems. The learnheuristic framework is designed to address optimization problems where the inputs to the model, whether embedded within the objective function or the set of constraints, are not predetermined. Rather, these inputs, such as customers' demands or serving times, may fluctuate predictably based on the current state of the partially constructed solution at each iteration of the constructive heuristic. In order to address optimization problems with dynamic inputs, the development of a learnheuristic framework is proposed. Figure Figure 17.1 illustrates the fundamental outline of this approach. Initially, historical data pertaining to diverse system states (such as various user-to-radio access technology assignments) and their associated inputs (like observed user demands corresponding to these assignments) are utilized to develop ML predictive models (such as regression or neural network models). Subsequently, these predictive models are iteratively integrated into the heuristic-based constructive process to continually update estimations of problem inputs (such as user demands) as the solution structure evolves (for instance, changes in the user-to-radio access technology assignment map). Ultimately, upon completion of the construction process, a comprehensive solution is generated. The incorporation of the learning mechanism is crucial, as without it, the heuristic-based construction process would neglect input variations resulting from changes in the solution structure, ultimately leading to suboptimal solutions.

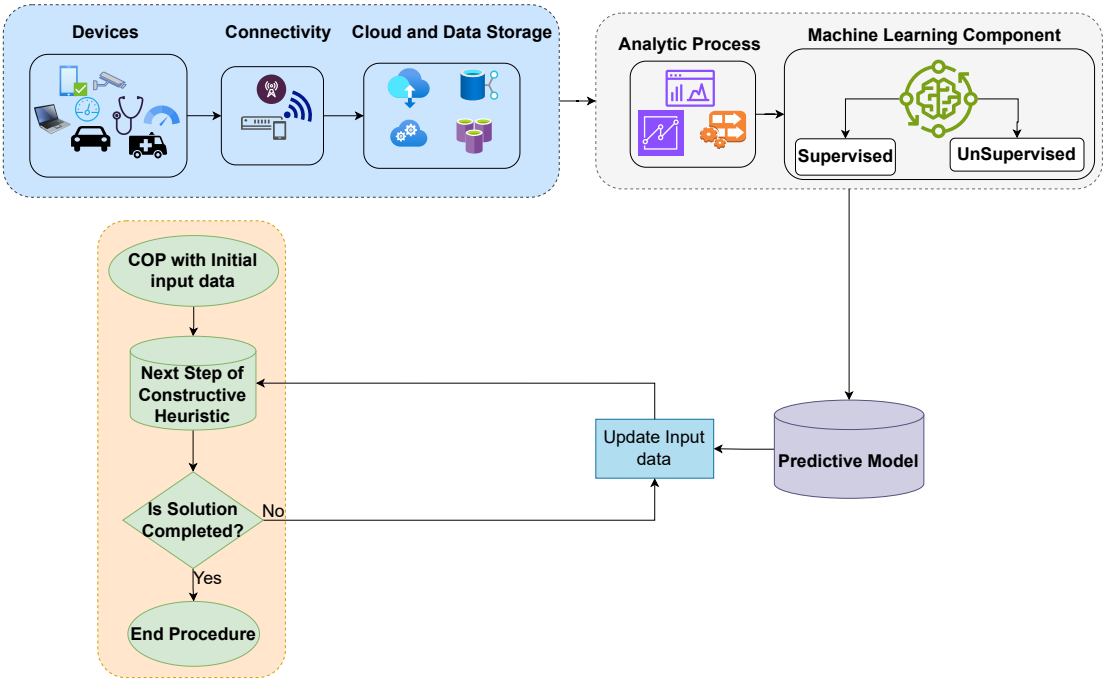


Figure 17.1: Basic scheme of a learnheuristic framework.

Source:Calvet et al. (2017)

This fundamental approach can be expanded in various manners, such as: (i) utilizing an online approach, where new inputs continuously update and enhance the predictive model; (ii) employing a reinforcement learning model, learning from interactions with the environment rather than relying solely on historical data; (iii) implementing an "assembled" or blending approach, constructing multiple models, and combining them to produce input estimates; and (iv) integrating it into a more intricate metaheuristic framework. Learnheuristics find applications across various domains, including transportation (Peyman, Fluechter, et al., 2023), and health (Bayliss and Panadero, 2024). Although the integration of simheuristics and learnheuristics is still in its early stages, it holds promise for developing efficient algorithms to tackle practical optimization problems in stochastic and dynamic environments (Reyes-Rubiano et al., 2020).

The sim-learnheuristic methodology extends this framework by integrating simulation, making it more adaptable to real-world problems that involve dynamic uncertainties. Traditional approaches assume problem parameters are either deterministic or governed by static stochastic processes. However, real-world scenarios, such as machine breakdowns in manufacturing or supply chain disruptions in logistics, often involve more complex uncertainties that static models cannot manage effectively. Addressing these evolving uncertainties requires methods capable of adapting in real time.

Figure 17.2 illustrates the fundamental elements of the sim-learnheuristic approach. This approach is organized into six main stages, labeled S1 to S6. The first step involves converting the optimization problem into a deterministic form by substituting the variables with their expected or most probable values, effectively eliminating uncertainty. Following this, the deterministic problem is tackled through a BR-MS framework, leading to the development of several high-quality solutions. These deterministic solutions are thoroughly assessed, considering the multiple sources of uncertainty that are intrinsic to the problem. This assessment entails conducting a restricted number of MCS runs, where random variables and dynamic components are assigned varying values based on their specific probability distributions or ML models. Additionally, constraints are analyzed under uncertain circumstances, and descriptive statistics are generated for each solution, yielding comprehensive insights into their effectiveness. The examination of these solutions informs the development of additional solutions within the BR-MS framework, progressing until a specified stopping condition, like a maximum allowable execution time, is met. In the following phase, a selection of the top-performing solutions, referred to as "elite" solutions, is subjected to a more in-depth assessment under uncertain conditions. This secondary evaluation consists of a higher number of MCS runs than those conducted during the initial evaluations. Based on this comprehensive assessment, the solutions are prioritized, and the highest-performing option is suggested. The choice of the most suitable solution may also consider supplementary factors of interest, such as variability or dependability, in addition to the anticipated value, to enhance the decision-making process. This innovative methodology integrates key components (simulation, ML, and optimization) to model scenarios that encompass both stochastic and dynamic elements. By combining these approaches, it enhances decision-making and operational efficiency across diverse conditions, making it especially effective in tackling real-world complexities.

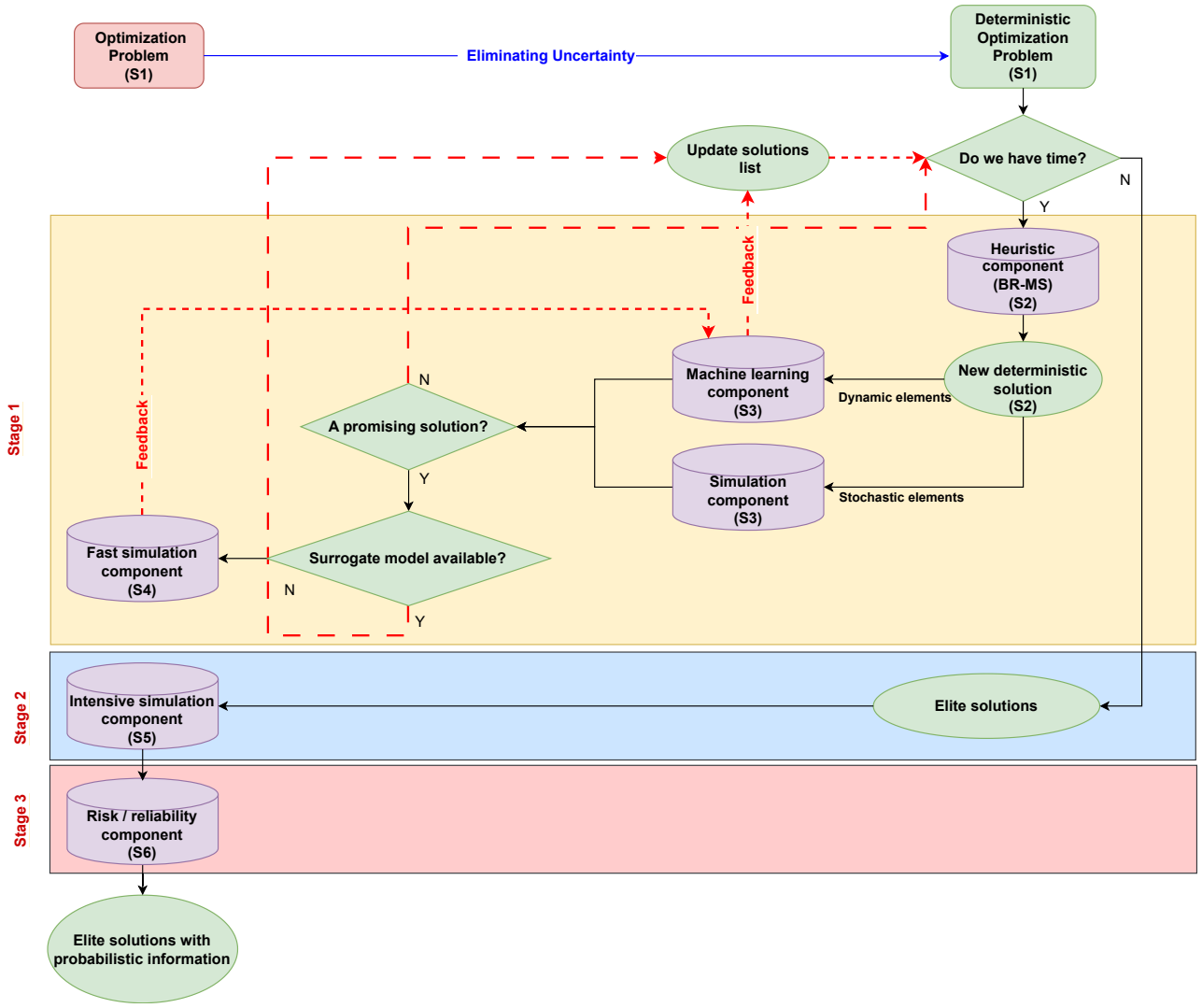


Figure 17.2: Schema of the sim-learnheuristic methodology.

Chapter 18

Sim-Learnheuristic for Team Orienteering Problem

The simheuristics and combinatory methods described in the previous part (Part IV), which integrate DES with optimization algorithms, generally assume that the problem's parameters are either deterministic or stochastic. However, in real-world scenarios, this is often not the case due to dynamic factors such as weather conditions, traffic patterns, or, in the case of EVs, battery capacity. To address these complexities, this chapter[1] use the introduced a novel sim-learnheuristic approach in Chapter 17 to solve the TOP, specifically focusing on its application with UAVs.

[1]The contents of this chapter are based on the following work:

- **Peyman, Mohammad**, Xabier A Martin, Javier Panadero, and Angel A Juan (2024). *"A sim-learnheuristic for the team orienteering problem: Applications to unmanned aerial vehicles"*. In: Algorithms 17.5, p. 200.

18.1 Problem Description

The TOP is a well-established optimization challenge discussed in Section 3. This problem is particularly important in fields like search and rescue, surveillance, and logistics. For instance, in a search and rescue operation, a fleet of UAVs may need to traverse a disaster-affected region to find survivors and supply necessities. UAVs have proven to be invaluable in these scenarios because of their ability to undertake perilous tasks and their proficiency in sensing, monitoring, and navigating. However, the challenges they encounter are significant (Gupta et al., 2021). These challenges are complex and encompass various unpredictable environmental elements, including weather and traffic situations. This unpredictability hinders accurate predictions of behavior, leading to less effective solutions and diminished mission effectiveness. Furthermore, the inherent constraints of UAVs, such as their restricted battery duration and payload capacity, necessitate meticulous optimization for successful mission execution. Nonetheless, UAVs continue to be an essential technology for addressing intricate issues across multiple fields, highlighting the significance of the TOP and its relevance to UAVs (Bayliss, Juan, et al., 2020).

Furthermore, incorporating IoT technologies is considered one of the most promising strategies for allowing UAVs to perceive their environment and gather data. This incorporation enables UAVs to collect and share real-time environmental information, thus enhancing their route optimization (Poudel et al., 2021). In recent years, a significant number of studies have concentrated on either deterministic or stochastic variations of the TOP. For deterministic TOP, many existing studies heavily depend on precise methodologies to address the issue (Gunawan et al., 2016). However, when faced with challenges arising from large-scale situations, these particular methods demonstrate restricted efficacy. Additionally, various adaptations of the TOP have been created to tackle specific real-world issues, such as differing vehicle capacities, customer time limitations, and variable travel durations and profits.

This section considers a hybrid variant of the TOP that integrates deterministic, stochastic, and dynamic features, thereby filling a gap identified in existing academic research. To tackle this intricate issue, a new sim-learnheuristic approach has been proposed, particularly applicable to UAVs for disaster rescue operations. This method merges a BR savings-based heuristic, MCS, and a multiple regression model. The primary aim of this innovative methodology is to use the advantages of all three elements, facilitating a nimble exploration of the solution space to yield high-quality outcomes for the problem at hand. Figure 18.1 illustrates a novel version of the TOP, showcasing a diverse array of travel time characteristics. These include fixed travel times (deterministic), variable travel times (stochastic), and travel times that fluctuate over time due to dynamic influences (dynamic), such as changes in traffic congestion and weather patterns. This combination of attributes leads to a more accurate depiction of the complexities found in

real-world scenarios.

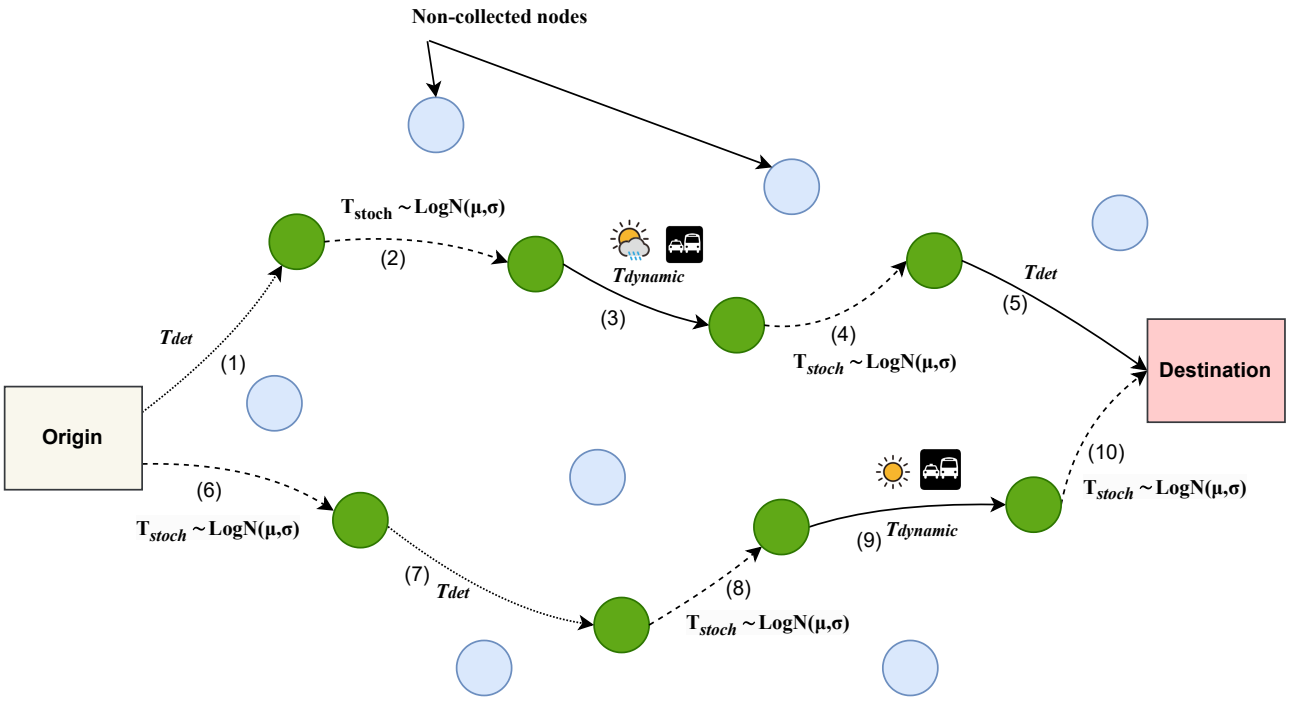


Figure 18.1: Hybrid variant of the TOP considering different kinds of travel times.

The primary contributions of this section are outlined as follows: *(i)* a hybrid version of the TOP is presented, which includes deterministic, stochastic, and variable travel times affected by elements like traffic congestion and weather; and *(ii)* a new approach, termed the sim-learnheuristic algorithm, is introduced to solve this advanced variant of the TOP, utilizing the combination of three different components to address more complex combinatorial optimization challenges. Figure 18.2 illustrates how the three components are interconnected. The process initiates with the metaheuristic component, which aims to tackle the deterministic features of the issue at hand. Nonetheless, real-world situations frequently present uncertainties. To manage this, the simulation component represents the random characteristics of the environment. The combination of the simulation and metaheuristic components allows the simheuristic component to proficiently address stochastic challenges. Additionally, since real-world conditions are constantly changing, ML algorithms are employed to forecast and adjust to these variations. By integrating these forecasts into the simheuristic framework, the approach can proficiently tackle and resolve intricate dynamic challenges.

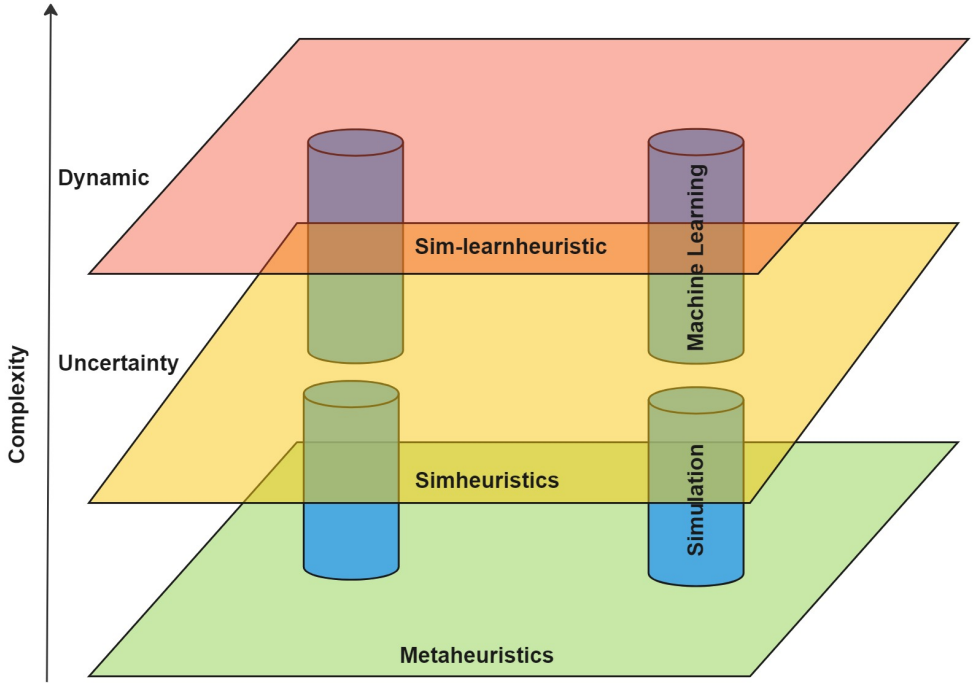


Figure 18.2: Integration of the sim-learnheuristic's main three components.

18.2 Problem Modeling

The TOP is classified as an NP-hard optimization challenge Chao et al., 1996, defined formally on a directed graph $G = (V, A)$. This graph consists of $V = \{0, 1, 2, \ldots, n+1\}$ nodes, where node 0 serves as the origin depot, node $n+1$ functions as the destination depot, and the intermediate nodes are represented by $N = \{1, \ldots, n\}$. The set of edges $A = \{(i, j)|i, j \in V, i \neq j\}$ denotes the connections established between the nodes. We examine a group of homogeneous UAVs labeled as D, where each UAV $d \in D$ commences its route from the main depot, services a selection of intermediate nodes, and ultimately arrives at the destination depot. The goal is to optimize the overall reward gathered by all UAVs. Following the mixed-integer linear programming model for the TOP introduced by Panadero, Juan, Ghorbani, et al. (2023), the hybrid variant of the problem, which encompasses travel times that may be deterministic, stochastic, or dynamic, can be articulated formally as:

$$\max \sum_{d \in D} \sum_{(i,j) \in A} u_j x_{ij}^d \tag{18.1}$$

The aim is to optimize the total deterministic reward obtained from the nodes visited along the specified edges. Each node encountered during the journey generates a reward denoted as $u_i \geq 0$. It is important to highlight that the rewards assigned to both the starting and ending depots are zero, while the rewards for the customer nodes are strictly positive. For every edge $(i, j) \in A$ and for each UAV $d \in D$, we define the binary variable $x^d_{ij},$. This variable is set to 1 if UAV d travels along edge (i, j) and is 0 in all other cases.

Below are the provided constraints with their explanations. In the course of the tour, each point is visited at most once to ensure that no point is revisited (18.2).

$$\sum_{d \in D} \sum_{i \in V} x^d_{ij} \leq 1 \quad \forall j \in N \tag{18.2}$$

The variable y^d_i is introduced to indicate the position of node i in the tour made by vehicle d, and constraint (18.3) makes sure that there are no sub-tours.

$$y^d_i - y^d_j + 1 \leq (1 - x^d_{ij}) \mid N \mid \quad \forall i, j \in N, \ \forall d \in D \tag{18.3}$$

The constraint outlined in equation (18.4) specifies that the total travel duration for each vehicle must remain within its designated limit (T_{max}). This is important because each UAV, denoted as $d \in D$, initiates its journey and is restricted to servicing only certain intermediate nodes owing to the constraints on travel time. The overall travel duration is calculated by adding together the travel times associated with deterministic, dynamic, and stochastic edges, represented by A_{det}, A_{dyn}, and A_{stoch}, correspondingly. These are separate groups that encompass all the edges of set A. For every edge $(i, j) \in A_{det}$, we consider that the travel time for each edge is fixed and known in advance ($t_{ij} = t_{ji} > 0$). In contrast, for each edge $(i, j) \in A_{stoch}$, we assume that the travel time T_{ij} is variable and can be represented through a probability distribution function. Lastly, for every edge $(i, j) \in A_{dyn}$, we consider the travel time $\hat{t}_{ij}$ to be dynamic, influenced by multiple factors including weather and traffic conditions. The inherent uncertainty of stochastic and dynamic travel times necessitates a probabilistic framework to manage this unpredictability. The parameter γ, which varies from 0 to 1, signifies a threshold that delineates the acceptable probability level for maintaining the travel time within the established limits.

$$P\left(\sum_{(i,j) \in A_{det}} t_{ij} x^d_{ij} + \sum_{(i,j) \in A_{dyn}} \hat{t}_{ij} x^d_{ij} + \sum_{(i,j) \in A_{stoch}} T_{ij} x^d_{ij} \leq T_{max}\right) \geq \gamma \quad \forall d \in D \tag{18.4}$$

Constraint (18.5) ensures that for every arrival at a node, there is a corresponding departure

from that node.

$$\sum_{i \in V} x_{ij}^{d} = \sum_{h \in V} x_{jh}^{d} \quad \forall d \in D \qquad \forall j \in N \tag{18.5}$$

Constraint (18.6) indicates that the tour always commences at the starting node 0 and ends at the ending node $n+1$.

$$\sum_{j \in N} x_{0jd} = \sum_{j \in N} x_{j(n+1)d} = 1 \qquad \forall d \in D \tag{18.6}$$

Constraints (18.7) and (18.8) refer to the nature of y_j^d and x_{ij}^d variables. Finally, the vehicle departs from each node it visited, except for the end node.

$$y_j^d \geq 0 \quad \forall j \in N \qquad \forall d \in D \tag{18.7}$$

$$x_{ij}^d \in \{0,1\} \quad \forall i,j \in A, \forall d \in D \tag{18.8}$$

Although the deterministic approach to the TOP has received considerable attention in existing research, the significant uncertainty present in practical applications of the TOP renders it suitable for our objectives. In this study, we incorporate both stochastic and dynamic travel times into the TOP. Hence, we will assume that the travel times for certain edges can be represented as random variables adhering to a theoretical probability distribution. Similarly, we will take into account that the travel times for other edges are uncertain and need to be estimated using a multiple regression model.

18.3 Methodological Approach

To address the hybrid TOP effectively, we introduce a new sim-learnheuristic approach that integrates a BR-MS framework, which is thoroughly detailed in Chapter 16, alongside MCS and a multiple regression model. This approach is designed to use the strengths of all components, facilitating a more efficient exploration of the search space to identify optimal solutions (Martí et al., 2013). Furthermore, this sim-learnheuristic methodology builds upon the simheuristic strategy outlined in Panadero, Juan, Bayliss, et al. (2020) to tackle the TOP with stochastic processing times. Figure 17.2 illustrates the key components of the sim-learnheuristic methodology. Based on the detailed explanation provided in the previous chapter (Chapter 17), Algorithm 17 details the key phases of the sim-learnheuristic method. This algorithm takes an instance as input, which includes the nodes, the upper limit on the number of vehicles, and the maximum allowed travel duration. Initially, a practical starting solution ($initSol$) is created using a savings-oriented heuristic (line 1). Then, a limited number of MCS runs (n_{short}) are conducted

to evaluate the anticipated reward of the initial solution (line 2). At this stage, the initial solution is recognized as the best-known solution to date ($bestSol$), and the collection of top stochastic solutions ($eliteSols$) is established with this solution (lines 3–4). Next, the algorithm engages in a repetitive process to create new solutions until it hits a predetermined maximum execution time (lines 6–16). During each iteration, a new solution ($newSol$) is produced through a BR variant of the savings-based heuristic (line 7). The BR heuristic incorporates a minor adjustment to the greedy approach, introducing an element of randomness while still adhering to the principles of the savings-based heuristic. The BR version considers each item in the edges list based on a probability that adheres to a geometric distribution characterized by a single parameter $\beta \in (0, 1)$, which determines the degree of greediness in the algorithm's randomized actions. By utilizing a BR variant of the savings-based heuristic, it is possible to produce various alternative solutions while preserving the fundamental reasoning of the original heuristic. Subsequently, the algorithm evaluates the newly created solution against the best-known solution. If the new solution offers a greater reward value (line 8), a short number of MCS runs are performed on the new solution to calculate its expected reward (line 9). In addition, if the newly generated solution obtains a higher expected reward (line 10), the best-known solution is updated and the new solution is added to the pool of elite solutions (lines 11–12). The computational time employed to generate new solutions is updated before checking if the maximum execution time (t_{max}) is reached (line 15). Once the stopping criteria are met, the algorithm conducts a refinement procedure using a larger number of MCS runs (n_{long}) over the pool of elite solutions (lines 17–22). This allows for more accurate estimations of the solutions in the elite pool. Finally, the solution with the highest expected reward is selected from the pool of elite solutions and returned by the algorithm (line 23).

Algorithm 17 Sim-learnheuristic algorithm

```
 1: initSol ← savingsHeuristic(inputs)
 2: simulation(initSol, n_short)
 3: bestSol ← initSol
 4: add(eliteSols, bestSol)
 5: time ← 0
 6: while time ≤ t_max do
 7:    newSol ← savingsHeuristicBR(inputs, β)
 8:    if reward(newSol) > reward(bestSol) then
 9:       simulation(newSol, n_short)
10:       if expReward(newSol) > expReward(bestSol) then
11:          bestSol ← newSol
12:          add(eliteSols, bestSol)
13:       end if
14:    end if
15:    updateTime(time)
16: end while
17: for sol ∈ eliteSols do
18:    simulation(sol, n_long)
19:    if expReward(sol) > expReward(bestSol) then
20:       bestSol ← sol
21:    end if
22: end for
23: return bestSol
```

The key components of the savings-based heuristic are explained in detail in Chapter 16. The simulation process is outlined in Algorithm 18. This process necessitates two input parameters: a solution (sol) and the number of simulation replications (n_{runs}). These parameters are essential for assessing the solution's performance in stochastic and dynamic environments. During the procedure, a variable is employed to monitor the total expected reward ($sumReward$), which is set to zero at the start of the simulation (line 1). The simulation is executed it_{max} times, and during each run, the expected reward of the solution ($solReward$) is added to the accumulated expected reward ($sumReward$) (lines 2–28). For every route within the solution, the anticipated reward of that route ($routeReward$) is determined, as well as its projected cost ($routeCost$) (lines 4–26). If the anticipated cost of the route exceeds the maximum permitted travel time, a penalty will be applied to the route, resulting in the loss of all rewards gained from the nodes visited. (lines 22–24). To determine the anticipated reward for a route, we examine each of its individual edges, calculating the expected cost associated with every edge (lines 7–21). The anticipated expense (expected cost) of an edge is established according to its deterministic, stochastic, or dynamic characteristics. For a deterministic edge, the expected cost is predetermined and tailored to each specific case. (line 12). Additionally, the expected cost of a stochastic edge is drawn from a non-negative probability distribution, like the Weibull or Log-Normal distributions, to address the uncertainty inherent in the route planning process (line 14). Finally, the expected cost of a dynamic edge is estimated using a multiple regression model that accounts for the current dynamic conditions at the given time (lines 16–17). Once

all of the simulation replications have been finalized, the expected reward is determined by calculating the average of the accumulated expected rewards from each replication, which is then designated as the expected reward for the solution (lines 29–30).

Algorithm 18 Simulation procedure

```
1: sumReward ← 0
2: for i ∈ {1, ..., n_runs} do
3:     solReward ← 0
4:     for route ∈ getRoutes(sol) do
5:         routeReward ← 0
6:         routeCost ← 0
7:         for edge ∈ getEdges(route) do
8:             edgeCost ← 0
9:             node ← getEndNode(edge)
10:            type ← getType(edge)
11:            if type = DETERMINISTIC then
12:                edgeCost ← getCost(edge)
13:            else if type = STOCHASTIC then
14:                edgeCost ← getStochasticValue(edge, varLevel)
15:            else if type = DYNAMIC then
16:                conditions ← getDynamicConditions(time)
17:                edgeCost ← getDynamicValue(edge, conditions)
18:            end if
19:            routeCost ← routeCost + edgeCost
20:            routeReward ← routeReward+ getReward(node)
21:        end for
22:        if routeCost > routeMaxCost then
23:            routeReward ← 0
24:        end if
25:        solReward ← solReward + routeReward
26:    end for
27:    sumReward ← sumReward + solReward
28: end for
29: expReward ← sumReward/n_runs
30: setExpReward(sol, expReward)
```

18.4 Computational Experiments

The sim-learnheuristic algorithm was implemented in Julia 1.8.2 and tested on a personal computer featuring an Intel Core i7 processor (2.8 GHz) and 16 GB of RAM. A computational time limit of 60 seconds was applied to all instances. The β parameter for the geometric distribution was randomly drawn from the range $(0.1, 0.3)$, based on a brief tuning process conducted on a random sample of instances, which indicated strong performance. The simulation phase included 100 runs for the exploratory stage and 1000 runs for the refinement stage. To validate the proposed methodology, 23 instances were randomly selected from a well-known benchmark presented below, and 10 independent executions were performed for each instance using different initial seeds for the algorithm. From these runs, the best solutions, determined

by the highest collected reward, were reported.

Due to the lack of publicly available instances for the dynamic and stochastic TOP, we opted to extend the benchmark instances introduced by Chao et al. (1996) for the deterministic version of the TOP. This benchmark collection comprises 320 examples, organized into seven distinct sets. Each instance within a set follows the naming convention px.y.z, where x represents the set number, y specifies the number of drones (ranging from two to four, depending on the instance), and z indicates the maximum driving range. These benchmark instances have been extensively utilized in previous studies to evaluate the performance of algorithms for solving the deterministic TOP Tang and Miller-Hooks (2005), Ke et al. (2008), and Dang et al. (2013). However, in this study, we extended these instances to include four distinct scenarios: deterministic, stochastic, dynamic, and hybrid.

In a deterministic context, the travel duration for every route is predetermined and is determined by the Euclidean distance between each node pair in the benchmark cases.

In the stochastic scenario, we incorporate variability into the travel times between nodes. Each edge $(i, j) \in A$ in the directed graph $G = (V, A)$ is now defined by a travel time, $T_{ij} = T_{ji} > 0$, which is not deterministic but follows a best-fit probability distribution function with a mean value $E[t_{ij}] > 0$. In our computational experiments, the Log-Normal probability distribution was used to model random travel times. The Log-Normal distribution is favored over the Normal distribution when representing non-negative random variables. The Log-Normal distribution is defined by two parameters: the location parameter μ and the scale parameter σ. These parameters can be determined based on the properties of the Log-Normal distribution, with the stochastic travel times between nodes i and j assumed to follow: $E[T_{ij}] = t_{ij}$ (i.e., the travel costs of the deterministic instances), and $\text{Var}[T_{ij}] = c \cdot t_{ij}$ for all $i, j \in \{0, 1, 2, \ldots, n + 1\}$. The parameter c functions as a design parameter, allowing for control over the level of uncertainty. As c approaches zero, the results of the stochastic scenario are expected to converge with those of the deterministic scenario. In this analysis, we set $c = 1$, introducing a substantial degree of uncertainty by using the deterministic travel time as the variance. For a more detailed discussion on the effects of varying the parameter c, refer to Panadero, Juan, Ghorbani, et al. (2023). Specifically, the authors define three levels of uncertainty: low ($c = 0.05$), medium ($c = 0.25$), and high ($c = 0.75$). Furthermore, in the stochastic scenario, edges are categorized into two groups based on the following condition: if the node index is even, the corresponding edge is classified as stochastic; otherwise, it is classified as deterministic.

In a dynamic context, travel durations between points are forecasted utilizing a multiple regression model that takes into account the prevailing conditions. Rather than depending on static or predetermined travel times, these durations are calculated using real-time or current information accessible during the route planning process. This multiple regression

model incorporates external factors, including weather and congestion conditions, to calculate the dynamic cost. The levels of weather adversity, denoted as w, and congestion adversity, represented by c, both vary from 0 (indicating low adversity) to 1 (indicating high adversity), serving as input variables for the regression analysis. These parameters are sampled from a uniform distribution, suggesting that both extreme weather and traffic scenarios are equally likely to occur as more typical conditions. This approach captures the system's dynamic characteristics, as each condition may arise at any moment. Therefore, the dynamic travel times are computed using Equation (18.9) as follows:

$$\hat{t}_{ij} = b_e \cdot t_{ij} + (b_1 w + b_2 c + \epsilon) \tag{18.9}$$

Where t_{ij} denotes the fixed travel time for the edge linking nodes i and j, b_e serves as the coefficient for the standard time of the edge, b_1 indicates the coefficient associated with weather conditions, and b_2 refers to the coefficient related to traffic congestion. Furthermore, the term ϵ signifies the independent error term, which is assumed to be 0 to preserve the deterministic travel time under optimal conditions (when both w and c parameters are 0). For our analysis, we utilized the coefficients $b_e = 1$, $b_1 = 0.005$, and $b_1 = 0.005$, which were established to allow the dynamic travel time to fluctuate between the deterministic travel time and 1.125 times that value. Additionally, the edges are divided into two categories for the dynamic scenario based on these criteria: edges connected to nodes with even indices are designated as dynamic, while those linked to nodes with odd indices are classified as deterministic.

In the hybrid model, we consider three different categories of edges: deterministic, stochastic, and dynamic. The classification of each edge is based on particular criteria. An edge is classified as stochastic when the node index is even. If the node index can be divided by three, it is classified as dynamic. In all other cases, the edge is deemed deterministic. This organized approach ensures accurate classification of edges within the framework.

Due to the unpredictable nature of travel durations, there exists a risk of route failure if a UAV fails to arrive at the destination depot within the specified timeframe. Consequently, the reward earned by a specific UAV, r_d, is defined as follows:

$$r_d = \begin{cases} \sum\limits_{(i,j)\in A} u_i \cdot x_{ij}^d & \text{if } \sum\limits_{(i,j)\in A_{det}} t_{ij} x_{ij}^d + \sum\limits_{(i,j)\in A_{dyn}} \hat{t}_{ij} x_{ij}^d + \sum\limits_{(i,j)\in A_{stoch}} T_{ij} x_{ij}^d \leq T_{max} \\ 0 & \text{otherwise} \end{cases} \tag{18.10}$$

Thus, when a UAV begins its route, it encounters uncertainty in the time required to travel between nodes because of the probabilistic nature of the travel times. If the UAV does not complete the planned route within the allotted time, it loses all the rewards accumulated during

its journey. It is important to note that any partial rewards earned are only considered valid if the UAV reaches the destination node before the travel time expires.

18.5 Analysis of Results

Table 18.1 displays the results of the experiments conducted for each instance. The first column lists the instances, while the other columns present the results obtained across the four scenarios considered. The first section of the table presents the best-known solutions for the deterministic variant of the problem (*BKS*) as reported in the literature, our best-found solutions for the deterministic version of the problem (*OBD*), and the percentage gaps between these two solutions. The second section of the table presents the results obtained for the stochastic scenario. First, the best-found solutions for the deterministic scenario, when evaluated in a stochastic environment, are reported (*OBD-S*). To calculate this solution, an extensive simulation procedure was applied to the *OBD*. The core objective of this process is to evaluate the best-found deterministic solutions while accounting for stochastic uncertainties. Following that, the best-found stochastic solutions, derived using our sim-learnheuristic approach, are shown (*OBS*). This approach takes into account the stochastic elements while searching for a high-quality solution. The next section of the table displays the results obtained for the dynamic scenario. First, the best-found solutions for the deterministic version, when evaluated in a dynamic environment, are presented (*OBD-Dy*). In a similar manner, the best-found solutions for the dynamic scenario are presented (*OBDy*). Lastly, the results for the hybrid scenario are displayed. First, the best-found solutions for the deterministic version, evaluated in the hybrid scenario, are shown (*OBD-H*). Following that, the best-found solutions for the hybrid scenario are provided (*OBH*).

It is important to note that for each instance, the best-found solutions from our sim-learnheuristic approach are consistently closer to the *BKS* compared to the best-found deterministic solutions simulated under the corresponding types of uncertainty. In essence, the best-found solutions for the deterministic version of the problem are not optimal when applied to the various stochastic, dynamic, and hybrid scenarios. For example, in the dynamic scenario, when simulating the *OBD*, solutions frequently show considerable fluctuations. This is demonstrated by the *p5.3.z* instance, where *OBD-Dy* results in an expected reward of 2.34, while *OBDy* yields an expected reward of 1525.00. This discrepancy can be attributed to the manner in which our sim-learnheuristic approach navigates the search space for solutions. Specifically, our innovative methodology accounts for this uncertainty during the process of finding high-quality solutions. In other words, integrating uncertainty elements into the search process yields better results than merely solving the deterministic version of the problem and subsequently applying the solution to a real-world scenario with dynamic uncertainty. This fundamental difference is the

key reason for the significant disparities observed between the best-found solutions obtained using our sim-learnheuristic method and the results obtained by simulating the best-found deterministic solutions across the various uncertainty scenarios.

Table 18.1: Experimental results for the different scenarios.

Instances	Deterministic Scenario			Stochastic Scenario		Dynamic Scenario		Hybrid Scenario	
	BKS (1)	OBD (2)	GAP% (1)–(2)	OBD-S	OBS	OBD-Dy	OBDy	OBD-H	OBH
p1.2.r	280	275	1.78	166.60	167.74	33.24	36.84	150.86	159.23
p1.2.h	110	110	0.00	70.21	73.96	84.70	105.00	68.46	72.72
p1.2.p	250	245	2.00	134.29	148.34	0.00	57.20	130.55	143.48
p1.4.k	100	100	0.00	69.85	71.01	52.48	58.62	68.47	70.60
p2.3.h	165	165	0.00	104.68	105.01	72.72	76.56	102.30	105.06
p2.3.e	120	120	0.00	79.83	83.77	58.02	63.10	80.69	80.72
p2.3.f	120	120	0.00	91.09	91.55	120.00	120.00	90.48	90.84
p2.2.i	230	230	0.00	173.92	176.50	146.62	147.21	163.42	172.80
p2.4.j	120	120	0.00	91.09	91.55	120.00	120.00	90.48	90.84
p3.3.j	380	380	0.00	258.12	267.25	130.36	130.63	262.35	262.82
p3.3.o	590	590	0.00	346.61	379.20	129.58	309.12	339.07	359.68
p3.2.a	90	90	0.00	58.10	61.26	54.69	56.39	58.21	58.62
p3.2.d	220	220	0.00	142.13	155.44	105.25	193.86	137.19	141.27
p3.4.j	310	310	0.00	200.25	225.31	159.61	251.68	194.09	222.26
p3.4.k	350	350	0.00	247.31	257.19	218.01	221.11	242.13	249.29
p3.4.i	270	270	0.00	170.68	178.18	90.48	117.33	166.88	173.25
p5.3.z	1635	1620	0.91	943.60	1235.26	2.34	1525.00	850.78	1240.57
p5.3.o	870	870	0.00	495.03	520.64	0.00	270.00	485.46	487.78
p5.2.i	480	450	6.25	297.43	306.67	90.22	424.38	265.05	292.62
p6.2.g	660	660	0.00	431.97	436.26	417.12	446.49	406.56	411.51
p6.2.f	588	588	0.00	336.04	361.62	1.18	2.65	327.81	332.51
p7.2.c	101	101	0.00	80.75	82.03	48.05	48.16	66.18	66.20
p7.4.e	123	123	0.00	113.46	113.81	113.90	115.47	76.49	77.01
Average:	354.86	352.47	0.47	221.87	243.02	97.76	212.90	209.73	233.10

Furthermore, by comparing the computed averages across the different scenarios, we can categorize the four scenarios based on their levels of uncertainty. Firstly, the deterministic scenario serves as a reference scenario with perfect information (i.e., no uncertainty) for the expected reward across different uncertainty conditions. In contrast, the stochastic scenario exhibits a lower level of uncertainty, leading to an average expected reward of 221.87 for *OBD-S* and 243.02 for *OBS*. This can be explained by the fact that half of the edges' travel times are stochastic, while the other half are deterministic. Similarly, in the hybrid scenario, a moderate level of uncertainty is observed, resulting in an average expected reward of 209.73 for *OBD-H* and 233.10 for *OBH*. Finally, in the dynamic scenario, the highest level of uncertainty is observed, driven by dynamic conditions that lead to highly variable travel times. The resulting average

expected reward is 97.76 for *OBD-Dy* and 243.02 for *OBDy*. The dynamic scenario consists of half dynamic and half deterministic travel times, while the hybrid scenario is made up of approximately one-third dynamic and one-third stochastic travel times. In other words, the combined uncertainty of the travel times in the dynamic scenario exceeds that of both the hybrid and stochastic scenarios.

Figure 18.3 offers a summary of Table 18.1, showcasing the performance of our sim-learnheuristic approach across all the scenarios analyzed. The horizontal axis corresponds to the four different uncertainty scenarios, while the vertical axis illustrates the percentage gap relative to the *BKS* values reported in the literature. The median values are indicated by triangles and lines, while outliers are marked with circles. The average percentage gap for *OBD-S* is 33.48, and for *OBS*, it is 30.16. This indicates that, on average, the *OBS* solutions are closer to the *BKS* solutions in the stochastic scenario. In the hybrid scenario, when comparing the percentage gaps of *OBD-H* with *OBH*, it is evident that *OBH* outperforms *OBD-H* and is closer to the *BKS*. However, the average gap for *OBD-Dy* is 55.12, which is quite high. This substantial gap indicates that the deterministic approach struggles to adapt to the changing dynamics of the system, leading to significantly poorer outcomes.

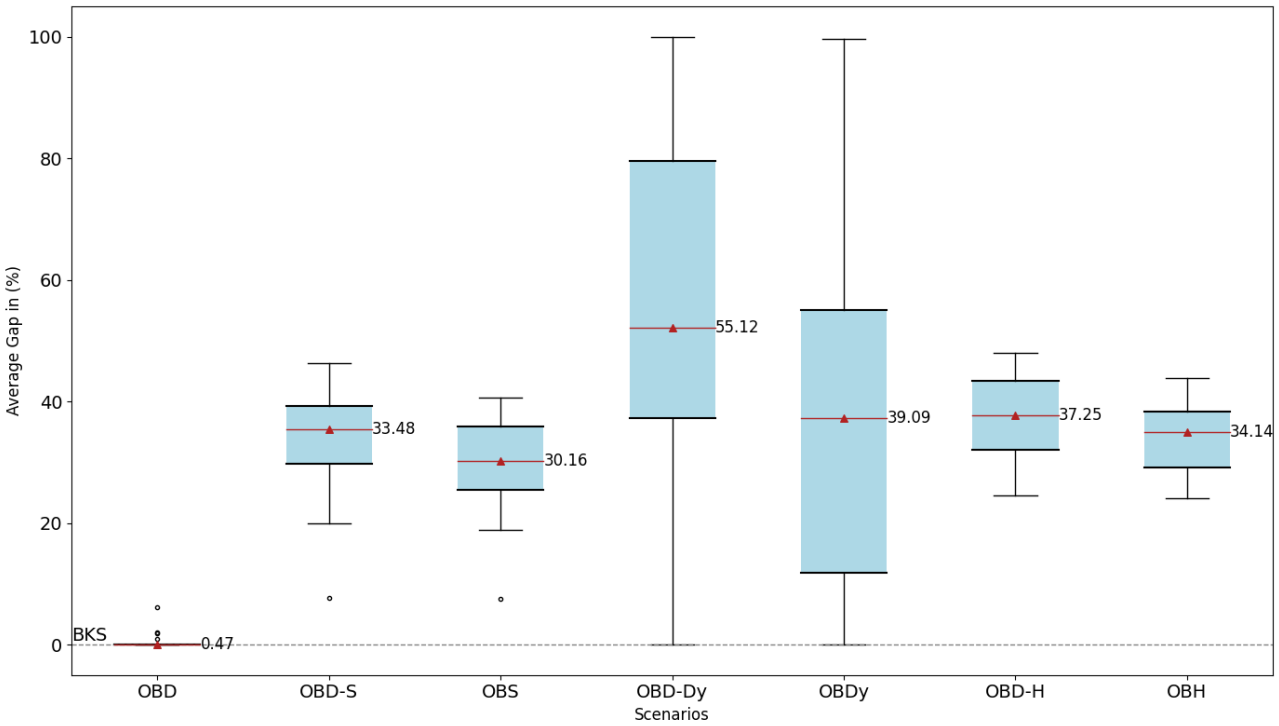


Figure 18.3: Gaps with respect to *BKS* for the different scenarios.

Part VI

Conclusions and Future Work

Chapter 19

General Conclusions

This PhD project was funded by the Spanish Ministry of Science and Innovation (PRE2020-091842 and PID2019-111100RB-C21/AEI/10.13039/501100011033), this dissertation was aimed at addressing the development of transport analytics and sustainable models for improving T&L activities in smart cities. By integrating IoT analytics, predictive modeling, and prescriptive optimization algorithms, the research sought to quantify environmental benefits, optimize real-time decision-making in urban transport systems, and support stakeholders (such as local governments, industries, and policy makers) in the efficient use of green technologies and sustainable transport modes. The project also focused on minimizing energy consumption and reducing the social and environmental impact of urban transportation activities. Hence, as indicated in Chapter 1, this thesis examined T&L problems that arise in the context of smart and sustainable cities. These problems span various application areas, including transportation, logistics, the retail industry, and production systems, each presenting unique challenges that require tailored solution techniques. By using advanced optimization methodologies, including simheuristics, learnheuristics, and DES platforms integrated with Python and ML techniques, the thesis offered innovative solutions to these challenges, enhancing decision-making processes in complex, dynamic environments.

Chapter 2 and 3 presented a comprehensive overview of key concepts and the current state of the art in areas such as vehicular networks, edge, fog, and cloud computing, as well as IoT analytics. These chapters also reviewed a range of problems in the context of logistics and smart city mobility, including the OVRP, RSP, TOP, task sequencing, and SLAP. Following this exploration, Chapter 4 explored the development of an agile optimization framework, focusing on BR heuristics, metaheuristics, and agile optimization algorithms. This framework serves as the core methodology for addressing and solving the complex logistics and mobility challenges discussed in the earlier chapters, demonstrating the adaptability and effectiveness of the proposed approach in dynamic, real-time environments.

Chapter 5 began by exploring urban mobility patterns in Barcelona, Spain, utilizing open data from the city's repositories. The chapter demonstrated the limitations of traditional models like ARIMA in predicting traffic patterns in Barcelona. ARIMA models failed to capture the non-linear and spatio-temporal dependencies of traffic data, proving not sufficient for this task. In contrast, XGBoost significantly outperformed ARIMA, achieving an average prediction accuracy of 75% and AUC values exceeding 80%. The model's robustness and interpretability, enhanced through SHAP analysis, identified key influencing factors such as geographical variables (FromNorth and ToWest) and temporal variables (DailyHour and DayMonth). These factors were critical in explaining traffic patterns and predicting states across the dataset. This approach demonstrated the potential of ML in improving urban planning and real-time decision-making by offering insights that traditional models could not. Building on these findings, Chapter 6 expanded the focus to dynamic optimization in T&L by integrating real-time data-driven methodologies. The first section in this chapter introduced a reactive heuristic approach for task scheduling of AGVs in environments with unpredictable delays. This approach utilized a dynamic delay matrix, constructed from historical data, to inform the scheduling process. The reactive heuristic was combined with traditional heuristic methods to minimize total task completion time under varying levels of uncertainty, the computational experiments showed the reactive approach consistently outperformed FIFO and non-reactive methods, with reductions in completion time averaging 5%, 6%, and 7% under low, medium, and high dynamism levels, respectively. Statistical analysis confirmed its superiority (*p-value* lower than 0.05) showing significant improvements over conventional static methods. In continuation of the use of agile optimization, Chapter 7 applied IoT analytics and agile optimization techniques to a waste collection problem, which is modeled as a DTOP-MV. This study was the first to incorporate real-time dynamic travel times derived from open data repositories. A case study in Barcelona validated the methodology, showcasing an average reward improvement of up to 18% in dynamic solutions compared to static ones, particularly during high traffic variability, proving that dynamic routing decisions based on live data outperform static methods in terms of rewards and scalability.

Chapter 8 explored methods for representing and managing uncertainty in optimization problems, introducing the concept of simheuristics (an approach that combines simulation with heuristic optimization techniques). The chapter also reviewed related work in this area, highlighting the effectiveness of simheuristics in tackling complex problems under uncertainty. Continuing this exploration Chapter 9 focused on the application of simheuristics in addressing the medical waste collection problem in smart cities. Simheuristics combine simulation, specifically MCS, with metaheuristic optimization to manage variability in real-world conditions. The chapter demonstrated how this hybrid approach could effectively handle stochastic elements,

such as random travel and pick-up times, which are often overlooked in traditional deterministic models. By integrating MCS into the simheuristic framework, the study was able to simulate various scenarios and evaluate the robustness of solutions under different random conditions, the computational experiments demonstrated the advantage of the simheuristic approach over traditional deterministic methods, achieving lower total routing times and significantly higher reliability. These results highlight the potential of simheuristics in solving stochastic optimization problems, providing efficient and reliable solutions for medical waste collection and other urban logistics challenges. Chapter 10, has provided a comprehensive overview of the concept and application of discrete-event heuristics within the broader framework of optimization techniques. The chapter explored how DES, when combined with heuristic methods, can effectively solve complex, dynamic, and event-driven problems. In particular, it examined key examples from the literature that apply these approaches across various domains, emphasizing their flexibility and adaptability in real-world scenarios. Through this discussion, the chapter highlighted the contributions of several important studies in advancing discrete-event heuristics. By reviewing these works, it demonstrated the potential of this methodology to solve problems that are otherwise difficult to handle using traditional optimization techniques alone. Chapter 11 explored the integration of cloud, fog, and edge computing to enhance transportation systems in smart cities. By using real-time data from IoT devices and open data repositories, the study demonstrated the effectiveness of agile optimization algorithms, particularly those based on BR. Tests across various scenarios demonstrated that dynamic solutions consistently outperformed static ones in terms of cost efficiency. Significant improvements were observed, with cost reductions ranging from moderate to substantial across different problem sizes. On average, the dynamic approach achieved cost savings approximately between 9% and 23%, highlighting its scalability and adaptability to varying conditions. These algorithms were shown to be capable of quickly generating high-quality solutions for large-scale vehicle routing problems, adapting continuously and making it a robust solution for real-world applications.

Chapter 12 explored the integration of DES platforms with Python to develop advanced simheuristics for various optimization problems. This chapter highlighted the versatility of combining DES with Python, enabling the development of sophisticated simulation-optimization frameworks across diverse applications, including logistics, manufacturing, and smart cities. The chapter included a tutorial on integrating multiple DES platforms, such as SimPy, AnyLogic, and Simul8, with Python. Chapter 13 expanded on these findings by offering a detailed tutorial on the integration of FlexSim and Python, presenting a practical use case that demonstrates how this integration can be used to optimize product allocation within a warehouse. The study compared three storage policies such as random, fixed, and an improved strategy using simheuristics. The fixed assignment increased throughput from 227 to 280 and reduced travel

distance from 62 to 51 meters. The improved strategy further boosted throughput to 289 and cut travel distance to 49 meters by dynamically considering future orders. This integration highlights the power of combining simulation and optimization to enhance decision-making in dynamic, real-world scenarios. In Chapter 14 a case study on task sequencing in manufacturing highlighted the effectiveness of this integration. A heuristic procedure was used to generate candidate task sequences, which were then evaluated using Simul8. A heuristic approach was used to generate task sequences, which were then evaluated using simulation to account for dynamic resource sharing and queuing complexities. The study demonstrated that the proposed methodology consistently outperformed the NEH-based baseline, achieving average makespan reductions of up to 20%, particularly in scenarios with significant resource-sharing constraints. Among the tested strategies, the MLRF/LPTF heuristic, which prioritized heavily loaded AGVs and workstations with longer processing times, proved to be the most effective. This combined approach delivered high-quality solutions while allowing stakeholders to visually assess and validate the results, enhancing decision-making and operational efficiency. To move beyond only evaluation and use communication to improve outcomes, Chapter 15 addressed key limitations of traditional approaches in the literature for solving the SLAP (such as overlooking the stochastic variability of processes and the interactions among different warehouse activities) by proposing a DES framework that delivers robust solutions under real-world warehouse conditions. This approach embraces the complexity of the problem by integrating order sequencing and picking routes into the solution construction process, using commercial simulation software to mitigate the impact of stochastic events on solution quality. In computational tests, the simheuristic consistently outperformed benchmarks. For instance A (150 products), it reduced traveled distances by 5% compared to the random strategy and 2% compared to the greedy heuristic. For instance B (756 products), reductions were approximately 3% and 1%, respectively. Average distances were reduced to 3,904 units in Instance A and 28,845 units in instance B. These results highlight the framework's ability to deliver robust, efficient solutions for real-world warehouse conditions. Similarly, Chapter 16 focused on enhancing urban mobility by addressing a complex variant of the TOP using real-world data from Barcelona's urban transportation hubs. It utilized the BR savings-based heuristic to quickly generate high-quality solutions while overcoming the limitations of traditional optimization methods. By using commercial simulation software to model dynamic urban conditions, the chapter demonstrated how the simheuristic framework bridges the gap between theoretical route planning and real-world application, as validated through computational experiments. Among the two approaches evaluated, BR-ILS consistently outperformed BR-MS. In stochastic conditions, BR-ILS achieved a median reward gap of approximately 17%, compared to 21% for BR-MS. This difference reflects the greater adaptability of BR-ILS in handling uncertainty. In a case study involving autonomous delivery

drones on a university campus, BR-ILS once again delivered higher rewards in both deterministic and uncertain scenarios. These results demonstrate the framework's effectiveness in delivering reliable and adaptable solutions for addressing complex urban logistics problems.

Chapter 17 provided a detailed overview of the learnheuristics framework, which combines ML with heuristic optimization to improve problem-solving in complex decision-making environments. The framework is recognized for its ability to adapt and enhance optimization by learning from historical data. The chapter also introduced the novel sim-learnheuristic approach, which integrates simulation into the learnheuristics framework to handle more complex scenarios involving uncertainty and dynamic conditions. This novel approach addresses challenges where traditional methods fall short. Finally, Chapter 18 introduced a hybrid version of the TOP, incorporating deterministic, stochastic, and dynamic travel times. To solve this complex problem, the chapter applied the novel sim-learnheuristic framework introduced earlier, combining a savings-based heuristic algorithm, MCS, and a multiple regression model. This integrated approach effectively accounted for randomness and variability, producing high-quality solutions in uncertain and dynamic scenarios, where in the stochastic scenario, it produced an average reward of 243 compared to 221 for deterministic solutions. In the hybrid and dynamic scenarios, it outperformed deterministic approaches with rewards of 233 and 212, compared to 209 and 98, respectively. The percentage gap in stochastic conditions was reduced from 33% (deterministic) to 30%, and in hybrid conditions, from 41% to 34%. In dynamic scenarios, deterministic solutions showed a high gap of 55%, highlighting their limitations under uncertainty. The study highlighted the limitations of relying solely on deterministic solutions, as they can lead to unreliable outcomes in real-world settings. In contrast, the sim-learnheuristic approach ensured that the generated solutions were both reliable and robust, even in environments with stochastic and dynamic travel times.

Chapter 20

Future Research Lines

The outcomes from developing and testing the various solution methods proposed in this thesis, and their application to different T&L problems, suggest several potential research directions for future exploration:

- Expanding IoT analytics for smart cities: future research could incorporate a wider range of variables to enhance the robustness, accuracy, and interpretability of IoT-based models. Potential variables include weather data (e.g., rainfall, temperature), traffic incidents, and major events that attract large crowds (e.g., sports games, conferences). Additionally, comparing open data portals from smart cities globally (focusing on factors like dataset quality, update frequency, granularity, and documentation) could provide valuable insights into their usage by different stakeholders.

- Exploring advanced ML models: there is potential to compare the performance of XGBoost with other ML algorithms and explore methods to improve its interpretability. Additionally, combining XGBoost with advanced spatio-temporal models could help better capture spatial and temporal dependencies in urban data, leading to more accurate and insightful predictive models.

- Incorporating additional uncertainties in models: the proposed solution approaches have primarily considered service and travel times as stochastic and weather and traffic conditions as dynamic. Future research could expand these models to include additional uncertain parameters such as battery capacity, battery consumption, and various other operational factors in T&L scenarios. This would provide a more comprehensive representation of real-world conditions.

- Integrating metaheuristics with DES: future research could develop a metaheuristic algorithm that interacts with the DES environment at specific stages of the solution search

process. This interaction would enable the algorithm to evaluate current solutions, extract key performance metrics from the simulation, and determine the next area to explore within the solution space, thereby optimizing the decision-making process.

- Enhancing data workflow integration: building on the integration and application strategies discussed in Part IV, further improvements could involve establishing a comprehensive data workflow (from initial inputs to final outputs). This would greatly aid decision-makers and facilitate the integration of simheuristics into warehouse management systems, enhancing operational efficiency.

- Combining simulation, metaheuristics, and deep reinforcement learning: future research could explore the simultaneous use of simulation, metaheuristics, and reinforcement learning in a unified framework. In this integrated approach, the metaheuristic component could facilitate the training of the reinforcement learning agent, thereby improving its ability to find optimal solutions and enhancing overall solution quality.

Chapter 21

Research Outcomes

All results presented in this thesis have been published, have been accepted in scientific articles or conference papers in peer-reviewed JCR/Scopus-indexed journals.

21.0.1 JCR-Indexed Articles

1. **Peyman, Mohammad**, Pedro J Copado, Rafael D Tordecilla, Leandro do C Martins, Fatos Xhafa, and Angel A Juan (2021). "Edge computing and IoT analytics for agile optimization in intelligent transportation systems". In: Energies 14.19, p. 6309.

2. Li, Yuda, **Mohammad Peyman**, Javier Panadero, Angel A Juan, and Fatos Xhafa (2022). "IoT analytics and agile optimization for solving dynamic team orienteering problems with mandatory visits". In: Mathematics 10.6, p. 982.

3. **Peyman, Mohammad**, Tristan Fluechter, Javier Panadero, Carles Serrat, Fatos Xhafa, and Angel A Juan (2023). "Optimization of vehicular networks in smart cities: from agile optimization to learnheuristics and simheuristics". In: Sensors 23.1, p. 499.

4. Leon, Jonas F, Yuda Li, **Mohammad Peyman**, Laura Calvet, and Angel A Juan (2023). "A discrete-event simheuristic for solving a realistic storage location assignment problem". In: Mathematics 11.7, p. 1577.

5. Martin, Xabier A, Sara Hatami, Laura Calvet, **Mohammad Peyman**, and Angel A Juan (2023). "Dynamic reactive assignment of tasks in real-time automated guided vehicle environments with potential interruptions". In: Applied Sciences 13.6, p. 3708.

6. Garcia, Eloi, **Mohammad Peyman**, Carles Serrat, and Fatos Xhafa (2023). "Join operation for semantic data enrichment of asynchronous time series data". In: Axioms 12.4, p. 349.

7. **Peyman, Mohammad**, Xabier A Martin, Javier Panadero, and Angel A Juan (2024). "A sim-learnheuristic for the team orienteering problem: Applications to unmanned aerial vehicles". In: Algorithms 17.5, p. 200.

8. Garcia, Eloi, Laura Calvet, Patricia Carracedo, Carles Serrat, Pau Miró, and **Mohammad Peyman** (2024). "Predictive analyses of traffic level in the city of Barcelona: from ARIMA to eXtreme gradient boosting". In: Applied Sciences 14.11, p. 4432.

9. Leon, Jonas F, **Mohammad Peyman**, Xabier A Martin, and Angel A Juan (2024). "Simulation of Heuristics for Automated Guided Vehicle Task Sequencing with Resource Sharing and Dynamic Queues". In: Mathematics 12.2, p. 271.

10. **Peyman, Mohammad**, Xabier A Martin, Javier Panadero, and Angel A Juan (2025)."A discrete-event simheuristic for enhancing urban mobility". In: Simulation Modelling Practice and Theory 140, p. 103084.

21.0.2 Other Scopus-Indexed Articles

1. **Peyman, Mohammad**, Pedro Copado, Javier Panadero, Angel A Juan, and Mohammad Dehghanimohammadabadi (2021). "A tutorial on how to connect python with different simulation software to develop rich simheuristics". In: 2021 Winter Simulation Conference (WSC). IEEE, pp. 1–12.

2. **Peyman, Mohammad**, Yuda Li, Rafael D Tordecilla, Pedro J Copado-Mendez, Angel A Juan, and Fatos Xhafa (2021). "Waste collection of medical items under uncertainty using internet of things and city open data repositories: A simheuristic approach". In: 2021 Winter Simulation Conference (WSC). IEEE, pp. 1–11.

3. Leon, Jonas F, Paolo Marone, **Mohammad Peyman**, Yuda Li, Laura Calvet, Mohammad Dehghanimohammadabadi, and Angel A Juan (2022). "A tutorial on combining Flexsim with Python for developing discrete-event simheuristics". In: 2022 Winter Simulation Conference (WSC). IEEE, pp. 1386–1400.

4. **Peyman, Mohammad**, Xabier A Martin, and Javier Panadero (2024). "Energy management of electric vehicles: AI-driven strategies for smart grid-connected charging hubs". In: International Summer Conference Decision Science Alliance on Computer Science, Valencia, Spain, June 6-8 2024. Springer.

5. **Peyman, Mohammad**, Xabier A Martin, Javier Panadero, and Angel A Juan (Accepted). "Combining heuristics, simulation, and machine learning for solving routing

problems". In: Operations Research Proceedings 2024: Selected Papers of the Annual International Conference of the German Operations Research Society (GOR), Munich, Germany, September 3-6, 2024. Springer.

Bibliography

Aazam, Mohammad and Eui-Nam Huh (2014). "Fog computing and smart gateway based communication for cloud of things". In: *Proceedings of the 2014 International Conference on Future Internet of Things and Cloud (FiCloud 2014), Barcelona, Spain, 27–29 August 2014*. IEEE, pp. 464–470.

Abdelgadir, Mayada, Rashid A Saeed, and Abuagla Babiker (2017). "Mobility routing model for vehicular ad-hoc networks (VANETs), smart city scenarios". In: *Vehicular Communications* 9, pp. 154–161.

Adi, Erwin, Adnan Anwar, Zubair Baig, and Sherali Zeadally (2020). "Machine learning and data analytics for the IoT". In: *Neural Computing and Applications* 32, pp. 16205–16233.

Agatz, Niels, Alan L Erera, Martin WP Savelsbergh, and Xing Wang (2011). "Dynamic ride-sharing: A simulation study in metro Atlanta". In: *Procedia-Social and Behavioral Sciences* 17, pp. 532–550.

Agency, European Environment (2019). *Transport*. URL: https://www.eea.europa.eu/themes/transport/intro.

Ahlgren, Bengt, Markus Hidell, and Edith C-H Ngai (2016). "Internet of things for smart cities: Interoperability and open data". In: *IEEE Internet Computing* 20.6, pp. 52–56.

Ahmed, Mohammed S and Allen R Cook (1979). "Analysis of freeway traffic time-series data by using Box-Jenkins techniques". In: *Transportation Research Record* 722.

Alatartsev, Sergey, Sebastian Stellmacher, and Frank Ortmeier (2015). "Robotic task sequencing problem: A survey". In: *Journal of Intelligent & Robotic Systems* 80, pp. 279–298.

Alisoltani, Negin, Mostafa Ameli, Mahdi Zargayouna, and Ludovic Leclercq (2022). "A Shareability Clustering Method to Solve the Dynamic Ride-Sharing Problem Considering Network Congestion". In: *TRB 2022, Transportation Research Board 101st Annual Meeting*.

Aliyu, Ahmed, Abdul Hanan Abdullah, Omprakash Kaiwartya, Yue Cao, Mohammed Joda Usman, Sushil Kumar, DK Lobiyal, and Ram Shringar Raw (2018). "Cloud computing in VANETs: architecture, taxonomy, and challenges". In: *IETE Technical Review* 35.5, pp. 523–547.

Alshaboti, Mohammed and Uthman Baroudi (2021). "Multi-robot task allocation system: Fuzzy auction-based and adaptive multi-threshold approaches". In: *SN Computer Science* 2.2, p. 87.

Amorim-Lopes, Mário, Luís Guimarães, João Alves, and Bernardo Almada-Lobo (2021). "Improving picking performance at a large retailer warehouse by combining probabilistic simulation, optimization, and discrete-event simulation". In: *International Transactions in Operational Research* 28.2, pp. 687–715.

Anwer, M Shahid and Chris Guy (2014). "A survey of VANET technologies". In: *Journal of Emerging Trends in Computing and Information Sciences* 5.9, pp. 661–671.

Arnau, Quim, Eva Barrena, Javier Panadero, Rocio de la Torre, and Angel A Juan (2022). "A biased-randomized discrete-event heuristic for coordinated multi-vehicle container transport across interconnected networks". In: *European Journal of Operational Research* 302.1, pp. 348–362.

Aydin, Omer Faruk, Ilgin Gokasar, and Onur Kalan (2020). "Matching Algorithm for Improving Ride-Sharing by Incorporating Route Splits and Social Factors". In: *PloS one* 15.3, e0229674.

Bagloee, Saeed Asadi, Madjid Tavana, Mohsen Asadi, and Tracey Oliver (2016). "Autonomous vehicles: challenges, opportunities, and future implications for transportation policies". In: *Journal of Modern Transportation* 24, pp. 284–303.

Bao, Wenhang and Xiao-Yang Liu (2019). "Spatial influence-aware reinforcement learning for intelligent transportation system". In: *arXiv preprint arXiv:1912.06880*.

Barba, Carolina Tripp, Miguel Angel Mateos, Pablo Reganas Soto, Ahmad Mohamad Mezher, and Mónica Aguilar Igartua (2012). "Smart city for VANETs using warning messages, traffic statistics and intelligent traffic lights". In: *2012 IEEE Intelligent Vehicles Symposium*. IEEE, pp. 902–907.

Barcelona, Ajuntament de (2018). *Dades Bàsiques de Mobilitat Barcelona. 2017*. URL: http://hdl.handle.net/11703/111727 (visited on 12/09/2022).

Bathla, Kanika, Vaskar Raychoudhury, Divya Saxena, and Ajay D Kshemkalyani (2018). "Real-Time Distributed Taxi Ride Sharing". In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 2044–2051. DOI: 10.1109/ITSC.2018.8569315.

Bayliss, Christopher, Roberto Guidotti, Alejandro Estrada-Moreno, Guillermo Franco, and Angel A Juan (2020). "A biased-randomized algorithm for optimizing efficiency in parametric earthquake (Re) insurance solutions". In: *Computers & Operations Research* 123, p. 105033.

Bayliss, Christopher, Angel A Juan, Christine SM Currie, and Javier Panadero (2020). "A learn-heuristic approach for the team orienteering problem with aerial drone motion constraints". In: *Applied Soft Computing* 92, p. 106280.

Bayliss, Christopher, Leandro do C Martins, and Angel A Juan (2020). "A two-phase local search with a discrete-event heuristic for the omnichannel vehicle routing problem". In: *Computers & Industrial Engineering* 148, p. 106695.

Bayliss, Christopher and Javier Panadero (2024). "Simheuristic and learnheuristic algorithms for the temporary-facility location and queuing problem during population treatment or testing events". In: *Journal of Simulation* 18.4, pp. 626–645.

Bechtsis, Dimitrios, Naoum Tsolakis, Dimitrios Vlachos, and Eleftherios Iakovou (2017). "Sustainable supply chain management in the digitalisation era: The impact of Automated Guided Vehicles". In: *Journal of Cleaner Production* 142, pp. 3970–3984.

Beek, T van der, D Souravlias, JT van Essen, J Pruyn, and K Aardal (2023). "Hybrid differential evolution algorithm for the resource constrained project scheduling problem with a flexible project structure and consumption and production of resources". In: *European Journal of Operational Research*.

Belloso, Javier, Angel A Juan, and Javier Faulin (2019). "An iterative biased-randomized heuristic for the fleet size and mix vehicle-routing problem with backhauls". In: *International Transactions in Operational Research* 26.1, pp. 289–301.

Beneicke, Julia, Angel A Juan, Fatos Xhafa, David Lopez-Lopez, and Alfons Freixes (2019). "Empowering citizens' cognition and decision making in smart sustainable cities". In: *IEEE Consumer Electronics Magazine* 9.1, pp. 102–108.

Bierzynski, Kay, Antonio Escobar, and Matthias Eberl (2017). "Cloud, fog and edge: Cooperation for the future?" In: *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE, pp. 62–67.

Bistaffa, Filippo, Christian Blum, Jesús Cerquides, Alessandro Farinelli, and Juan A Rodríguez-Aguilar (2019). "A computational approach to quantify the benefits of ridesharing for policy makers and travellers". In: *IEEE Transactions on Intelligent Transportation Systems* 22.1, pp. 119–130.

Bitam, Salim, Abdelhamid Mellouk, and Sherali Zeadally (2015). "VANET-cloud: a generic cloud computing model for vehicular Ad Hoc networks". In: *IEEE Wireless Communications* 22.1, pp. 96–102.

Blazewicz, Jacek, Jan Karel Lenstra, and AHG Rinnooy Kan (1983). "Scheduling subject to resource constraints: classification and complexity". In: *Discrete Applied Mathematics* 5.1, pp. 11–24.

Bonomi, Flavio, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli (2012). "Fog computing and its role in the internet of things". In: *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, pp. 13–16.

Borshchev, Andrei (2013). "Multi-method modeling". In: *Proceedings of the 2013 Winter Simulation Conference*. Ed. by R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl. Institute of Electrical and Electronics Engineers, Inc. Piscataway, New Jersey, pp. 4089–4100.

Boysen, Nils, René De Koster, and Felix Weidinger (2019). "Warehousing in the e-commerce era: A survey". In: *European Journal of Operational Research* 277.2, pp. 396–411.

Braekers, Kris, Katrien Ramaekers, and Inneke Van Nieuwenhuyse (2016). "The vehicle routing problem: State of the art classification and review". In: *Computers & Industrial Engineering* 99, pp. 300–313.

C. Martins, Leandro do, Patrick Hirsch, and Angel A Juan (2021). "Agile optimization of a two-echelon vehicle routing problem with pickup and delivery". In: *International Transactions in Operational Research* 28.1, pp. 201–221.

Calabrese, Francesco, Mi Diao, Giusy Di Lorenzo, Joseph Ferreira Jr, and Carlo Ratti (2013). "Understanding individual mobility patterns from urban sensing data: A mobile phone trace example". In: *Transportation Research Part C: Emerging Technologies* 26, pp. 301–313.

Calvet, Laura, Jésica de Armas, David Masip, and Angel A Juan (2017). "Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs". In: *Open Mathematics* 15.1, pp. 261–280.

Camero, Andrés and Enrique Alba (2019). "Smart city and information technology: A review". In: *Cities* 93, pp. 84–94.

Cao, Erbao and Mingyong Lai (2010). "The open vehicle routing problem with fuzzy demands". In: *Expert Systems with Applications* 37.3, pp. 2405–2411.

Cao, Hung, Monica Wachowicz, and Sangwhan Cha (2017). "Developing an edge computing platform for real-time descriptive analytics". In: *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, pp. 4546–4554.

Carlton, Gregory J and Selima Sultana (2022). "Electric vehicle charging station accessibility and land use clustering: A case study of the Chicago region". In: *Journal of Urban Mobility* 2, p. 100019.

Castaneda, Juliana, Mattia Neroni, Majsa Ammouriova, Javier Panadero, and Angel A Juan (2022). "Biased-randomized discrete-event heuristics for dynamic optimization with time dependencies and synchronization". In: *Algorithms* 15.8, p. 289.

Castillo, C, J Panadero, EJ Alvarez-Palau, and AA Juan (2024). "Towards greener city logistics: an application of agile routing algorithms to optimize the distribution of micro-hubs in Barcelona". In: *European Transport Research Review* 16.1, p. 44.

Chakraa, Hamza, François Guérin, Edouard Leclercq, and Dimitri Lefebvre (2023). "Optimization techniques for multi-robot task allocation problems: Review on the state-of-the-art". In: *Robotics and Autonomous Systems*, p. 104492.

Chao, I-Ming, Bruce L Golden, and Edward A Wasil (1996). "The team orienteering problem". In: *European Journal of Operational Research* 88.3, pp. 464–474.

Chen, Chen, Lanlan Chen, Lei Liu, Shunfan He, Xiaoming Yuan, Dapeng Lan, and Zhuang Chen (2020). "Delay-optimized v2v-based computation offloading in urban vehicular edge computing and networks". In: *IEEE Access* 8, pp. 18863–18873.

Chen, Chiu-Hung, Fu-I Chou, and Jyh-Horng Chou (2021). "Optimization of robotic task sequencing problems by crowding evolutionary algorithms". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 52.11, pp. 6870–6885.

Chen, Lu, André Langevin, and Diane Riopel (2011). "A tabu search algorithm for the relocation problem in a warehousing system". In: *International Journal of Production Economics* 129.1, pp. 147–156.

Chen, Songlin, Hong Wen, Jinsong Wu, Wenxin Lei, Wenjing Hou, Wenjie Liu, Aidong Xu, and Yixin Jiang (2019). "Internet of things based smart grids supported by intelligent edge computing". In: *IEEE access* 7, pp. 74089–74102.

Chen, Tianqi and Carlos Guestrin (Aug. 2016). "XGBoost". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 785–794. DOI: 10.1145/2939672.2939785.

Chen, Wei, Fangzhou Guo, and Fei-Yue Wang (2015). "A survey of traffic data visualization". In: *IEEE Transactions on Intelligent Transportation Systems* 16.6, pp. 2970–2984.

Chen, Zixuan, Ahmed WA Hammad, and Mana Alyami (2024). "Building construction supply chain resilience under supply and demand uncertainties". In: *Automation in Construction* 158, p. 105190.

Chiang, Mung and Tao Zhang (2016). "Fog and IoT: An overview of research opportunities". In: *IEEE Internet of Things Journal* 3.6, pp. 854–864.

Chica, Manuel, Angel A Juan, Christopher Bayliss, Oscar Cordón, and W David Kelton (2020). "Why Simheuristics?: Benefits, limitations, and best practices when combining metaheuristics with simulation". In: *SORT: Statistics and Operations Research Transactions* 44.2, pp. 0311–334.

Chica, Manuel, Angel A Juan, Bayliss Christopher, Cordón Oscar, and Kelton W. David (2020). "Why simheuristics? Benefits, limitations, and best practices when combining metaheuristics with simulation". In: *Statistics and Operations Research Transactions* 44.2, pp. 311–334.

Chin, Kwan-Wu, John Judge, Aidan Williams, and Roger Kermode (2002). "Implementation experience with MANET routing protocols". In: *ACM SIGCOMM Computer Communication Review* 32.5, pp. 49–59.

Choi, Hosoon, Piyali Chatterjee, John D Coppin, Julie A Martel, Munok Hwang, Chetan Jinadatha, and Virender K Sharma (2021). "Current understanding of the surface contami-

nation and contact transmission of SARS-CoV-2 in healthcare settings". In: *Environmental Chemistry Letters* 19, pp. 1935–1944.

Clarinval, Antoine and Bruno Dumas (2021). "Intra-city traffic data visualization: A systematic literature review". In: *IEEE Transactions on Intelligent Transportation Systems* 23.7, pp. 6298–6315.

Commission, European (2021). *New EU Urban Mobility Framework Roadmap*. URL: https://ec.europa.eu/info/law/better-regulation/have-your-say/initiatives/12916-Sustainable-transport-new-urban-mobility-framework_en.

Corlu, Canan G, Javier Panadero, Angel A Juan, and Bhakti Stephan Onggo (2020). "On the scarcity of observations when modelling random inputs and the quality of solutions to stochastic optimisation problems". In: *2020 Winter Simulation Conference (WSC)*. IEEE, pp. 2105–2113.

Corson, Scott and Joseph Macker (1999). *RFC2501: Mobile ad hoc networking (MANET): Routing protocol performance issues and evaluation considerations*.

Council, European (2014). *Conclusions on 2030 Climate and Energy Policy Framework, SN 79/14*. URL: https://www.buildup.eu/en/practices/publications/european-council-23-and-24-october-2014-conclusions-2030-climate-and-energy.

Crippa, Monica, Diego Guizzardi, Efisio Solazzo, Marilena Muntean, Edwin Schaaf, Fabio Monforti-Ferrario, Manola Banja, Jos Olivier, Giacomo Grassi, Simone Rossi, et al. (2021). "GHG emissions of all world countries". In: *Publications Office of the European Union*.

Dagkakis, Georgios and Cathal Heavey (2016). "A review of open source discrete event simulation software for operations research". In: *Journal of Simulation* 10.3, pp. 193–206.

Dai, Xin and Tianshan Ma (2021). "IoT perception and public transportation network optimization based on big data algorithms". In: *Personal and Ubiquitous Computing*, pp. 1–12.

Dang, Duc-Cuong, Rym Nesrine Guibadj, and Aziz Moukrim (2013). "An effective PSO-inspired algorithm for the team orienteering problem". In: *European Journal of Operational Research* 229.2, pp. 332–344.

Daniel, Alfred, Anand Paul, Awais Ahmad, and Seungmin Rho (2016). "Cooperative intelligence of vehicles for intelligent transportation systems (ITS)". In: *Wireless Personal Communications* 87.2, pp. 461–484.

De Armas, Jesica, Angel A Juan, Joan M Marquès, and João Pedro Pedroso (2017). "Solving the deterministic and stochastic uncapacitated facility location problem: from a heuristic to a simheuristic". In: *Journal of the Operational Research Society* 68.10, pp. 1161–1176.

Dehghanimohammadabadi, Mohammad and Thomas K Keyser (2017). "Intelligent simulation: Integration of SIMIO and MATLAB to deploy decision support systems to simulation environment". In: *Simulation Modelling Practice and Theory* 71, pp. 45–60.

Dehghanimohammadabadi, Mohammad, Mandana Rezaeiahari, and Thomas K Keyser (2017). "Simheuristic of patient scheduling using a table-experiment approach—Simio and Matlab integration application". In: *2017 Winter Simulation Conference (WSC)*. IEEE, pp. 2929–2939.

Delfani, Fatemeh, Abolfazl Kazemi, Seyed Mohammad SeyedHosseini, and Seyed Taghi Akhavan Niaki (2020). "A novel robust possibilistic programming approach for the hazardous waste location-routing problem considering the risks of transportation and population". In: *International Journal of Systems Science: Operations & Logistics*, pp. 1–13.

Demographia (2022). *Demographia world urban areas 18th annual edition*. URL: http://www.demographia.com/db-worldua.pdf.

Dharmaraj, Selvakumar, Veeramuthu Ashokkumar, Sneha Hariharan, Akila Manibharathi, Pau Loke Show, Cheng Tung Chong, and Chawalit Ngamcharussrivichai (2021). "The COVID-19 pandemic face mask waste: a blooming threat to the marine environment". In: *Chemosphere* 272, p. 129601.

Dias, Luis MS, Antonio AC Vieira, Guilherme AB Pereira, and Jose A Oliveira (2016). "Discrete simulation software ranking—A top list of the worldwide most popular and used tools". In: *2016 Winter Simulation Conference (WSC)*. IEEE, pp. 1060–1071.

Dike, Sheldon H. (Dec. 1964). "Project scheduling with resource constraints". In: *IEEE Transactions on Engineering Management* EM-11.4. Conference Name: IEEE Transactions on Engineering Management, pp. 155–157. ISSN: 1558-0040.

Dimitrakopoulos, George and Panagiotis Demestichas (2010). "Intelligent transportation systems". In: *IEEE Vehicular Technology Magazine* 5.1, pp. 77–84.

Ding, Hongyan, Cunbo Zhuang, and Jianhua Liu (2023). "Extensions of the resource-constrained project scheduling problem". In: *Automation in Construction* 153, p. 104958.

Dobre, Ciprian and Fatos Xhafa (2014). "Intelligent services for big data science". In: *Future Generation Computer Systems* 37, pp. 267–281.

Dobrovnik, Mario, David M Herold, Elmar Fürst, and Sebastian Kummer (2018). "Blockchain for and in Logistics: What to Adopt and Where to Start". In: *Logistics* 2.3, p. 18.

Dominguez, Oscar, Daniel Guimarans, Angel A Juan, and Ignacio de la Nuez (2016). "A biased-randomised large neighbourhood search for the two-dimensional vehicle routing problem with backhauls". In: *European Journal of Operational Research* 255.2, pp. 442–462.

Dominguez Rivero, Oscar L, Angel A Juan Pérez, Ignacio A De La Nuez Pestana, and Djamila Ouelhadj (2016). "An ILS-biased randomization algorithm for the two-dimensional loading

HFVRP with sequential loading and items rotation". In: *Journal of the Operational Research Society* 67.1, pp. 37–53.

Douc, Randal, Eric Moulines, and David Stoffer (2014). *Nonlinear time series: Theory, methods and applications with R examples*. CRC press.

Elder, Mark (2014). "DES view on simulation modelling: SIMUL8". In: *Discrete-Event Simulation and System Dynamics for Management Decision Making*, pp. 199–214.

Elzein, Almiqdad and Gianni A Di Caro (2022). "A clustering metaheuristic for large orienteering problems". In: *PloS one* 17.7, e0271751.

Estrada-Moreno, Alejandro, Albert Ferrer, Angel A Juan, Adil Bagirov, and Javier Panadero (2020). "A biased-randomised algorithm for the capacitated facility location problem with soft constraints". In: *Journal of the Operational Research Society* 71.11, pp. 1799–1815.

Eurostat (2017). *Statistics on European Cities*. URL: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Statistics_on_European_cities (visited on 12/09/2022).

Fagnant, Daniel J and Kara M Kockelman (2018). "Dynamic ride-sharing and fleet sizing for a system of shared autonomous vehicles in Austin, Texas". In: *Transportation* 45.1, pp. 143–158.

Faulin, Javier, Scott Grasman, Angel Juan, and Patrick Hirsch (2018). *Sustainable transportation and smart logistics: Decision-making models and solutions*. Elsevier.

Ferone, Daniele, Aljoscha Gruler, Paola Festa, and Angel A Juan (2019). "Enhancing and extending the classical GRASP framework with biased randomisation and simulation". In: *Journal of the Operational Research Society* 70.8, pp. 1362–1375.

Ferrari, G, S Busanelli, N Iotti, and Y Kaplan (2011). "Cross-network information dissemination in VANETs". In: *2011 11th International Conference on ITS Telecommunications*. IEEE, pp. 351–356.

Ferrer, Alberto, Daniel Guimarans, Helena Ramalhinho, and Angel A Juan (2016). "A BRILS metaheuristic for non-smooth flow-shop problems with failure-risk costs". In: *Expert Systems with Applications* 44, pp. 177–186.

Figueira, Gonçalo and Bernardo Almada-Lobo (2014). "Hybrid simulation–optimization methods: A taxonomy and discussion". In: *Simulation Modelling Practice and Theory* 46, pp. 118–134.

Fikar, Christian, Angel A Juan, Enoc Martinez, and Patrick Hirsch (2016). "A discrete-event driven metaheuristic for dynamic home service routing with synchronised trip sharing". In: *European Journal of Industrial Engineering* 10.3, pp. 323–340.

Frade, Inês, Anabela Ribeiro, Gonçalo Gonçalves, and António Pais Antunes (2011). "Optimal location of charging stations for electric vehicles in a neighborhood in Lisbon, Portugal". In: *Transportation Research Record* 2252.1, pp. 91–98.

Freudenberg, Nicholas, Nevin Cohen, Janet Poppendieck, and Craig Willingham (2017). "Ten years of food policy governance in New York City: lessons for the next decade". In: *Fordham Urb. LJ* 45, p. 951.

Fumi, Andrea, Laura Scarabotti, and Massimiliano M Schiraldi (2013). "Minimizing warehouse space with a dedicated storage policy". In: *International Journal of Engineering Business Management* 5, p. 21.

Health Care Waste (2021). Available at http://residus.gencat.cat/en/ambits_dactuacio/tipus_de_residu/sanitaris/index.html. Generalitat de Catalunya. (Visited on 03/24/2021).

Gestió extracentre dels residus sanitaris (2019). Available at http://residus.gencat.cat/web/.content/home/ambits_dactuacio/tipus_de_residu/residus_sanitaris/esquema_de_gestio/sanitaris02.pdf. Generalitat de Catalunya. (Visited on 03/24/2021).

Gonzalez-Martin, Sergio, Angel A Juan, Daniel Riera, Quim Castella, Rodrigo Muñoz, and Alejandra Perez (2012). "Development and assessment of the SHARP and RandSHARP algorithms for the arc routing problem". In: *AI Communications* 25.2, pp. 173–189.

Grasas, Alex, Angel A Juan, Javier Faulin, Jésica De Armas, and Helena Ramalhinho (2017). "Biased randomization of heuristics using skewed probability distributions: A survey and some applications". In: *Computers & Industrial Engineering* 110, pp. 216–228.

Graser, Anita (2019). "MovingPandas: efficient structures for movement data in Python". In: *GIForum* 1, pp. 54–68.

Gruler, Aljoscha, Christian Fikar, Angel A Juan, Patrick Hirsch, and Carlos Contreras-Bolton (2017). "Supporting multi-depot and stochastic waste collection management in clustered urban areas via simulation–optimization". In: *Journal of Simulation* 11.1, pp. 11–19.

Gruler, Aljoscha, Angel A Juan, Carlos Contreras-Bolton, and Gustavo Gatica (2018). "A biased-randomized heuristic for the waste collection problem in smart cities". In: *Applied Mathematics and Computational Intelligence 24*. Springer, pp. 255–263.

Gruler, Aljoscha, Javier Panadero, Jesica de Armas, José A Moreno Pérez, and Angel A Juan (2018). "Combining variable neighborhood search with simulation for the inventory routing problem with stochastic demands and stock-outs". In: *Computers & Industrial Engineering* 123, pp. 278–288.

— (2020). "A variable neighborhood search simheuristic for the multiperiod inventory routing problem with stochastic demands". In: *International Transactions in Operational Research* 27.1, pp. 314–335.

Gruler, Aljoscha, Antoni Pérez-Navarro, Laura Calvet Liñan, and Angel A Juan (2020). "A simheuristic algorithm for time-dependent waste collection management with stochastic travel times". In: *SORT: Statistics and Operations Research Transactions* 44.2, pp. 0285–310.

Gruler, Aljoscha, Carlos L Quintero-Araújo, Laura Calvet, and Angel A Juan (2017). "Waste collection under uncertainty: A simheuristic based on variable neighbourhood search". In: *European Journal of Industrial Engineering* 11.2, pp. 228–255.

Guimarães, Alexandre Magno Castañon, José Eugenio Leal, and Paulo Mendes (2018). "Discrete-event simulation software selection for manufacturing based on the maturity model". In: *Computers in Industry* 103, pp. 14–27.

Gunawan, Aldy, Hoong Chuin Lau, and Pieter Vansteenwegen (2016). "Orienteering problem: A survey of recent variants, solution approaches and applications". In: *European Journal of Operational Research* 255.2, pp. 315–332.

Guo, Xiaolong, Ran Chen, Shaofu Du, and Yugang Yu (2021). "Storage assignment for newly arrived items in forward picking areas with limited open locations". In: *Transportation Research Part E: Logistics and Transportation Review* 151, p. 102359.

Gupta, Anunay, Tanzina Afrin, Evan Scully, and Nita Yodo (2021). "Advances of UAVs toward future transportation: The state-of-the-art, challenges, and opportunities". In: *Future Transportation* 1.2, pp. 326–350.

Habtemichael, Filmon G and Mecit Cetin (2016). "Short-term traffic flow rate forecasting based on identifying similar traffic patterns". In: *Transportation Research Part C: Emerging Technologies* 66, pp. 61–78.

Halkos, George and Eleni-Christina Gkampoura (2021). "Where do we stand on the 17 sustainable development goals? An overview on progress". In: *Economic Analysis and Policy* 70, pp. 94–122.

Hall, Dale and Nic Lutsey (2017). "Emerging best practices for electric vehicle charging infrastructure". In: *The International Council on Clean Transportation (ICCT): Washington, DC, USA* 54.

Ham, Ruud van der (2018). "Salabim: open source discrete event simulation and animation in python". In: *Proceedings of the 2018 Winter Simulation Conference*, pp. 4186–4187.

Hammami, Farouk (2024). "An efficient hybrid adaptive large neighborhood search method for the capacitated team orienteering problem". In: *Expert Systems with Applications* 249, p. 123561.

Harri, Jerome, Fethi Filali, and Christian Bonnet (2009). "Mobility models for vehicular ad hoc networks: a survey and taxonomy". In: *IEEE Communications Surveys & Tutorials* 11.4, pp. 19–41.

Harrison, Colin, Barbara Eckman, Rick Hamilton, Perry Hartswick, Jayant Kalagnanam, Jurij Paraszczak, and Peter Williams (2010). "Foundations for smarter cities". In: *IBM Journal of Research and Development* 54.4, pp. 1–16.

Hartman, Soenke and Dirk Briskorn (2011). "A survey of variants and extensions of the resource-constrained project scheduling problem". In: *Operations Research Management Science* 51.1, p. 67.

He, Bowen and Uzair Aslam Bhatti (2024). "Smart cities and smart networks: AI applications in urban geography and industrial communication". In: *International Journal of High Speed Electronics and Systems*, p. 2440122.

He, Wu, Gongjun Yan, and Li Da Xu (2014). "Developing vehicular data cloud services in the IoT environment". In: *IEEE Transactions on Industrial Informatics* 10.2, pp. 1587–1595.

Herrera, Erika M, Javier Panadero, Angel A Juan, Patricia Carracedo, Elena Perez-Bernabeu, and Rocio De La Torre (2022). "Combining survival analysis and simheuristics to predict the risk of delays in urban ridesharing operations with random travel times". In: *2022 Winter Simulation Conference (WSC)*. IEEE, pp. 1660–1671.

Hezarkhani, Behzad, Marco Slikker, and Tom Van Woensel (2016). "A competitive solution for cooperative truckload delivery". In: *Or Spectrum* 38.1, pp. 51–80.

Hörcher, Daniel, Ramandeep Singh, and Daniel J Graham (2022). "Social distancing in public transport: mobilising new technologies for demand management under the Covid-19 crisis". In: *Transportation* 49.2, pp. 735–764.

Huang, Yan, Ruoming Jin, Favyen Bastani, and Xiaoyang Sean Wang (2013). "Large scale real-time ridesharing with service guarantee on road networks". In: *arXiv preprint arXiv:1302.6666*.

Hussain, Rasheed, Junggab Son, Hasoo Eun, Sangjin Kim, and Heekuck Oh (2012). "Rethinking vehicular communications: Merging VANET with cloud computing". In: *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*. IEEE, pp. 606–609.

Hyndman, Rob J and Yeasmin Khandakar (2008). "Automatic time series forecasting: the forecast package for R". In: *Journal of Statistical Software* 27, pp. 1–22.

Javed, Muhammad Atif, Faiz Ul Muram, Hans Hansson, Sasikumar Punnekkat, and Henrik Thane (2021). "Towards dynamic safety assurance for Industry 4.0". In: *Journal of Systems Architecture* 114, p. 101914.

Jiang, Daniel and Luca Delgrossi (2008). "IEEE 802.11 p: Towards an international standard for wireless access in vehicular environments". In: *VTC Spring 2008-IEEE Vehicular Technology Conference*. IEEE, pp. 2036–2040.

Jiménez-Meza, A, J Arámburo-Lizárraga, and E de la Fuente (2013). "Framework for estimating travel time, distance, speed, and street segment level of service (los), based on GPS data". In: *Procedia Technology* 7, pp. 61–70.

Jochem, Patrick, Sonja Babrowski, and Wolf Fichtner (2015). "Assessing CO2 emissions of electric vehicles in Germany in 2030". In: *Transportation Research Part A: Policy and Practice* 78, pp. 68–83.

Juan, Angel A, Pedro Copado, Javier Panadero, Christoph Laroque, and Rocio de la Torre (2020). "A discrete-event heuristic for makespan optimization in multi-server flow-shop problems with machine re-entering". In: *2020 Winter Simulation Conference (WSC)*. IEEE, pp. 1492–1502.

Juan, Angel A, Javier Faulin, Scott E Grasman, Markus Rabe, and Gonçalo Figueira (2015). "A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems". In: *Operations Research Perspectives* 2, pp. 62–72.

Juan, Angel A, Scott E Grasman, Jose Caceres-Cruz, and Tolga Bektaş (2014). "A simheuristic algorithm for the single-period stochastic inventory-routing problem with stock-outs". In: *Simulation Modelling Practice and Theory* 46, pp. 40–52.

Juan, Angel A, W David Kelton, Christine SM Currie, and Javier Faulin (2018). "Simheuristics applications: dealing with uncertainty in logistics, transportation, and other supply chain areas". In: *Proceedings of the 2018 Winter Simulation Conference*. IEEE, pp. 3048–3059.

Juan, Angel A, Helena R Lourenço, Manuel Mateo, Rachel Luo, and Quim Castella (2014). "Using iterated local search for solving the flow-shop problem: parallelization, parametrization, and randomization issues". In: *International Transactions in Operational Research* 21.1, pp. 103–126.

Juan, Angel A, Iñaki Pascual, Daniel Guimarans, and Barry Barrios (2015). "Combining biased randomization with iterated local search for solving the multidepot vehicle routing problem". In: *International Transactions in Operational Research* 22.4, pp. 647–667.

Juan, Angel Alejandro, Carlos Alberto Mendez, Javier Faulin, Jesica De Armas, and Scott Erwin Grasman (2016). "Electric vehicles in logistics and transportation: A survey on emerging environmental, strategic, and operational challenges". In: *Energies* 9.2, p. 86.

Ke, Liangjun, Claudia Archetti, and Zuren Feng (2008). "Ants can solve the team orienteering problem". In: *Computers & Industrial Engineering* 54.3, pp. 648–665.

Keung, Kin Lok, Carman KM Lee, and P Ji (2022). "Industrial internet of things-driven storage location assignment and order picking in a resource synchronization and sharing-based robotic mobile fulfillment system". In: *Advanced Engineering Informatics* 52, p. 101540.

Keung, Kin Lok, Carman KM Lee, and Ping Ji (2021). "Data-driven order correlation pattern and storage location assignment in robotic mobile fulfillment and process automation system". In: *Advanced Engineering Informatics* 50, p. 101369.

Khamis, Alaa, Ahmed Hussein, and Ahmed Elmogy (2015). "Multi-robot task allocation: A review of the state-of-the-art". In: *Cooperative Robots and Sensor Networks 2015*, pp. 31–51.

Khan, A. B. Feroz and Perl Ivan (2023). "Integrating machine learning and deep Learning in smart cities for enhanced traffic congestion management: An empirical review". In: *Journal of Urban Development and Management*.

Kim, Dongwook and Ilkyeong Moon (2024). "Scheduling-location problem with drones". In: *International Transactions in Operational Research* 31.5, pp. 2850–2874.

Kirac, Emre, Ridvan Gedik, and Furkan Oztanriseven (2023). "Solving the team orienteering problem with time windows and mandatory visits using a constraint programming approach". In: *International Journal of Operational Research* 46.1, pp. 20–42.

Kizys, Renatas, Angel A Juan, Bartosz Sawik, and Laura Calvet (2019). "A biased-randomized iterated local search algorithm for rich portfolio optimization". In: *Applied Sciences* 9.17, p. 3509.

Klingenberg, Cristina Orsolin, Marco Antônio Viana Borges, and José Antônio do Vale Antunes Jr (2022). "Industry 4.0: What makes it a revolution? A historical framework to understand the phenomenon". In: *Technology in Society* 70, p. 102009.

Kornhauser, Alain L, Paul Mottola, and Brian Stephenson (1977). "Transportation efficiency and the feasibility of dynamic ride sharing". In: *Transportation Research Record* 650.

Korsah, G Ayorkor, Anthony Stentz, and M Bernardine Dias (2013). "A comprehensive taxonomy for multi-robot task allocation". In: *The International Journal of Robotics Research* 32.12, pp. 1495–1512.

Koster, Rene de and Edo Van Der Poort (1998). "Routing orderpickers in a warehouse: a comparison between optimal and heuristic solutions". In: *IIE transactions* 30.5, pp. 469–480.

Kraak, Menno-Jan (2003). "The space-time cube revisited from a geovisualization perspective". In: *Proceedings 21st International Cartographic Conference*, pp. 1988–1996.

Kumar, Adarsh, Krishnamurthi Rajalakshmi, Saurabh Jain, Anand Nayyar, and Mohamed Abouhawwash (2020). "A novel heuristic simulation-optimization method for critical infrastructure in smart transportation systems". In: *International Journal of Communication Systems* 33.11, e4397.

Kumar, Korupalli V Rajesh, K Dinesh Kumar, Ravi Kumar Poluru, Syed Muzamil Basha, and M Praveen Kumar Reddy (2020). "Internet of things and fog computing applications in intelligent transportation systems". In: *Architecture and Security Issues in Fog Computing Applications*. IGI Global, pp. 131–150.

Lanza, Giacomo, Mauro Passacantando, and Maria Grazia Scutellà (2022). "Assigning and sequencing storage locations under a two level storage policy: Optimization model and matheuristic approaches". In: *Omega* 108, p. 102565.

Laroque, Christoph, Madlene Leißau, Pedro Copado, Javier Panadero, Angel A Juan, and Christin Schumacher (2021). "A biased-randomized discrete-event heuristic for the hybrid

flow shop problem with batching and multiple paths". In: *2021 Winter Simulation Conference (WSC)*. IEEE, pp. 1–11.

Laroque, Christoph, Madlene Leißau, Pedro Copado, Christin Schumacher, Javier Panadero, and Angel A Juan (2022). "A biased-randomized discrete-event algorithm for the hybrid flow shop problem with time dependencies and priority constraints". In: *Algorithms* 15.2, p. 54.

Lee, Dong Ho and Jaemyung Ahn (2024). "Multi-start team orienteering problem for UAS mission re-planning with data-efficient deep reinforcement learning". In: *Applied Intelligence*, pp. 1–23.

Lee, In Gyu, Sung Hoon Chung, and Sang Won Yoon (2020). "Two-stage storage assignment to minimize travel time and congestion for warehouse order picking operations". In: *Computers & Industrial Engineering* 139, p. 106129. DOI: 10.1016/j.cie.2019.106129.

Lenstra, Jan Karel and AHG Rinnooy Kan (1981). "Complexity of vehicle routing and scheduling problems". In: *Networks* 11.2, pp. 221–227.

Lenstra, Jan Karel, AHG Rinnooy Kan, and Peter Brucker (1977). "Complexity of machine scheduling problems". In: *Annals of Discrete Mathematics*. Vol. 1. Elsevier, pp. 343–362.

Leon, Jonas F, Yuda Li, Mohammad Peyman, Laura Calvet, and Angel A Juan (2023). "A discrete-event simheuristic for solving a realistic storage location assignment problem". In: *Mathematics* 11.7, p. 1577.

Li, Donghui, Qingbin Wang, Wei Zou, Hu Su, Xingang Wang, and Xinyi Xu (2022). "An Efficient Approach for Solving Robotic Task Sequencing Problems Considering Spatial Constraint". In: *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*. IEEE, pp. 60–66.

Li, Feiyue, Bruce Golden, and Edward Wasil (2007). "The open vehicle routing problem: algorithms, large-scale test problems, and computational results". In: *Computers & Operations Research* 34.10, pp. 2918–2930.

Li, Hong, Si-Yang Liu, Yan-Wei Huang, Yong-Quan Chen, and Zhang-Hua Fu (2019). "An efficient 2-opt operator for the robotic task sequencing problem". In: *2019 IEEE International Conference on Real-time Computing and Robotics (RCAR)*. IEEE, pp. 124–129.

Lin, Jie, Wei Yu, Nan Zhang, Xinyu Yang, Hanlin Zhang, and Wei Zhao (2017). "A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications". In: *IEEE Internet of Things Journal* 4.5, pp. 1125–1142.

Liu, Jason (2020). "Simulus: easy breezy simulation in python". In: *2020 Winter Simulation Conference (WSC)*. IEEE, pp. 2329–2340.

Liu, Jingyu, Jing Wu, and Linan Sun (2020). "Control method of urban intelligent parking guidance system based on Internet of Things". In: *Computer Communications* 153, pp. 279–285.

Lourenço, Helena Ramalhinho, Olivier C Martin, and Thomas Stützle (2019). "Iterated local search: Framework and applications". In: *Handbook of Metaheuristics*, pp. 129–168.

Lundberg, Scott M., Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee (Jan. 2020). "From local explanations to global understanding with explainable AI for trees". In: *Nature Machine Intelligence* 2.1, pp. 56–67. DOI: 10.1038/s42256-019-0138-9.

Lundberg, Scott M. and Su-In Lee (2017). "A unified approach to interpreting model predictions". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., pp. 4768–4777. ISBN: 9781510860964.

Luxen, Dennis and Christian Vetter (2011). "Real-time routing with OpenStreetMap data". In: *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM Digital Library, pp. 513–516.

Mahdavinejad, Mohammad Saeid, Mohammadreza Rezvan, Mohammadamin Barekatain, Peyman Adibi, Payam Barnaghi, and Amit P Sheth (2018). "Machine learning for Internet of Things data analysis: A survey". In: *Digital Communications and Networks* 4.3, pp. 161–175.

Maimon, Oded (1990). "The robot task-sequencing planning problem". In: *IEEE Transactions on Robotics and Automation* 6.6, pp. 760–765.

Mantel, Ronald J, Peter C Schuur, and Sunderesh S Heragu (2007). "Order oriented slotting: a new assignment strategy for warehouses". In: *European Journal of Industrial Engineering* 1.3, pp. 301–316.

Marmol, Mage, Anita Goyal, Pedro Jesus Copado-Mendez, Javier Panadero, and Angel A Juan (2021). "Maximizing customers' lifetime value using limited marketing resources". In: *Marketing Intelligence & Planning* 39.8, pp. 1058–1072.

Marmol, Mage, Leandro do C Martins, Sara Hatami, Angel A Juan, and Vicenc Fernandez (2020). "Using biased-randomized algorithms for the multi-period product display problem with dynamic attractiveness". In: *Algorithms* 13.2, p. 34.

Martí, Rafael (2003). "Multi-start methods". In: *Handbook of Metaheuristics*. Springer, pp. 355–368.

Martí, Rafael, Mauricio GC Resende, and Celso C Ribeiro (2013). "Multi-start methods for combinatorial optimization". In: *European Journal of Operational Research* 226.1, pp. 1–8.

Martins, Leandro do C, Christopher Bayliss, Pedro J Copado-Méndez, Javier Panadero, and Angel A Juan (2020). "A simheuristic algorithm for solving the stochastic omnichannel vehicle routing problem with pick-up and delivery". In: *Algorithms* 13.9, p. 237.

Martins, Leandro do C, Daniele Tarchi, Angel A Juan, and Alessandro Fusco (2021). "Agile optimization for a real-time facility location problem in Internet of Vehicles networks". In: *Networks*.

— (2022). "Agile optimization for a real-time facility location problem in Internet of Vehicles networks". In: *Networks* 79.4, pp. 501–514.

Martins, Leandro do C, Rocio de la Torre, Canan G Corlu, Angel A Juan, and Mohamed A Masmoudi (2021). "Optimizing ride-sharing operations in smart sustainable cities: Challenges and the need for agile algorithms". In: *Computers & Industrial Engineering* 153, p. 107080.

Matusiewicz, Maria, Ryszard Rolbiecki, and Marcin Foltyński (2019). "The tendency of urban stakeholders to adopt sustainable logistics measures on the example of a Polish metropolis". In: *Sustainability* 11.21, p. 5909.

Merkuryeva, Galina and Vitalijs Bolshakovs (2010). "Vehicle schedule simulation with AnyLogic". In: *2010 12th International Conference on Computer Modelling and Simulation*. IEEE, pp. 169–174.

Mohandu, Anjaneyulu and Mohan Kubendiran (2021). "Survey on big data techniques in intelligent transportation system (ITS)". In: *Materials Today: Proceedings* 47, pp. 8–17.

Montanari, Roberto, Rosa Micale, Eleonora Bottani, Andrea Volpi, and Giada La Scalia (2021). "Evaluation of routing policies using an interval-valued TOPSIS approach for the allocation rules". In: *Computers & Industrial Engineering* 156, p. 107256.

Mufalli, Frank, Rajan Batta, and Rakesh Nagi (2012). "Simultaneous sensor selection and routing of unmanned aerial vehicles for complex mission plans". In: *Computers & Operations Research* 39.11, pp. 2787–2799.

Muravev, Dmitri, Hao Hu, Aleksandr Rakhmangulov, and Pavel Mishkurov (2021). "Multi-agent optimization of the intermodal terminal main parameters by using AnyLogic simulation platform: Case study on the Ningbo-Zhoushan Port". In: *International Journal of Information Management* 57, p. 102133.

Muthuramalingam, S, A Bharathi, N Gayathri, R Sathiyaraj, B Balamurugan, et al. (2019). "IoT based intelligent transportation system (IoT-ITS) for global perspective: A case study". In: *Internet of Things and Big Data Analytics for Smart Generation*. Springer, pp. 279–300.

Nagy, Gábor and Saïd Salhi (2007). "Location-routing: Issues, models and methods". In: *European Journal of Operational Research* 177.2, pp. 649–672. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2006.04.004.

Nawaz, Muhammad, E Emory Enscore Jr, and Inyong Ham (1983). "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem". In: *Omega* 11.1, pp. 91–95.

Nordgren (2003). "Flexsim simulation environment". In: *Proceedings of the 2003 Winter Simulation Conference, 2003.* Vol. 1. IEEE, pp. 197–200.

Nzediegwu, Christopher and Scott X Chang (2020). "Improper solid waste management increases potential for COVID-19 spread in developing countries". In: *Resources, Conservation, and Recycling* 161, p. 104947.

Okulewicz, Michał and Jacee Mańdziuk (2017). "The impact of particular components of the PSO-based algorithm solving the dynamic vehicle routing problem". In: *Applied Soft Computing* 58, pp. 586–604.

Olaniyi, Ojo, Gbadgesin Adeolu Emmanuel, et al. (2024). "An examination of the relationship between traffic congestion and vehicle operating cost in Lagos State". In: *Asian Journal of Advanced Research and Reports* 18.6, pp. 21–29.

Oliva, Diego, Pedro Copado, Salvador Hinojosa, Javier Panadero, Daniel Riera, and Angel A Juan (2020). "Fuzzy simheuristics: Solving optimization problems under stochastic and uncertainty scenarios". In: *Mathematics* 8.12, p. 2240.

Osmólski, Waldemar, Roksolana Voronina, and Adam Koliński (2019). "Verification of the possibilities of applying the principles of the Physical Internet in economic practice". In: *LogForum* 15.1.

Pagès-Bernaus, Adela, Helena Ramalhinho, Angel A Juan, and Laura Calvet (2019). "Designing e-commerce supply chains: a stochastic facility–location approach". In: *International Transactions in Operational Research* 26.2, pp. 507–528.

Panadero, Javier, Alfons Freixes, Jose M Mozos, and Angel A Juan (2018). "Agile optimization for routing unmanned aerial vehicles under uncertainty". In: *Proceedings of the XIII Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB 2017), Granada, Spain*, pp. 23–26.

Panadero, Javier, Angel A Juan, Christopher Bayliss, and Christine Currie (2020). "Maximising reward from a team of surveillance drones: A simheuristic approach to the stochastic team orienteering problem". In: *European Journal of Industrial Engineering* 14.4, pp. 485–516.

Panadero, Javier, Angel A Juan, Elnaz Ghorbani, Javier Faulin, and Adela Pagès-Bernaus (2023). "Solving the stochastic team orienteering problem: comparing simheuristics with the sample average approximation method". In: *International Transactions in Operational Research*.

— (2024). "Solving the stochastic team orienteering problem: comparing simheuristics with the sample average approximation method". In: *International Transactions in Operational Research* 31.5, pp. 3036–3060.

Pappalardo, Luca, Filippo Simini, Gianni Barlacchi, and Roberto Pellungrini (2019). "scikit-mobility: A Python library for the analysis, generation and risk assessment of mobility data". In: *arXiv preprint arXiv:1907.07062*.

Pasaoglu, Guzay, D Fiorello, A Martino, G Scarcella, A Alemanno, A Zubaryeva, and C Thiel (2012). "Driving and parking patterns of European car drivers-a mobility survey". In: *Luxembourg: European Commission Joint Research Centre*.

Peidro, David, Xabier A Martin, Javier Panadero, and Angel A Juan (2024). "Solving the uncapacitated facility location problem under uncertainty: a hybrid tabu search with path-relinking simheuristic approach". In: *Applied Intelligence* 54.7, pp. 5617–5638.

Peter, Nisha (2015). "Fog computing and its real time applications". In: *Int. J. Emerg. Technol. Adv. Eng* 5.6, pp. 266–269.

Peyman, Mohammad, Pedro J Copado, Rafael D Tordecilla, Leandro do C Martins, Fatos Xhafa, and Angel A Juan (2021). "Edge computing and IoT analytics for agile optimization in intelligent transportation systems". In: *Energies* 14.19, p. 6309.

Peyman, Mohammad, Tristan Fluechter, Javier Panadero, Carles Serrat, Fatos Xhafa, and Angel A Juan (2023). "Optimization of vehicular networks in smart cities: from agile optimization to learnheuristics and simheuristics". In: *Sensors* 23.1, p. 499.

Pinedo, Michael L and Michael L Pinedo (2012). "Modeling and solving scheduling problems in practice". In: *Scheduling: Theory, Algorithms, and Systems*, pp. 431–458.

Pińskwar, Iwona, Adam Choryński, and Dariusz Graczyk (2024). "Good weather for a ride (or not?): how weather conditions impact road accidents—a case study from Wielkopolska (Poland)". In: *International Journal of Biometeorology* 68.2, pp. 317–331.

Pisinger, David and Stefan Ropke (2007). "A general heuristic for vehicle routing problems". In: *Computers & Operations Research* 34.8, pp. 2403–2435.

Poudel, Sabitri and Sangman Moh (2021). "Hybrid path planning for efficient data collection in UAV-aided WSNs for emergency applications". In: *Sensors* 21.8, p. 2839.

Prieto, Marc, George Baltas, and Valentina Stan (2017). "Car sharing adoption intention in urban areas: What are the key sociodemographic drivers?" In: *Transportation Research Part A: Policy and Practice* 101, pp. 218–227.

Qi, Qi, Jingyu Wang, Zhanyu Ma, Haifeng Sun, Yufei Cao, Lingxin Zhang, and Jianxin Liao (2019). "Knowledge-driven service offloading decision for vehicular edge computing: A deep reinforcement learning approach". In: *IEEE Transactions on Vehicular Technology* 68.5, pp. 4192–4203.

Qiao, Guanhua, Supeng Leng, Sabita Maharjan, Yan Zhang, and Nirwan Ansari (2019). "Deep reinforcement learning for cooperative content caching in vehicular edge computing and networks". In: *IEEE Internet of Things Journal* 7.1, pp. 247–257.

Quintero-Araujo, Carlos L, Juan Pablo Caballero-Villalobos, Angel A Juan, and Jairo R Montoya-Torres (2017). "A biased-randomized metaheuristic for the capacitated location routing problem". In: *International Transactions in Operational Research* 24.5, pp. 1079–1098.

Quintero-Araujo, Carlos L, Aljoscha Gruler, Angel A Juan, and Javier Faulin (2019). "Using horizontal cooperation concepts in integrated routing and facility-location decisions". In: *International Transactions in Operational research* 26.2, pp. 551–576.

R Core Team (2022). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. Vienna, Austria. URL: https://www.R-project.org/.

— (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. URL: https://www.R-project.org/.

Raba, David, Alejandro Estrada-Moreno, Javier Panadero, and Angel A Juan (2020). "A reactive simheuristic using online data for a real-life inventory routing problem with stochastic demands". In: *International Transactions in Operational Research* 27.6, pp. 2785–2816.

Rabe, Markus, Maik Deininger, and Angel A Juan (2020). "Speeding up computational times in simheuristics combining genetic algorithms with discrete-event simulation". In: *Simulation Modelling Practice and Theory* 103, p. 102089.

Raschka, Sebastian and Vahid Mirjalili (2019). *Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow 2*. Packt publishing ltd.

Rashidi, Kamran and Kevin Cullinane (2019). "Evaluating the sustainability of national logistics performance using Data Envelopment Analysis". In: *Transport Policy* 74, pp. 35–46.

Reyes, J, E Solano-Charris, and Jairo Montoya-Torres (2019). "The storage location assignment problem: A literature review". In: *International Journal of Industrial Engineering Computations* 10.2, pp. 199–224.

Reyes-Rubiano, Lorena, Laura Calvet, Angel A Juan, Javier Faulin, and Lluc Bové (2020). "A Biased-Randomized Variable Neighborhood Search for Sustainable Multi-Depot Vehicle Routing Problems". In: *Journal of Heuristics* 26.3, pp. 401–422.

Rodrigues, Eduardo Rocha, Airam Jonatas Preto, and Stephan Stephany (2005). "A new parallel environment for interactive simulations implementing safe multithreading with MPI." In: *Computer Architecture and High Performance Computing, Symposium on*. IEEE Computer Society, pp. 151–158.

Russo, Francesco, Corrado Rindone, and Paola Panuccio (2016). "European plans for the smart city: from theories and rules to logistics test case". In: *European Planning Studies* 24.9, pp. 1709–1726.

Saberian, Mohammad, Jie Li, Shannon Kilmartin-Lynch, and Mahdi Boroujeni (2021). "Repurposing of COVID-19 single-use face masks for pavements base/subbase". In: *Science of the Total Environment* 769, p. 145527.

Saeedvand, Saeed, Hadi S Aghdasi, and Jacky Baltes (2020). "Novel hybrid algorithm for Team Orienteering Problem with Time Windows for rescue applications". In: *Applied Soft Computing* 96, p. 106700.

Sanchez, Susan M and Thomas W Lucas (2002). "Exploring the world of agent-based simulations: Simple models, complex analyses". In: *Proceedings of the Winter Simulation Conference*. Vol. 1. IEEE, pp. 116–126.

Sanguesa, Julio A, Samuel Salvatella, Francisco J Martinez, Johann M Marquez-Barja, and Manuel P Ricardo (2019). "Enhancing the ns-3 simulator by introducing electric vehicles features". In: *2019 28th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, pp. 1–7.

Sariklis, Dimitrios and Susan Powell (2000). "A heuristic method for the open vehicle routing problem". In: *Journal of the Operational Research Society* 51.5, pp. 564–573.

Schmitt-Ulms, Fynn, André Hottung, Meinolf Sellmann, and Kevin Tierney (2022). "Learning to solve a stochastic orienteering problem with time windows". In: *International Conference on Learning and Intelligent Optimization*. Springer, pp. 108–122.

Schriber, Thomas J, Daniel T Brunner, and Jeffrey S Smith (2012). "How discrete-event simulation software works and why it matters". In: *Proceedings of the 2012 Winter Simulation Conference (WSC)*. IEEE, pp. 1–15.

Shumway, Robert H. and David S. Stoffer (2000). "Time series regression and ARIMA models". In: *Time Series Analysis and Its Applications*. New York, NY: Springer New York, pp. 89–212. DOI: 10.1007/978-1-4757-3261-0_2.

Silva, Allyson, Leandro C Coelho, Maryam Darvish, and Jacques Renaud (2020). "Integrating storage location and order picking problems in warehouse planning". In: *Transportation Research Part E: Logistics and Transportation Review* 140, p. 102003.

Simangunsong, Eliot, Linda C Hendry, and Mark Stevenson (2012). "Supply-chain uncertainty: a review and theoretical foundation for future research". In: *International Journal of Production Research* 50.16, pp. 4493–4523.

Speranza, M Grazia (2018). "Trends in transportation and logistics". In: *European Journal of Operational Research* 264.3, pp. 830–836.

Stach, Christoph and Andreas Brodt (2011). "vHike-a dynamic ride-sharing service for smartphones". In: *2011 IEEE 12th International Conference on Mobile Data Management*. Vol. 1. IEEE, pp. 333–336.

Story, R. (2020). *Folium*. https://python-visualization.github.io/folium/. Version: 0.11.0.

Suárez-Ruiz, Francisco, Teguh Santoso Lembono, and Quang-Cuong Pham (2018). "Robotsp–a fast solution to the robotic task sequencing problem". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1611–1616.

Sundar, Kaarthik, Sujeevraja Sanjeevi, and Christopher Montez (2022). "A branch-and-price algorithm for a team orienteering problem with fixed-wing drones". In: *EURO Journal on Transportation and Logistics* 11, p. 100070.

Swarnamugi, M and R Chinnaiyan (2021). "Modelling and reasoning techniques for context aware computing in intelligent transportation system". In: *arXiv: 2107.14374*.

Talbi, El-Ghazali (2009). *Metaheuristics: from design to implementation*. Vol. 74. John Wiley & Sons, pp. 268–308.

Tang, Chaogang, Shixiong Xia, Chunsheng Zhu, and Xianglin Wei (2019). "Phase timing optimization for smart traffic control based on fog computing". In: *IEEE Access* 7, pp. 84217–84228.

Tang, Hao and Elise Miller-Hooks (2005). "A tabu search heuristic for the team orienteering problem". In: *Computers & Operations Research* 32.6, pp. 1379–1407.

Taymaz, S, C Iyigun, ZP Bayindir, and NP Dellaert (2020). "A healthcare facility location problem for a multi-disease, multi-service environment under risk aversion". In: *Socio-Economic Planning Sciences* 71, p. 100755.

Team, RStudio (2019). *RStudio: Integrated Development Environment for R*. RStudio, Inc. Boston, MA. URL: http://www.rstudio.com/.

Teng, Haojun, Yuxin Liu, Anfeng Liu, Neal N Xiong, Zhiping Cai, Tian Wang, and Xuxun Liu (2019). "A novel code data dissemination scheme for Internet of Things through mobile vehicle of smart cities". In: *Future Generation Computer Systems* 94, pp. 351–367.

Thangam, E Cassin, M Mohan, J Ganesh, CV Sukesh, and A Prof (2018). "Internet of Things (IoT) based smart parking reservation system using raspberry-pi". In: *International Journal of Applied Engineering Research* 13.8, pp. 5759–5765.

Tordecilla, Rafael D, Pedro J Copado-Méndez, Javier Panadero, Leandro do C Martins, and Angel A Juan (2021). "An agile and reactive biased-randomized heuristic for an agri-food rich vehicle routing problem". In: *Transportation Research Procedia* 58, pp. 385–392.

Tordecilla, Rafael D, Angel A Juan, Jairo R Montoya-Torres, Carlos L Quintero-Araujo, and Javier Panadero (2021). "Simulation-optimization methods for designing and assessing resilient supply chain networks under uncertainty scenarios: A review". In: *Simulation Modelling Practice and Theory* 106, p. 102166.

Tricoire, Fabien, Martin Romauch, Karl F Doerner, and Richard F Hartl (2010). "Heuristics for the multi-period orienteering problem with multiple time windows". In: *Computers & Operations Research* 37.2, pp. 351–367.

Tufail, Ali, Abdallah Namoun, Ahmed Alrehaili, and Arshad Ali (2021). "A survey on 5G enabled multi-access edge computing for smart cities: issues and future prospects". In: *International Journal of Computer Science & Network Security* 21.6, pp. 107–118.

Upadhyay, Ram Krishna, Sunil Kumar Sharma, and Vikram Kumar (2024). "Introduction to intelligent transportation system and advanced technology". In: *Intelligent Transportation System and Advanced Technology*. Springer, pp. 3–6.

Urbanet (2023). *How many people live in cities worldwide? Infographic*. URL: https://www.urbanet.info/world-urban-population/.

Ustundag, Alp, Emre Cevikcan, Ceren Salkin, Mahir Oner, Alp Ustundag, and Emre Cevikcan (2018). "A conceptual framework for Industry 4.0". In: *Industry 4.0: Managing the Digital Transformation*, pp. 3–23.

Van den Berg, Jeroen P and Willem HM Zijm (1999). "Models for warehouse management: Classification and examples". In: *International Journal of Production Economics* 59.1-3, pp. 519–528.

Van Eynde, Rob and Mario Vanhoucke (2020). "Resource-constrained multi-project scheduling: benchmark datasets and decoupled scheduling". In: *Journal of Scheduling* 23, pp. 301–325.

Van Rossum, Guido and Fred L. Drake (2009). *Python 3 reference manual*. Scotts Valley, CA: CreateSpace, p. 400. ISBN: 1441412697.

Vis, Iris FA (2006). "Survey of research in the design and control of automated guided vehicle systems". In: *European Journal of Operational Research* 170.3, pp. 677–709.

Vivaldini, Kelen CT, Luís F Rocha, Marcelo Becker, and António Paulo Moreira (2015). "Comprehensive review of the dispatching, scheduling and routing of AGVs". In: *CONTROLO'2014–Proceedings of the 11th Portuguese Conference on Automatic Control*. Springer, pp. 505–514.

Wang, Bijun, Zheyong Bian, and Mo Mansouri (2023). "Self-adaptive heuristic algorithms for the dynamic and stochastic orienteering problem in autonomous transportation system". In: *Journal of Heuristics* 29.1, pp. 77–137.

Wang, Kai, Hao Yin, Wei Quan, and Geyong Min (2018). "Enabling collaborative edge computing for software defined vehicular networks". In: *IEEE Network* 32.5, pp. 112–117.

Wang, Xiaojie, Zhaolong Ning, Song Guo, and Lei Wang (2020). "Imitation learning enabled task scheduling for online vehicular edge computing". In: *IEEE Transactions on Mobile Computing*.

Wei, Junjun, Kejun Long, Jian Gu, Qingling Ju, and Piao Zhu (2020). "Optimizing bus line based on metro-bus integration". In: *Sustainability* 12.4, p. 1493.

Wesseling, JH, J Faber, and MP Hekkert (2014). "How competitive forces sustain electric vehicle development". In: *Technological Forecasting and Social Change* 81, pp. 154–164.

Wu, Qinghua, Mu He, Jin-Kao Hao, and Yongliang Lu (2024). "An effective hybrid evolutionary algorithm for the clustered orienteering problem". In: *European Journal of Operational Research* 313.2, pp. 418–434.

Xiao, Yu and Chao Zhu (2017). "Vehicular fog computing: Vision and challenges". In: *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom workshops)*. IEEE, pp. 6–9.

Xie, Jing, Yi Mei, Andreas T Ernst, Xiaodong Li, and Andy Song (2014). "A genetic programming-based hyper-heuristic approach for storage location assignment problem". In: *2014 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, pp. 3000–3007.

Xu, Wenzheng, Zichuan Xu, Jian Peng, Weifa Liang, Tang Liu, Xiaohua Jia, and Sajal K Das (2020). "Approximation algorithms for the team orienteering problem". In: *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, pp. 1389–1398.

Xu, Xiangbin and Chenhao Ren (2022). "A novel storage location assignment in multi-pickers picker-to-parts systems integrating scattered storage, demand correlation, and routing adjustment". In: *Computers & Industrial Engineering* 172, p. 108618.

Yan, Gongjun, Ding Wen, Stephan Olariu, and Michele C Weigle (2012). "Security challenges in vehicular cloud computing". In: *IEEE Transactions on Intelligent Transportation Systems* 14.1, pp. 284–294.

Yigitcanlar, Tan, Nayomi Kankanamge, and Karen Vella (2022). "How are smart city concepts and technologies perceived and utilized? A systematic geo-Twitter analysis of smart cities in Australia". In: *Sustainable smart city transitions*. Routledge, pp. 133–152.

Yongjun, Zhu, Zhu Xueli, Zhu Shuxian, et al. (2012). "Intelligent transportation system based on Internet of Things". In: *World Automation Congress 2012*. IEEE, pp. 1–3.

Yousefi, Saleh, Mahmoud Siadat Mousavi, and Mahmood Fathy (2006). "Vehicular ad hoc networks (VANETs): challenges and perspectives". In: *2006 6th International Conference on ITS Telecommunications*. IEEE, pp. 761–766.

Yousefikhoshbakht, Majid, Farzad Didehvar, and Farhad Rahmati (2014). "Solving the heterogeneous fixed fleet open vehicle routing problem by a combined metaheuristic algorithm". In: *International Journal of Production Research* 52.9, pp. 2565–2575.

Yu, Bing, Haoteng Yin, and Zhanxing Zhu (2018). "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18.*, pp. 3634–3640.

Yuriy, Greg and Nick Vayenas (2008). "Discrete-event simulation of mine equipment systems combined with a reliability assessment model based on genetic algorithms". In: *International Journal of Mining, Reclamation and Environment* 22.1, pp. 70–83.

Zanella, Andrea, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi (2014). "Internet of things for smart cities". In: *IEEE Internet of Things journal* 1.1, pp. 22–32.

Zhan, Ming and Kan Yu (2018). "Wireless communication technologies in automated guided vehicles: Survey and analysis". In: *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, pp. 4155–4161.

Zhang, Caiming and Yong Chen (2020). "A review of research relevant to the emerging industry trends: Industry 4.0, IoT, blockchain, and business analytics". In: *Journal of Industrial Integration and Management* 5.01, pp. 165–180.

Zhang, Guoqing, Xiaoting Shang, Fawzat Alawneh, Yiqin Yang, and Tatsushi Nishi (2021). "Integrated production planning and warehouse storage assignment problem: An IoT assisted case". In: *International Journal of Production Economics* 234, p. 108058.

Zhang, Ren-Qian, Meng Wang, and Xing Pan (2019). "New model of the storage location assignment problem considering demand correlation pattern". In: *Computers & Industrial Engineering* 129, pp. 210–219. DOI: 10.1016/j.cie.2019.01.027.

Zhao, Yunlong, Xiaoping Liu, Gang Wang, Shaobo Wu, and Song Han (2020). "Dynamic resource reservation based collision and deadlock prevention for multi-AGVs". In: *IEEE Access* 8, pp. 82120–82130.

Zhu, Li, Fei Richard Yu, Yige Wang, Bin Ning, and Tao Tang (2018). "Big data analytics in intelligent transportation systems: A survey". In: *IEEE Transactions on Intelligent Transportation Systems* 20.1, pp. 383–398.