University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Data Science

Optimization of Electric Scooter Rebalancing Tour through Mathematical Programming

Supervisor

Prof. Luigi De Giovanni

University of Padova

Master Candidate

Gita Rayung Andadari (2041509)

Academic Year

2022-2023

Hardwork never betrays.

Abstract

The pursuit of Sustainable Development Goal 11 (SDG11), striving to create inclusive, resilient, and sustainable urban environments, has become a global priority. One of SDG11's key objectives is to ensure safe, affordable, accessible, and sustainable transport systems for all. In response to this imperative, the concept of electric scooter (e-scooter) sharing has gained remarkable popularity. This innovative model allows users to access e-scooters on demand, providing a personalized last-mile solution that complements public transportation and has the potential to reduce private car ownership and greenhouse gas emissions.

As e-scooters take on an increasingly pivotal role in urban mobility, the efficient management of their distribution and charging presents a critical challenge. One of the key problems in this sharing system is to ensure that e-scooter is not over-saturated and under-utilized. This thesis embarks on a comprehensive exploration of the complexities surrounding this challenge and proposes solutions to enhance the integration of e-scooter sharing into everyday urban life. The core idea revolves around conducting a night tour operation that allow operator to drop full-charge e-scooter, but also swap the battery of the low charged unit to re-balance the e-scooter distribution.

Within this thesis, two mathematical programming formulations are presented in order to plan ahead the route and suggested actions at each station during the night tour. The first model, adapted from previous literature work on bike sharing systems rebalancing, provides a benchmark and introduces readers to the night rebalancing tour problem. The second model represents an original improved version, allowing night tour operators to swap only the battery rather than the whole e-scooter unit during operations. Both models confront the challenge of managing exponential growth in constraints and size of the solution space as the number of stations increases. In response, tailor-made branch-and-cut algorithms are developed to efficiently solve this problem. This scalable framework offers the potential to manage extensive e-scooter fleets and station networks within the city, enabling companies to enhance their operations, foster citizen trust, and establish e-scooter sharing systems as a dependable choice for daily last-mile transportation. This thesis aims to make the topic accessible and easily understandable, inviting broader participation and contributions to research in this vital field.

Contents

Listing of figures

Listing of tables

Listing of acronyms

EBSS . . . . . Electric Bike Sharing System

BFS . . . . . Breadth-First Search

DFS . . . . . Depth-First Search

BSS . . . . . Bike Sharing System

E-bike . . . . . Electric bike

E-scooter . . . . . Electric scooter

NRTP-BS . . . . . Night Rebalancing Tour Problem with Battery Swaps

MILP . . . . . Mixed Integer Linear Programming

TSP . . . . . Travelling Salesman Problem

QP . . . . . Quadratic Programming

VRP . . . . . Vehicle Routing Problem

1 Introduction

In the fantasy world, one day we all lived close to our loved ones, in a place where it is safe, clean, and well connected by public transportation. This vision mirrors the core essence of Sustainable Development Goal 11 (**SDG11**) set by the United Nations (UN), which strives to transform cities and human settlements into inclusive, safe, resilient, and sustainable spaces [32]. One of the key targets of this topic is to provide access to safe, affordable, accessible, and sustainable transport systems for all, improving road safety, notably by expanding public transport, with special attention to the needs of those in vulnerable situations, women, children, persons with disabilities, and older persons by 2030.

Back to the present world, we have seen the uprising popularity of electric scooter (e-scooter) sharing. This ride-sharing concept allows users to rent e-scooters on demand for a fee, enabling them to navigate within the city. Users can locate available scooters through smartphone applications, unlock them using QR codes, and then pay for their use. Once done, the user can leave the scooter in designated areas, making it available for the next person to use. This core concept resonates very well with the previously mentioned SDG11 target. Personally, I see this concept as a key component in encouraging more people to use public transportation by providing personalized last-mile solutions. In the long run, it can help reduce the need for private car ownership and minimize greenhouse gas emissions.

In the rapidly evolving landscape of urban transportation, electric scooters have emerged as pivotal players, offering a convenient and sustainable mode of commuting. However, with their growing prominence, the effective orchestration of their distribution and charging has become a paramount challenge. The intricate task of managing these electric scooters encompasses the need to strike a delicate balance, preventing both over-saturation and under-utilization across the city. The challenge lies not only in providing accessible and available scooters for users but also in optimizing their distribution to meet demand while minimizing operational inefficiencies. Achieving this delicate equilibrium is crucial for the sustainability and success of e-scooter sharing systems, emphasizing the pressing importance of devising innovative solutions to address the intricacies of distribution and utilization in urban environments.

Electric scooter companies must address the challenge of rebalancing. This challenge encompasses two distinct time frames: the beginning of each day and throughout the day [49]. The operations conducted during the night, before a new day begins, are what we refer to as the "night rebalancing tour." This tour aims to achieve two primary objectives: to guarantee that the scooters are fully charged and to redistribute them efficiently across the city. All of this must be achieved while minimizing operational costs.

Given the importance of the topic at hand, this thesis aspires to be more than just a technical document. We endeavor to make this thesis an accessible and comprehensible resource, ensuring that readers from various backgrounds can follow our research and gain insights into this critical field. By providing comprehensive explanations and clear illustrations of our methods and findings, we aim to encourage wider participation and contributions to research in this area.

The core concept revolves around translating the diverse criteria governing the rebalancing tour into a mathematical model with a set of linear constraints, providing a foundation for the effective utilization of mathematical optimization techniques. Within this framework, two distinct mathematical programming formulations will be explored in this thesis. The initial model draws inspiration from the work of Dell'Amico et al. on bicycle rebalancing problems [11]. Functioning as a benchmark model, it serves to familiarize the reader with the intricacies of the night rebalancing tour problem. The second model represents an original enhancement, introducing the capability for night tour operators to swap batteries during operations. We call this model as "Night Rebalancing Tour-Battery Swap" (NRTP-BS) model. Both models encounter challenges associated with the exponential growth of constraints as the number of stations increases. Consequently, this thesis delves into the development of tailor-made branch-and-cut algorithms designed to efficiently tackle this issue. The proposed solution offers a scalable framework adept at managing extensive fleets of e-scooters and nodes across urban landscapes. This scalability enhances operational management for companies, fostering increased public trust in e-scooter sharing systems as a reliable daily last-mile transportation option.

1.1 Content and Contributions

The remainder of the thesis is organized as follows:

Chapter 2 presents a comprehensive literature review, highlighting the critical nature of over-saturation and under-utilization in the context of e-scooter sharing. The review underscores the novel proposition of allowing battery swaps during the night rebalancing tour activity, a concept that still remains largely unexplored. Additionally, this chapter provides a detailed problem description to illuminate the intricacies of the issue at hand.

In Chapter 3, an exploration of existing methods and tools unfolds, shedding light on potential solution approaches for the identified problem. The chapter aims to equip the reader with a nuanced understanding of the tools available to address the complexities of e-scooter distribution dynamics. The chapter commences by acquainting the reader with the fundamental concept of mathematical programming. It then explores well-known relevant problems like the traveling salesman and vehicle routing problems. Subsequently, we delve into various methods applicable to mathematical programming problem-solving, including branch and bound, row generation, and cutting plane techniques. Finally, we present the reader with an introduction to CPLEX, a state-of-the-art solver to solve mathematical programming models. In this chapter, we will also introduce the CPLEX implementation using the Docplex library, an extension of CPLEX based on a Python API.

Chapter 4 delves into the meticulous decision-making process behind the mathematical model formulation for both the Simple Night Tour and the NRTP-BS models. This section articulates the rationale behind the chosen approaches, laying the groundwork for their subsequent implementation. The extension of the basic model ends up with a new Mixed Integer Linear Programming (MILP) formulation for the NRTP-BS model, which represents the main methodological contribution of this thesis.

The intricacies of implementing the proposed models in Docplex are unveiled in Chapter 5. The chapter delves into the specific methods and frameworks that must be constructed to ensure the generation of accurate and meaningful solutions, according to a branch-and-cut approach. It offers a detailed exploration of the technical aspects of bringing the models to life, including devising the required separation procedures. The full implementation of the branch-and-cut approach to solve the proposed MILP model is another contribution of the thesis.

Chapter 6 focuses on testing the robustness of the two proposed models, providing insights into their performance as the size increases with a higher number of stations. The analysis of the model's scalability is scrutinized to ascertain its effectiveness under varying conditions.

Finally, the concluding chapter synthesizes the entire research journey, offering a comprehensive recapitulation of the NRTP-BS solved by Mathematical Programming based techniques. This chapter serves as a cohesive endpoint, summarizing the key findings and contributions of the research while providing a holistic perspective on the proposed solutions and their implications for the broader landscape of urban transportation.

2

Literature Review and Problem Definition

2.1 The Evolution: Bike Sharing Systems to E-Scooter Sharing Systems

The e-scooter sharing system ecosystem today can thrive thanks to the great contribution of its predecessor, the conventional bike sharing system (BSS), which has already been around since 1965 [12]. One of the first known initiatives was the "Witte Fietsenplan" (White Bicycle Plan) in Amsterdam in 1965. The idea was to have a fleet of white bicycles that people could use and leave for others when they were done. However, these early programs faced challenges such as theft and vandalism.

In an effort to prevent the increasing issue of bike theft, Copenhagen, Denmark, took a pioneering step by introducing one of the earliest modern bike-sharing programs in 1995, known as *Bycyklen* [42]. Recognizing the need for enhanced security measures, this program featured specially designed bicycles equipped with a built-in lock and a coin deposit system. This innovative approach not only aimed to provide convenient and accessible transportation but also addressed the prevalent problem of bicycle theft in urban environments. The success of *Bycyklen* set a precedent, prompting other cities in Europe and Asia to adopt similar models, ushering in a new era of bike-sharing initiatives designed to enhance user experience and curb incidents of theft.

Entering the epoch of digitalization marked a significant transformation in BSS, where traditional models saw a revolution with the integration of cutting-edge technology. This evolution brought forth advancements such as electronic payment systems, GPS tracking, and the implementation of smart docking stations. The infusion of technology not only streamlined the user experience but also enhanced the overall efficiency and management of bike-sharing networks. A pivotal moment in the timeline of modern bike-sharing initiatives occurred in 2007 when Paris introduced the groundbreaking Vélib' system [42]. This program stands out as one of the most successful and widely replicated bike-sharing programs globally. Vélib' embraced a novel approach by allowing users

Figure 2.1: Docking stations: fixed [left][26], dockless [middle][16], hybrid [right][51].

the flexibility to pick up and return bikes at any station within the extensive network, a feature that significantly contributed to its widespread popularity. The Vélib' system in Paris became a trailblazer, influencing and inspiring numerous cities around the world to adopt similar technological advancements in their bike-sharing programs. This propagation of innovative systems not only expanded the reach of bike-sharing but also laid the groundwork for future developments in urban transportation, fostering a more sustainable and interconnected approach to mobility in the digital age.

The influence of technology did not stop there. In the mid-2010s, a new generation of BSS emerged with the introduction of dockless bikes. Unlike traditional systems with fixed docking stations, dockless systems allowed users to locate and unlock bikes using smartphone apps, and they could be left anywhere when the ride was complete. Companies like Ofo and LimeBike pioneered this approach [2] [15].

The advancements made over the years have contributed to the establishment of a resilient framework for vehicle sharing systems, allowing seamless integration of electric vehicles such as e-bikes and e-scooters into the sharing system ecosystem once they become popular in society. In fact, numerous bike-sharing systems, such as Ridemovi in Italy, immediately incorporated electric bikes (e-bikes) and electric scooters (e-scooters) within their framework, broadening transportation choices for users and alleviating concerns related to physical effort and extended distances [40].

E-bike and e-scooter innovation is remarkable as it opens up cycling and scooting to a broader audience. Individuals who were previously unable to ride bicycles due to steep hills and limited physical abilities can now participate, fostering a more inclusive environment and enabling greater mobility for a wider range of people.

Since e-bikes first appeared, companies around the world have done many trial and error to figure out the most effective and efficient end-to-end operation [18] [48]. To begin a journey, the customer needs to first locate an available vehicle. These available vehicles can be hosted at a fixed docking station, a dockless docking station, or a hybrid docking station (Figure 2.1).

Fixed docking stations are designated locations with physical infrastructure, such as racks or docks, where e-scooters are stationed and charged. The customer needs to locate the location of these docking stations in order to begin and end their journey, resulting in organized parking, ease of access, and centralized charging infrastructure. The drawback of this type of docking station is the decrease in flexibility for customer use. Since each e-scooter needs to have its own rack, if a station has already reached its maximum capacity, customers who want to return an e-scooter to that location need to find an alternative. Furthermore, the expansion of the new e-scooter station

will require a good amount of investment and can't easily be moved from one location to another.

The second type of docking station is a dock-less system. Contrary to its name, this type of station actually does not require a physical location. Users can locate, unlock, and initiate their e-scooter rides using a mobile app without the need for physical docking points. To control the population of e-scooters within a region, companies usually set a perimeter around the region. This way, users can park dockless e-scooters at any convenient location within the service area. The end of a trip is marked when the user responsibly parks the e-scooter within the allowed operational zone. However, this flexibility comes with a cost. It heavily relies on user consciousness to park responsibly, leading to occasional breaches such as parking on sidewalks, within residential buildings, or in locations challenging for other users to access [18].

The Hybrid type is a combination of fixed and dockless systems. Recall the dockless system concept, where users can park anywhere within the perimeter zone. The concept is similar, but instead of having a large region, hybrid types divide this large region into several small regions. Then, the user can choose to park or take a bike from anywhere within any small region. This structure combines the advantages of a fixed docking system with the added flexibility of the dockless model. Notably, there are no physical docking stations installed of this type, simplifying operational efforts when integrating new stations. If expansion or enlargement of an existing station is required, operators can effortlessly adjust the mapping of the respective station, resulting in a streamlined and cost-effective expansion process. It is also possible to install several charging stations within the perimeter where customers will get an incentive to recharge the battery upon returning the vehicle to the station. This way, the sharing system is more self-sustainable and relies less on human intervention to maintain the population battery level.

2.2 Success and Scale Bring Broad Responsibility: Addressing Over-saturation and Under-utilization Challenges as the E-scooter Sharing System Grows

Despite their widespread popularity, bike-sharing systems encountered a range of challenges, encompassing issues such as theft, vandalism, maintenance concerns, and financial sustainability. The delicate balance between supply and demand posed a persistent challenge for certain cities, resulting in instances of oversaturation and underutilization of bikes. As the e-scooter ecosystem is constructed upon the foundation of the bike-sharing system, it encounters similar challenges and issues. Striking the right equilibrium became crucial for the effective functioning and long-term viability of these systems, prompting ongoing efforts to address and overcome these obstacles within the dynamic landscape of urban transportation.

In 2017-2018, China's bike-sharing industry experienced explosive growth in a short period, with several companies flooding cities with bikes [46]. This led to oversaturation, as the sheer number of bikes exceeded demand. Many sidewalks and public spaces became cluttered with unused or broken bicycles. The intense competition resulted in financial losses for some companies, leading to the collapse of several bike-sharing startups.

In a different part of the world, Dallas, Texas, faced issues related to over saturation when multiple dockless bike-sharing companies entered the market in 2017 [39]. The city experienced a proliferation of bikes, with thousands

scattered across sidewalks, parks, and public spaces. This not only created aesthetic and logistical problems but also led to bikes being left in disrepair. Eventually, the city had to revise its regulations to address the oversupply issue.

On the other hand, Seattle's docked bike-sharing system, Pronto Cycle Share, faced challenges related to underutilization [44]. Despite efforts to promote the service, the system struggled to attract sufficient ridership. High operating costs and low utilization rates ultimately led to the discontinuation of the program in 2017. The city transitioned to exploring alternative mobility options, including dockless bike sharing and e-scooters.

These cases highlight the importance of finding a balance between supply and demand in bike-sharing systems. Oversaturation can lead to issues of space, aesthetics, and financial sustainability, while underutilization may result in the system being economically unviable. Cities and bike-sharing operators often need to carefully plan and manage the deployment of bikes to ensure the system's success and integration into the urban transportation landscape.

Numerous research efforts have been undertaken to address issues related to oversaturation and underutilization in vehicle distribution. From a strategic point of view, comprehending customer behavior is crucial for a more profound understanding and the identification of areas for improvement. Research found that the user base for e-scooter and e-bike sharing vehicles is the same [1]. Therefore, in order to solve the oversaturation and underutilization of one type of electric scooter sharing, companies need to control the combination of these two vehicles. Furthermore, within the same Almannaa research, there exist subtle differences in the purposes for which individuals use e-bike sharing and e-scooter sharing. Users tend to opt for e-bike sharing for daily activities, while e-scooter sharing is favored more for recreational purposes. This information can be used to advise an e-scooter sharing company to put a docking station close to recreational areas like parks, landmarks, and the city center. On the other hand, e-bike sharing companies can be advised to ensure they install a docking station close to office buildings, schools, or grocery stores.

Exploring customer behavior further becomes crucial in identifying optimal usage periods for vehicle-sharing services. A thorough examination of shared e-scooter mobility patterns across various cities and countries reveals a consistently observed right-skewed distribution in both trip distance and duration across all locations. The pattern suggests that users prefer utilizing electric sharing vehicles for shorter distances and durations. This valuable insight can be applied to mitigate the issue of underutilization. Effective promotional strategies, like providing discounts for short journeys or introducing a membership scheme where users pay for one month and receive free usage for the initial 30 minutes (as demonstrated by the Vèloh company in Luxembourg), can attract both new and returning users. Additionally, this information provides assurance to vehicle-sharing companies and city planners that their strategy of integrating electric vehicle sharing as a first and last-mile solution aligns harmoniously with customer natural behavior.

Furthermore, in a study conducted in Montreal, Canada, the effects of weather, temporal characteristics, bicycle infrastructure, land use, and urban form attributes on bike rentals and returns at the station level were examined using linear mixed models [13]. The research revealed correlations between the type of day and bike rentals and returns. Additionally, the proximity of bike-sharing stations to large public places, such as malls and universities, significantly influenced the utilization of bikes. This study offers valuable insights for vehicle-sharing companies in determining optimal station locations to maximize vehicle utilization. With a better understanding of the impact of weather and the type of day, companies can strategically plan their logistics. For instance, during the rainy season, when bike sharing is less utilized, companies can adjust the number of released bikes in service to

maintain a balanced distribution.

Knowing user demand in advance is crucial to determining the number of bicycles in service needed to avoid over and under distribution. Yang et al. (2016) constructed random forest models to forecast demand [47], while Li and Zheng (2019) introduced an efficient Gaussian process regressor based on similarity for predicting the number of bikes to be rented at various locations, including stations, clusters, and the system level [27]. In a separate study, Kim (2021) employed random forest models to predict bike rentals and returns at the cluster level [25]. Additionally, Kim utilized two distinct methods within a hierarchical demand prediction framework to estimate demands at the station level.

At the operational level, addressing oversaturation and underutilization often involves implementing rebalancing operations. The fundamental concept behind rebalancing operations is to strategically relocate vehicles within the system to achieve a more equitable distribution. This process guarantees that no specific location is burdened with an excess of unused vehicles, ensuring the optimal placement of vehicles at the right location and time.

Rebalancing strategies are categorized into staff-based strategies [11] [49] and user-based strategies [36]. Staff-based strategies involve determining routes and quantities of bikes for the fleet of vehicles responsible for rebalancing. User-based strategies, on the other hand, encourage users to pick up bikes at high-inventory-level stations and return them at low-inventory-level stations through incentives or regulations.

In the staff-based rebalancing process, a truck is utilized to transport fully charged e-bikes or e-scooters from the depot station to locations requiring additional vehicles. For dockless and hybrid docking stations, the same truck is employed to transfer low-battery e-scooters to the depot for charging. To minimize vehicle idle time at the depot, the option of charging bikes while the truck is in motion has been explored by Osorio et al. in 2021 [33]. Given the resulting varied levels of e-scooter charge, the system employs a mixed-integer program for the multi-commodity inventory routing problem, treating the commodities as e-scooters with distinct states of charge.

Staff-based rebalancing strategies can be further divided into static and dynamic categories based on when rebalancing occurs. Static rebalancing happens at night with less demand and road traffic, aiming to minimize vehicle travel time or user dissatisfaction. Specific features, such as constraints on truck visits [7]. Dynamic rebalancing, more widely used in large cities, efficiently addresses dynamic rebalancing problems during peak times. Objectives include minimizing travel costs, user dissatisfaction, or deviations from target inventory levels. As computational power continues to experience exponential growth, the e-bike sharing system has harnessed the advantages of machine learning technology as well. In particular, reinforcement learning approaches have become increasingly preferred for addressing rebalancing challenges [28].

With the improvement of technology, e-scooter manufacturers these days have enabled an easy-to-switch battery mechanism, which gives e-scooter sharing companies more flexibility to deal with low-battery vehicles: charge them or just swap the batteries as pictured in Figure 2.2. This flexibility also helps the e-scooter company bring down the price of vehicle investment. Instead of keeping adding new scooters for about 500€/unit, companies like Lime can upkeep their inventory by buying a couple more new batteries for about 150€/unit. On top of bringing the investment price down, swapping batteries also reduces the vehicle's idltime,me which increases the usability of the vehicle, supporting citizens zipping around the city.

In 2020, Shangyao et al. introduced an innovative concept in their research [43], suggesting the implementation of battery exchange stations. Differing from conventional docking stations, these facilities empower customers to visit and swap batteries, ensuring a seamless continuation of their journeys. An alternative strategy involves strategically placing low-cost charging piles in appropriate locations to conserve battery charges during

Figure 2.2: E-Scooter easy battery change [8]

extended trips. Consequently, the critical consideration of identifying optimal locations for electric scooter battery swapping stations, combined with the deployment of charging piles, emerges as pivotal for enhancing average distance and travel time. A similar approach has also been proposed by Zhou et al. in 2023 [52]. In this model, power-deficient batteries are replaced and subsequently charged in a central depot or street-side cabinets. The paper proposes a method for modeling the system states of the Electric Bike Sharing System (EBSS), encompassing e-bike stations and battery cabinet stations, utilizing Markov chain dynamics that consider both e-bike numbers and battery power levels.

2.3 Research Gap, Goals, and Problem Definition

Extensive research efforts have been dedicated to addressing the challenges of oversaturation and underutilization in bike-sharing systems. Some of them tackle the issue from a strategic point of view, while others tackle it from the operational side. From the operational side, we've talked about the static rebalancing tour and the recent distribution at night. While research on this topic does exist, there has been limited discussion about integrating battery swaps as an option during night tour operations. This is despite our earlier findings in the preceding section, which conclusively demonstrated that investing in additional batteries is more cost-effective than acquiring more e-scooter units. Moreover, the practice of battery swaps facilitates more effective rebalancing in terms of space requirements, as a battery normally takes less space than a unit of an e-scooter. These facts enable operators to turn a low-battery e-scooter into a fully charged one during a single cycle of night tour operation, which leads to more efficient night tour operation. Moreover, many operators use balancing vehicles without battery recharge equipment, making it necessary to move already charged batteries.

Having established the need for further research into the integration of battery swaps during nighttime tour operations, let's deep dive into this specific type of operation. In the night tour operation, a vehicle, specifically a truck, transports e-scooters and batteries around town, facilitating the reconfiguration of e-scooter distribution to return it to its normal state. To support night tour operations, companies need to have a designated place that stores fully charged e-scooters and batteries. This place is called *depot*. To simplify the problem, let's focus on the hybrid docking e-scooter (Figure 2.1), where the user has to start and end their journey from a docking station,

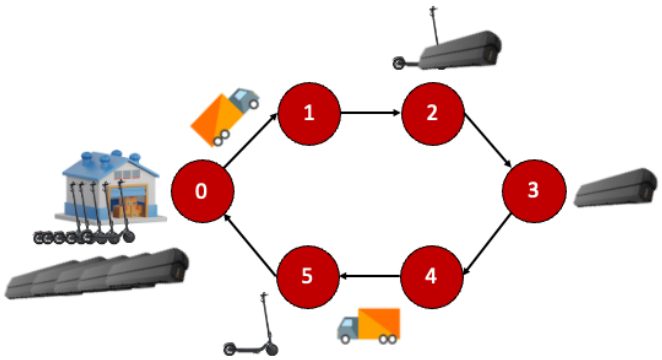

Figure 2.3: Night tour operation

but the maximum capacity of each station can easily be adjusted. Therefore, the journey of the truck can simply be viewed as a tour of these nodes across the city.

Consider Figure 2.3, which represents the night tour operation. For demonstration purposes and without losing generality, let's consider that there are 5 nodes across the city and 1 supporting depot. To rebalance the e-scooter distribution in the morning, the company will send one truck to check the vehicle availability at docking stations across the city. The truck will start from a depot and bring enough supplies for the night. Of course, the supplies will be subject to the availability of space in the truck. After loading the supplies, the truck will start its journey by visiting the docking stations one by one. At the docking station, the night tour operator will act according to the needs of that particular docking station. If it needs more e-scooters, the operator will drop more e-scooters. If there are too many vehicles, the operator will pick up some e-scooters so they can be better placed at other docking stations. In the following, we list some possible operations while visiting a docking station:

- operator pick-up full-charge e-scooters;
- pick-up low-battery e-scooters;
- drop-off full-charge e-scooters;
- swap the battery of low-battery e-scooters on-board and drop them off;
- swap the battery of low-battery e-scooters on-site.

Up to this point, the night tour operation may seem straightforward; however, a significant challenge remains: determining where to drive and what actions to take upon arrival. While drivers could potentially navigate aimlessly, relying on their intuition upon reaching a station, such an approach would lead to inefficient operations. Consider the following scenario: a truck can only accommodate one e-scooter, and there are two docking stations to be visited. Node 1 has an excess e-scooter, while Node 2 requires two e-scooters. If the driver randomly chooses to "visit Node 2 first," unaware that they cannot rebalance the station due to having only one e-scooter in the truck, the operation becomes inefficient. In this scenario, the ideal solution is to visit Node 1 first, collect the surplus e-scooter, and then proceed to Node 2 to unload all the scooters. Knowing the ideal solution ahead of time ensures the efficiency of the operation by reducing the amount of back and forth between stations that

the operator has to endure. This approach allows the e-scooter company to maintain a cost-effective night tour operation.

This thesis aims to address the critical question of "where to drive and what actions to take upon arrival" during the night tour operation. By solving this question, we are making a great contribution to solving the oversaturation and under-utilization of e-scooter vehicles in the system. The proposed solution involves providing operators with a predefined set of tours based on field conditions, along with recommendations for actions to be taken at each station before the commencement of the operation. This solution is derived from an objective to ensure the shortest route while rebalancing the e-scooter distribution at night. We will utilize mathematical programming techniques to translate and solve this problem, allowing a scalable solution as the ecosystem of e-scooters increases over time. The stated problem is referred to NRTP-BS. In the subsequent Chapter 4, we will provide a more formal statement as well as a mathematical formulation based on Mathematical Programming techniques, namely MILP.

3

Methodological and Implementation Tools

In this chapter, we will step aside from the e-scooter night tour universe for a bit. We will focus on understanding the methodologies and tools that are available in order to be able to customize a method that fits our problem like a glove. We will see the core concept of the Traveling Salesman Problem and a branch of this, the Vehicle Routing Problem. We will also get a deeper understanding of what an optimization problem is and how it can be solved through mathematical programming and, in particular, MILP. After becoming familiar with MILP, we will explore the branch and bound algorithms that can be used to solve MILP. Moreover, we will also explore methods that can help us expedite the runtime of the program by introducing the concepts of lazy constraint and valid inequality (cut).

3.1 Mathematical Programming

Mathematical programming, also known as mathematical optimization, is a field of mathematics and computer science that deals with the formulation and solution of mathematical models for decision-making problems [29]. The objective is to find the best possible solution from a set of feasible solutions, where "best" is defined based on certain criteria or objectives. In mathematical programming, a mathematical model is created to represent the relationships among decision variables, constraints, and an objective function. Decision variables are the variables that the decision-maker can control, and the objective function represents the goal to be optimized (maximized or minimized). Constraints are conditions that the solution must satisfy.

There are various types of mathematical programming, including linear programming (LP), integer programming (IP), mixed-integer programming (MIP), nonlinear programming (NLP), and others [29]. The appropriate type depends on the nature of the decision problem and the mathematical relationships involved. Solving these mathematical models helps in making optimal decisions in various fields, such as operations research, logistics, finance, engineering, and more.

Mathematical optimization is a fundamental concept that permeates various disciplines, ranging from mathematics and computer science to engineering, economics, and beyond. In reality, an optimization problem refers to the process of making something as effective or functional as possible. Bringing reality to the context of mathematical modeling, an optimization problem involves identifying the best solution from a set of feasible solutions. This 'best' solution is often determined by maximizing or minimizing an objective function, which represents the quantity to be optimized. An optimization problem typically consists of the following components:

1. decision variables,
2. objective function,
3. constraints,
4. feasible region.

Decision variables in an optimization problem are the elements that the algorithm adjusts to discover the best possible solution. Essentially, they embody the choices or decisions that are integral to the problem. To uncover the optimal solution, an optimization problem requires a well-defined function that illustrates the relationship between these decision variables and the desired outcome. This function, known as the objective function, serves to quantify the success or effectiveness of the problem. Depending on the problem's nature, the objective may involve maximizing or minimizing this function. In practical terms, real-world constraints often come into play; for instance, a company might have a limited fleet, or an operation may need to be completed within a specified timeframe. These limitations are translated into the model through the introduction of constraints—conditions that must be met for a solution to be deemed feasible. Constraints essentially outline the boundaries or restrictions on the decision variables. Lastly, the feasible region encompasses all conceivable combinations of decision variable values that adhere to the established constraints. In essence, it is the space where a viable solution can be found.

Optimization problems can be broadly categorized into three main types: continuous, discrete, and mixed optimization [17]. In continuous optimization, decision variables have the flexibility to assume any real value within a defined range. This is often seen in calculus problems, such as determining the minimum of a differentiable function. The second category, discrete optimization, pertains to problems where decision variables can only adopt specific, separate values. Essentially, a discrete optimization problem introduces additional constraints to a continuous problem. Examples include well-known problems like the Traveling Salesman Problem and the Knapsack Problem. Lastly, mixed optimization problems involve decision variables that encompass both continuous and discrete components. In other words, while one variable may take any real value, another is constrained to integer or discrete values.

Optimization problems manifest across diverse fields, addressing unique challenges and enhancing efficiency. In the realm of operations research, these problems tackle the optimization of resource allocation, logistics, and scheduling. In engineering, the focus extends to design optimization, structural optimization, and fine-tuning parameters [37]. In economics, optimization comes into play when maximizing profits, minimizing costs, and optimizing portfolios [14]. Machine learning also leverages optimization for tasks like hyper-parameter tuning, feature selection, and refining model performance. This wide-reaching applicability highlights the versatility of optimization across various domains [50].

To better understand the optimization problem concept, let's take a look at the following example: Consider this integer programming problem in equation 3.1, involving an objective to maximize, three constraints, and three non-negative integer variables.

$$\begin{aligned} \text{maximize } & x_1 + x_2 \\ \text{subject to } & 2x_1 + 12x_2 \leq 30 \\ & 4x_1 - x_2 \leq 10 \\ & x_1, x_2 \geq 0 \end{aligned} \tag{3.1}$$

To solve this problem, we need to explore the feasible region and find the minimum value within the feasible region (Figure 3.1). It's crucial to note that the feasible set is distinct from the objective function, which outlines the criterion for optimization in this example. Considering all values within the feasible region, the point that maximizes the objective function is when $x_1 = 3$ and $x_2 = 2$. This point lies at the intersection of two of the three constraints that are linearly independent of each other. This point is considered a corner point.

Formally, a corner point is a point that lies on the boundary of many linearly independent constraints. This point is a point of interest for us because of Lemma 1 stated in [41].

Lemma 1. *A line is a set $L = \{r + \lambda s : \lambda \in \mathbb{R}\}$ where $r, s \in \mathbb{R}^n$ and $s \neq 0$. Let $P = \{x : a_i^T x \leq b_i \forall i\}$. Suppose P does not contain any line. Suppose the LP max $\{c^T x, x \in P\}$ has an optimal solution. Then some extreme point is an optimal solution.*

Guaranteed by this lemma, the journey of finding the extreme point of the optimization problem has now narrowed down a lot from considering all the points that exist in the feasible region (the amount is normally infinite when the decision variable is continuous) to a couple of corner points in the feasible region. In our example, we have four corner points:

1. $x_1 = x_2 = 0$, objective function $= 0$
2. $x_1 = 2.5, x_2 = 0$, objective function $= 2.5$
3. $x_1 = 0, x_2 = 2.5$, objective function $= 2.5$
4. $x_1 = 3, x_2 = 2$, objective function $= 5$

By analyzing the objective function value from the 4 corner points, it can be concluded that the solution to the problem in 3.1 is indeed 5, when $x_1 = 3$ and $x_2 = 2$. Based on Lemma 1, an efficient-in-practice algorithm to solve problems formulated a an LP has been devised, the *simplex algorithm*.

3.2 Travelling Salesman and Vehicle Routing Problem

The Traveling Salesman Problem (TSP) is a classic optimization problem in the field of combinatorial optimization [3]. First formulated in the 1800s, the TSP involves finding the shortest possible route that visits a set of cities and returns to the original city.

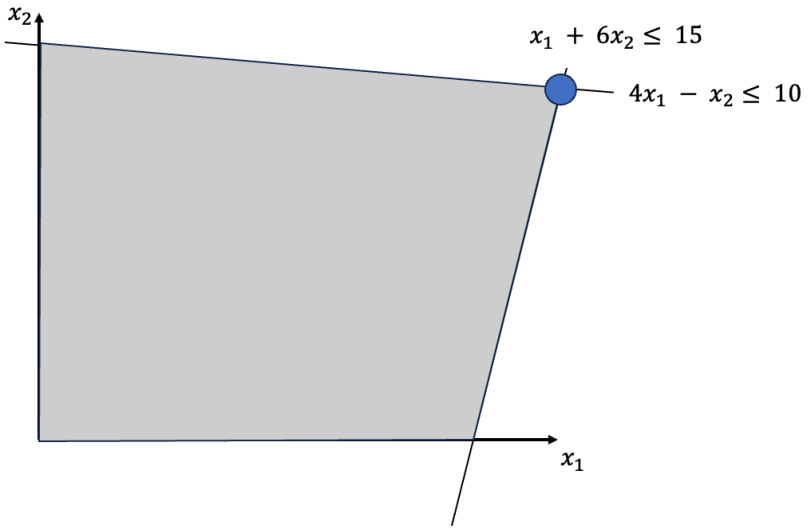

Figure 3.1: Feasible region for optimization problem stated in equation 3.1

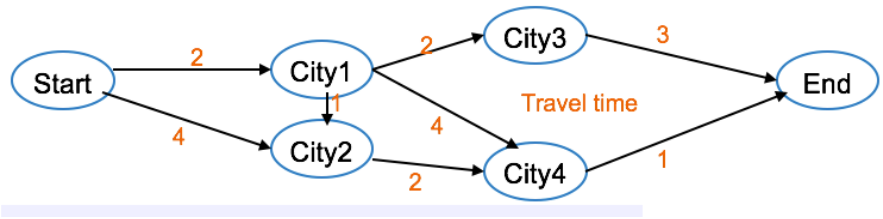

Figure 3.2: Shortest Path. Source: [23]

The Shortest Path problem is finding the quickest route through a map without the constraint of visiting all of the cities, like in TSP. Consider it like figuring out the fastest way to travel between two cities in a web of interconnected cities. This problem stands out as a specific type within the broader transshipment challenge. In this specialized scenario, there's a single starting point and a single destination, each with an equal demand (let's say, 1 unit).

Take Figure 2.3 as an example. Imagine each node on the map as a city and the connecting lines as roads. The travel time for each road is explicitly mentioned. Now, we introduce the variable $x(i, j)$, which equals to 1 if the road between cities i and j is part of the shortest route and 0 if it is not. We can now formulate the Shortest Path Problem as an ILP. The primary objective is to minimize the overall travel time. Similar to other network problems, we can interpret the constraints as flow conservation principles. Each city has its own constraints, emphasizing that precisely one road should be selected into and out of each city. Despite discussing the x variables in terms of 0–1 values, we can treat them as continuous. This is because the formulation of the shortest path problem has the *integrality property*, that is, the corner points of the feasible region of its LP relaxation take integer values. The simplex algorithm is thus sufficient to find an optimal solution.

The Traveling Salesman Problem (TSP) is classified as NP-hard because it belongs to a class of computational

problems that are at least as hard as the hardest problems in NP (nondeterministic polynomial time) [10]. In other words, the difficulty of solving the TSP increases rapidly with the size of the problem, and no known algorithm can guarantee a solution in polynomial time for all instances of the problem (unless $P = NP$). This NP-hard problem has applications in various domains, including logistics, transportation, and manufacturing. Over the years, researchers have proposed numerous algorithms to address different aspects of the TSP, ranging from exact algorithms to heuristics and meta-heuristics.

Even the TSP can be formulated as an ILP, using variables similar to those of the shortest path problem. In this case, the integrality property does not hold. Early solutions to the TSP involved exact algorithms such as the branch-and-bound method, which systematically explores the solution space to find the optimal solution [38]. However, with the growing complexity of real-world instances, heuristics like the nearest neighbor and farthest insertion algorithms, or metaheuristics like genetic algorithms and neighborhood search gained popularity for providing reasonable solutions in shorter computational times.

The Vehicle Routing Problem (VRP) is another significant problem in logistics and transportation. It extends the TSP by introducing a fleet of vehicles (in the capacitated version), aiming to serve a set of customers with known demands [6]. The primary objective is to minimize the total travel cost or time while satisfying capacity constraints and visiting all customers. Numerous variants of the VRP exist, each posing unique challenges and requiring tailored solutions. In the literature, researchers have explored various methods to tackle the VRP, including exact algorithms like branch and bound [23], metaheuristics such as simulated annealing, tabu search [4] and hybrid approaches that combine multiple techniques. The choice of the most appropriate method often depends on the specific characteristics of the VRP instance at hand.

Recalling the night tour operation problem discussed in Chapter 2 and the explanation of VRP, a notable similarity emerges between these two concepts. Both aim to determine the optimal route, with a focus on minimizing distance. In the night tour operation problem, the question is posed: "What is the most efficient tour, considering the shortest distance, for the night tour operation?" On the other hand, the VRP, in general, addresses the broader challenge of "finding the shortest possible routes that visit a set of cities and return to the original city, given a fleet of vehicles with limited capacity, with the objective of serving a set of customers with known demands." Drawing on this similarity, we intend to approach the solution of the night tour operation problem by casting it as a vehicle routing problem.

3.3 Branch and Bound

Now that we have established that we are dealing with a vehicle routing problem, one method that can be used to solve this is the branch and bound method. It is a general algorithmic technique that was crafted to solve combinatorial optimization problems, including the TSP and VRP and, more generally, MILP models. This divide-and-conquer approach systematically divides the problem into smaller sub-problems, solves them individually, and combines their solutions to obtain the global optimum. In the context of the TSP and VRP, researchers have utilized the branch-and-bound method to explore the solution space efficiently, pruning branches that cannot lead to an optimal solution.

The effectiveness of branch and bound lies in its ability to discard partial solutions that are provably worse than the best-known solution, thereby reducing the search space and improving computational efficiency [31].

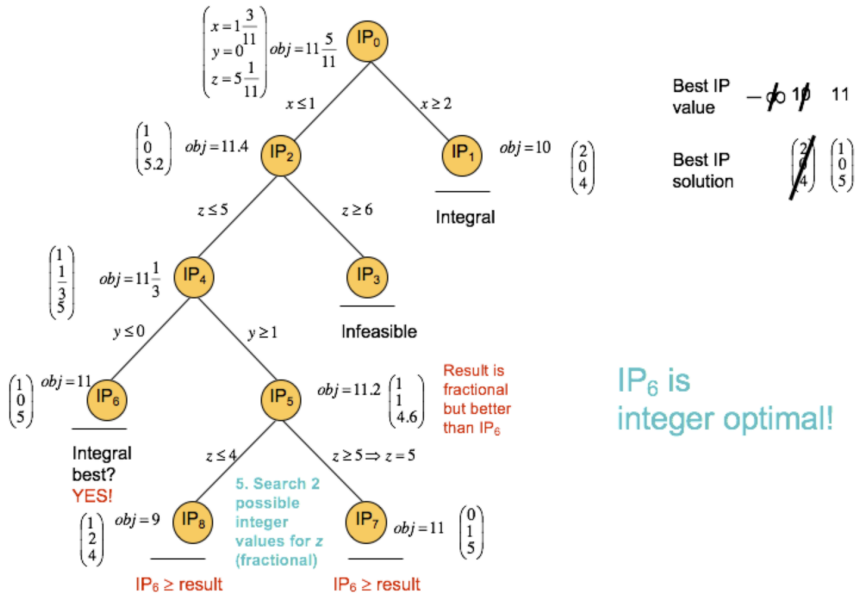

Figure 3.3: Branch and bound example. Source: [23]

However, the success of branch and bound depends on the problem's characteristics, and its performance may vary across different instances.

To help the reader understand how this concept can be applied to solve MILP models, let's take a look at the following example (equation 3.2). The branch and bound operation of this problem is summarized in Figure 3.3.

$$\begin{aligned} \text{maximize } & x + y + 2x \\ \text{subject to } & 7x + 2y + 3z \leq 36 \\ & 5x + 4y + 7z \leq 42 \\ & 2x + 3y + 5z \leq 28 \\ & x, y, z \in \mathbb{Z}_+ \end{aligned} \tag{3.2}$$

The initial node in the branch-and-bound tree corresponds to the LP relaxation of the original IP model. In LP relaxation, the integral variables are treated as continuous variables. Solving the LP relaxation provides an upper bound for the original IP problem; in this instance, the bound is eleven and five elevenths. Currently, the lower bound is negative infinity.

Since the solution in this case is fractional, the algorithm persists in the tree search to discover an integer solution. In the branch-and-bound process, the algorithm selects a variable to branch on, such as x and introduces two constraints to create two sub-problems. These constraints are based on the relaxed value of x, specifically, one and three elevenths. In one sub-problem, x must be less than or equal to one, and in the other, x must be greater than or equal to two to eliminate the fractional solution. While IP2 yields another fractional solution, IP3 produces an

integer solution. This integer solution of value 10 becomes the new lower bound for the original maximization problem, representing the best current solution.

The algorithm continues its iterations, generating new sub-problems and pruning infeasible ones. For instance, from IP4, two sub-problems emerge based on the fractional value of z. IP4 provides another fractional solution, and IP3 is pruned as it is infeasible. This process continues, creating sub-problems and refining solutions. The optimal solution reported is the integer solution with the best objective value found first, as in the case of IP6.

3.4 Optimization Techniques in Linear Programming

As the VRP is a generalization of the TSP, it inherits the NP-hard nature of the problem. Consequently, a cutting-edge technique is essential to obtain solutions despite the continuously increasing scale of the problem. This method assists in facing the complexity of the problem, considering the constraints of limited computational power. It enables a frugal allocation of resources, ensuring more efficient problem-solving while avoiding unnecessary resource bottlenecks. Within the realm of optimization techniques, this thesis will specifically concentrate on the Row Generation method and on the Cutting Plane method. These approaches will be employed to create a scalable solution and expedite the computational process to solve MILP formulation of TSP-like and VRP-like problems.

3.4.1 Row Generation Method

A first useful tool to address this challenge is the implementation of the Row Generation Method. This optimization technique, commonly employed in linear programming, proves effective in tackling large-scale problems characterized by a considerable number of constraints [35]. The row generation method retains all variables from the master problem but initially employs only a subset of constraints. Gradually, it verifies whether the generated solution indeed satisfies all the constraints of the problem. If not, it will add more constraints to the problem until no further constraints are violated, hence an optimal solution has been found.

The row generation method steps can be summarized as the following:

1. Initial Integer Master Problem: Formulate the problem with a subset of constraints. Then, solve the problem to obtain an initial solution.
2. Identify Violated Constraints: Examine the initial solution to identify constraints that are violated (not satisfied).
3. Add Violated Constraints: Add some of the identified violated constraints to the initial integer master problem.
4. Reoptimize: Solve the current Integer Master problem to obtain a new solution. Repeat the process of identifying and adding violated constraints until no violation exists, hence an optimal solution is achieved.

This iterative process continues, gradually including more constraints to the problem, until the optimal solution for the entire VRP problem is found. When the number of constraints is not tractable, the Row Generation Method allows the problem to be solved more efficiently by dynamically adding constraints based on the current solution's characteristics. Following the definition adopted by CPLEX [20], the solver that we will use in this thesis, the constraints that are not included by the Integer Master Problem are informally called *lazy constraints*.

3.4.2 Cutting Plane and Valid Inequalities

Cutting-plane method is employed for solving MILP problems by iteratively solving LP problems and refining the feasible region until an optimal integer-feasible solution is achieved. The central idea is to add constraints, known as cutting planes, that tighten the feasible region without excluding the optimal integer solution. To this end, we introduce a distinction of the constraints between "necessary" constraints and valid inequalities. Necessary constraints cannot be removed from the formulation, as otherwise further unfeasible integer solutions would appear in the feasible region. Valid inequalities can be removed from the set of constraints without affecting the set of integer solutions in the feasible region. The step-by-step procedure while implementing cutting plane can be summarized as follows:

1. Initialization: Begin with a formulation including all necessary constraints and only a subset (in case empty) of valid inequalities, and consider its linear relaxation (Relaxed Master Problem)
2. Solving the valid Relaxed Master Problem: Solve the Relaxed master problem to obtain a solution that may not satisfy all valid inequalities.
3. Identifying Violated Valid Inequalities: Identify valid inequalities that are violated by the solution obtained from the master problem.
4. Adding Cutting Planes: Introduce cutting planes corresponding to the violated valid inequalities to tighten the feasible region.
5. Iterative Refinement: Repeat steps 2-4 iteratively until no violated inequalities are identified.

It is important to highlight that the introduced cuts during each iteration serves the primary purpose of narrowing down the feasible region. In the other word, if this method is going to be used, it is crucial to ensure that the Integer Master Problem can still be solved effectively even in the absence of this particular cut. This means that a (further) procedure to deal with the integrality of some variables (e.g. branch and bound) is still needed at the end of the process (unless the class of inequalities considered during separation are able to characterize the convex hull of all integer feasible points). The purpose of incorporating this cut is to expedite the convergence of the integer-aware procedure, e..g., by improving the bounds from linear relaxation in a branch-and-bound approach.

3.4.3 Integrating Row generation and Cutting plane procedures

The cutting-plane method and row generation method plays two disctinct roles, each designed to address specific challenges in solving MILP problems. For visualization, refer to Figure 3.4. Both subfigures present examples of the feasible region in a two-variable integer linear programming scenario. Given the current Integer Master Problem, the polygon outlined in black represents the feasible region of the linear program relaxation, also known as the linear hull or LP hull. This is achieved by relaxing the integrality restrictions while adhering to all functional constraints and variable bounds. The blue dots within the polygon denote integer points, signifying feasible solutions that satisfy the integrality restrictions. The blue polygon, recognized as the integer hull or IP hull, is the smallest convex set that encompasses all integer-feasible solutions. Notably, the IP hull is consistently a subset of the LP hull. Assuming a linear objective function, the optimal solution to the current Integer Master Problem lies at one of the vertices of the IP hull, each of which corresponds to integer points. Additionally, as evident from the graph, the cutting plane does not eliminate any potential integer solutions, as the number of integer points

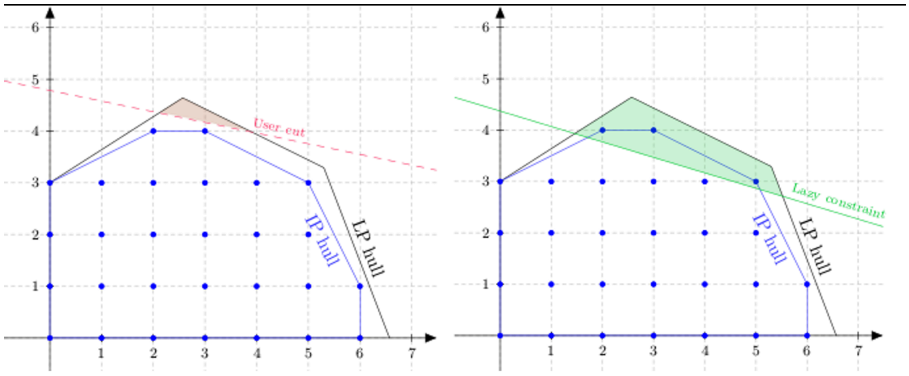

Figure 3.4: Cutting Plane (left) vs Row Generation method (right)

within the IP hull remains unchanged post-cut. Conversely, upon the application of row generation cuts, some integer blue points are no longer included in the set of integer feasible solutions.

3.4.4 Separation Procedures

Both row generation and cutting plane methods require to identify violated constraints or violated valid inequalities. To these end, constraint or cut *separation algorithms* have to be devised.

We now focus on a methodological tool that will be useful to separate violated constraints in the TSP context, which is relevant for this thesis. As we will detail in Chapter 4, MILP formulation for TSP-like problems often contain the so-called *subtour elimination constraints*, aimed at guaranteeing that the solution depicts an hamiltonian cycle. The separation of these constraints can be seen as a *max-flow problem*.

The Max-flow problem can be roughly stated as follows. We consider a network made of several points connected by links. Each link has a limit on the maximum amount of traffic it can handle. Given an origin point and a destination point, the max flow problem asks for determining the maximum amount of traffic that can leave the origin and reach the destination by flowing through the network and without exceeding the link capacities. The problem can model different optimization requests arising in many context, such as, transportation planning, communication network design, resource allocation.

Among the different methods proposed by literature to solve the max-flow problem, we recall the *Ford-Fulkerson algoroithm* [9]. It's a clever strategy that works step by step, gradually increasing the flow until it hits the maximum. To really make this algorithm click, we also need to understand the concept of residual graph.

Consider a network with a feasible flow. The *residual graph*, as illustrated in Figure 3.5, serves as a graphical representation indicating areas where additional flow can be accommodated. The existence of a path from the source to the destination in this graph signifies the potential for additional flow. Each edge on this graphical representation is associated with a numerical value, akin to the available capacity on a road after a certain volume has been transported. This numerical value is referred to as the *residual capacity*, representing the remaining capacity that can be utilized for further flow.

At the outset of the algorithm, we initiate with an empty flow, and the residual map mirrors the original map since, initially, no flow has transpired, rendering every route unobstructed. To identify a viable path for increased

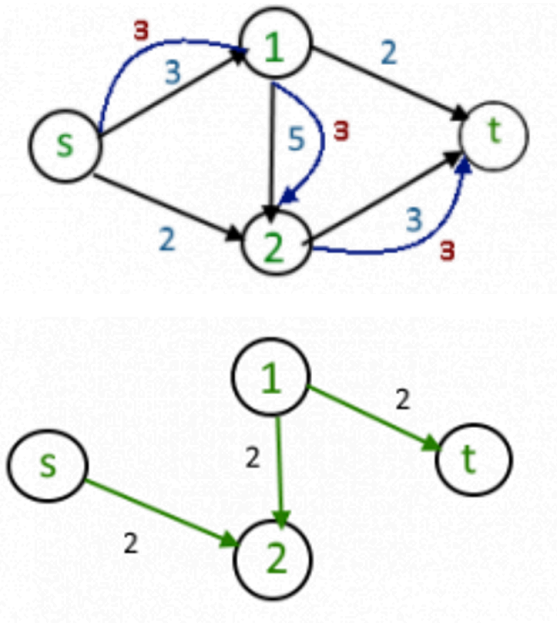

Figure 3.5: [Top] Network Flow yielding the maximum flow. [Bottom] The residual graph

flow, two approaches can be employed: a breadth-first search (BFS) [19] or a more direct route through a depth-first search (DFS) [45]. In this particular instance, the breadth-first search (BFS) methodology is adopted. Not only does BFS determine the existence of a path from the source to the destination, but it also maintains a record of the traversal process. Employing this route map, we ascertain the maximum flow that can be directed along the path by inspecting the minimum capacity encountered. Subsequently, we augment the overall flow with this newly ascertained flow amount.

In the course of determining the maximum flow, the graph is updated along the way. The procedure involves subtracting the transmitted amount from the capacities along the path and increasing the capacities on the reverse routes. The *residual graph* serves as a concealed instrument for identifying avenues where additional flow can be accommodated, and the algorithm serves as a systematic guide to incrementally enhance the flow until reaching the maximum capacity. For a practical demonstration of this process, refer to Figure 3.5.

From the source (s), there is a path that connects s to the sink (t), namely [s -> 1 -> 2 -> t]. The current flow of the patch chosen (highlighted in blue) is 3 units. Considering the original capacity and the amount of capacity left, we will end up with the residual graph shown in the bottom part of Figure 3.5. Since there is no more path that can connect s to t, it can be shown that the current maxflow number is already the absolute max flow number for the network.

3.4.5 Branch-and-cut

The branch-and-bound, row generation, and cutting plane procedures can be integrated into a method to effectively solve MILP models with a very large number of necessary constraints and/or valid inequalities. The

procedure is called *branch-and-cut*: it is essentially the branch-and-bound algorithm where the solution of linear relaxation at each node iteratively considers the constraints and/or the cut separation procedure to identify violated necessary constraints and/or violated cuts. We observe that, here, the separation of both necessary constraints ("lazy constraints") and valid inequalities (cuts) runs after the solution of the current Relaxed Master Problem, which may be fractional or integral. By definition, if we deal with an integer solution, no valid inequality is violated, hence the cut separation procedure is not run. On the contrary, an integer solution may violate some of the necessary constraints that are not included in the current Relaxed Master Problem; hence, the "lazy-constraint" separation procedure has to be run.

3.5 Mathematical Programming Implementation in Docplex

So far, we have seen how powerful mathematical programming is and its wide range of applications to solve real-world problems. To effectively harness the capabilities of mathematical programming, optimization solvers play a crucial role. An optimization solver is a computational tool or piece of software designed to find the best solution to a mathematical optimization problem programmatically. The main goal of an optimization solver is to systematically explore the solution space to identify the optimal solution, which could be the maximum or minimum value of an objective function [30]. The objective function represents the quantity to be maximized or minimized, and it is subject to certain constraints, which are conditions that the solution must satisfy.

Optimization solvers use various algorithms and techniques to search through the feasible solution space efficiently. They can handle linear programming (LP), integer programming (IP), mixed-integer programming (MIP), quadratic programming (QP), nonlinear programming (NLP), and other types of optimization problems. One prominent solver in this domain is CPLEX. IBM ILOG CPLEX Optimization Studio, commonly known as CPLEX [24], stands out as a robust optimization solver widely used for addressing LP, MILP, and even quadratic programming (QP) problems. Developed by IBM, CPLEX offers a comprehensive set of tools and algorithms for solving complex optimization challenges. Its versatility and efficiency make it a preferred choice for tackling real-world problems.

CPLEX provides seamless integration with many programming languages, including C and Python, offering flexibility in implementing mathematical programming solutions. In C, developers can leverage the CPLEX Callable libraries, a set of APIs and libraries that enable efficient interaction with CPLEX solvers. This integration allows for the formulation and solution of optimization problems using the C programming language. Moreover, Python users can benefit from the CPLEX Python API, which facilitates the incorporation of CPLEX capabilities within Python scripts. The 'Docplex' library, an extension of CPLEX based on the Python APIs, further streamlines the integration process, making it convenient for Python developers to construct and solve optimization models using familiar Python syntax.

Utilizing CPLEX within Python brings several advantages. Python's readability and simplicity make it an accessible language for a broader audience, and its extensive ecosystem of libraries enhances its functionality. By combining the power of CPLEX with the ease of Python, users can seamlessly transition from problem formulation to solution, all within a Python environment.

To apply mathematical programming, it's essential to have the Docplex library installed and a valid license for the application [24]. To gain insight into the structure of the complex, we can examine the implementation of the problem, as illustrated in (3.1).

1. **Problem Formulation**: Define your decision variables, objective function, and constraints using the Docplex modeling language. For example:

```
from docplex.mp.model import Model

# Create a model
model = Model(name='IntegerProgrammingExample')

# Define decision variables
x1 = model.continuous_var(name='x1')
x2 = model.continuous_var(name='x2')

# Define objective function
model.maximize(x1 + x2)

# Define constraints
model.add_constraint(2 * x1 + 12 * x2 <= 30)
model.add_constraint(4 * x1 - x2 <= 10)
model.add_constraint(x1 >= 0)
model.add_constraint(x2 >= 0)
```

2. **Solver Configuration**: Configure the solver settings. Docplex allows you to use the CPLEX solver, which is a powerful optimization engine.

```
model.print_information()
model.solve().display()
```

3. **Solution Retrieval**: Retrieve and inspect the solution. The following is the return result upon calling the display function. We can see that the optimal solution is 5, where $x_1 = 3$ and $x_2 = 2$. Aligned with the solution yielded from corner point method.

```
Model: IntegerProgrammingExample
 - number of variables: 2
   - binary=0, integer=0, continuous=2
 - number of constraints: 4
   - linear=4
 - parameters: defaults
 - objective: maximize
 - problem type is: LP
solution for: IntegerProgrammingExample
objective: 5.000
status: OPTIMAL_SOLUTION(2)
x1 = 3.000
```

```
x2 = 2.000
```

CPLEX mainly adopts a branch-and-cut approach to solve MILP problems. In this context, it provides mechanisms to integrate the separation procedures for constraints and cuts in the branch-and-cut schema. The basic tool is the "callback" mechanism, that can be seen as a customizable procedure that is called by the CPLEX branch-and-cut schema. Relevant to this thesis are the "users cut callback" [21] and the "lazy constraint callback" [22]. They are both called at the end of the solution of every Relaxed Master Problem. The former is normally used to separate cuts violated by fractional solutions, and the later is used to separate necessary constraints. We will further detail the use of these callbacks in Chapter 5, devoted to the implementation of the MILP models devised in this thesis.

4

Mathematical Formulation for the Rebalancing Tour Problem

In this chapter, we will translate the business problem as stated in Chapter 2 in a mathematical formulation, in order to solve it pragmatically. As often happens in mathematical modeling, we will start with the simplified version of the problem, and add more complexity incrementally. The first model will focus on the case where e-scooters company can only drop/pick up e-scooters during the night tour operation (neglecting battery concept). The purpose of having this model is to familiarize reader to the night tour problem without having to deal with excessive complication. Once the reader becomes more familiar with the concept, we will add more complexity to the model allowing the scenario of operator swapping batteries during a night tour operation.

4.1 Simple Night Tour Model

The simple night tour model presented in this sction is inspired by Dell'Amico [11] model and solely changes the object from bikes to e-scooter. This model is a state-of-the-art benchmark model because it has been proven that it works with various testing.

We will start the model formulation with the objective function. Since we have established that the night tour route can be represented with graph, the terms docking station, stations, and nodes will be used interchangeably from this point on wards. Moreover, in Chapter 2, we have covered that the cost of operation is reflective to the distance of the tour. Therefore, to minimize the cost of night tour operation, one can simply minimize the distance traveled during the operation. Let binary variable $x_{ij} = 1$ represent a situation when a truck travels from node i to j and c_{ij} is a given distance between node i and j. Objective function of the first model can be written as equation 4.1 given V is the set of the network nodes *including* the depot whereas V_s is the set of network nodes

representing the stations and *excluding* the depot.

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{4.1}$$

Let's consider the e-scooters night rebalancing problem under the following operational assumptions:

1. all nodes must be visited and can only be visited once;
2. a truck departs from and arrives at the same depot;
3. the truck only transports **full-charge** e-scooters. That means, operator will neglect any low-battery e-scooter
4. each truck transports up to a maximum number of e-scooters available in the depot;
5. for each station, we are given:
    - the initial number of full-battery e-scooters on ground before rebalancing;
    - the required balanced number of full-battery e-scooters on ground after rebalancing;
6. when a truck visits a station, it can make any of the following operations:
    - pick-up full-charge e-scooters;
    - drop-off full-charge e-scooters;
7. e-scooters cannot be charged on-board.

In order to ensure that these operational assumptions are satisfied, they will be translated into set of constraint in the model. The first and second assumption can be written as (4.2) and (4.3) respectively. Given m is the number of available trucks for the night tour operation, equation 4.4 regulate the number of trucks that will be used.

$$\begin{aligned} \sum_{i \in V} x_{ij} &= 1, \forall j \in V \\ \sum_{j \in V} x_{ij} &= 1, \forall i \in V \end{aligned} \tag{4.2}$$

$$\sum_{j \in V_s} x_{0j} = \sum_{i \in V_s} x_{i0}. \tag{4.3}$$

$$\sum_{j \in V} x_{0j} \leq m \tag{4.4}$$

Unlike assumption 1 and 2, assumption 3 needs to be stated in order to simplify the operation. This kind of assumption does not directly translate to mathematical constraint, but help the model to shape another constraint.

Suppose assumption 3 is omitted, the model now have to have more variables to represent commodity in the vehicle. For example, it is possible to divide the battery into 2 groups: low battery and full charge. It may not look like a big deal, but imagine there are a lot of nodes in the city. The size of the model will sensibly grow by having an additional variable. Model needs to calculate the changes of variable in each node and make sure it does not violate other constraint.

Assumption 4 is needed to ensure that the night tour has a realistic solution. A company must have limited number of e-scooter. Without this assumption, model could generate a solution such as to bring a thousand e-scooters from the depot even though the company only have one hundred e-scooters in total. To translate this requirement, new variable θ_i is introduced to represent the number of e-scooters in the truck after leaving station i, or the depot if $i = 0$. Thus, assumption 4 can be written as (4.5).

$$0 \leq \theta_0 \leq \text{number of e-scooters in depot} \tag{4.5}$$

Since the model is given both the initial and the required number of full-battery e-scooters, the needs of each station can be computed.

Suppose q_i = ideal number of e-scooters in station i− number of e-scooters in station i denote the deviation of station i. When $q_i > 0$, it means station i need q_i e-scooters so operator need to drop-off q_i e-scooter. On the other hand, when $q_i < 0$, it means station i has too many e-scooters. Therefore, operator need to pick up q_i e-scooters.

While traversing the stations, company needs to ensure the capacity of the truck is not overfill. This implies that the model needs to keep track the load of the truck on each station. Suppose a truck is going from station i to station j, the load of the truck at station j is depending on q_j. This relationship can be translated as the following:

$$\theta_j = (\theta_i - q_j)x_{ij}, i \in V, j \in V_s$$

These two equations are needed to ensure that when a truck is traveling from i to j, $x_{ij} = 1$, $\theta_j = \theta_i - q_j$. However, these constraints are non-linear. To deal with this situation, we can apply linearization method by applying "Big M" method [5]. The "Big M" represents a large positive number, and it's used to make constraints conveniently redundant or not, depending on the value of some variables. The linearized version of the constraints represented in equation 4.6.

$$\begin{aligned} \theta_j &\geq \theta_i - q_j - M(1 - x_{ij}), i \in V, j \in V_s \\ \theta_i &\geq \theta_j + q_j - M(1 - x_{ij}), i \in V, j \in V_s. \end{aligned} \tag{4.6}$$

Now that the model already tracks the updated load along the tour, we can regulate the load of the truck at all times. When a truck visits a node j, from operational perspective, the truck load can be as low as 0, which is when truck does not have any more e-scooters to drop-off. On the other hand, the load of the truck can be as high as its maximum capacity C. To make the bound tighter, we can benefit from the fact that $\theta_j = \theta_i - q_j$. Hence, the

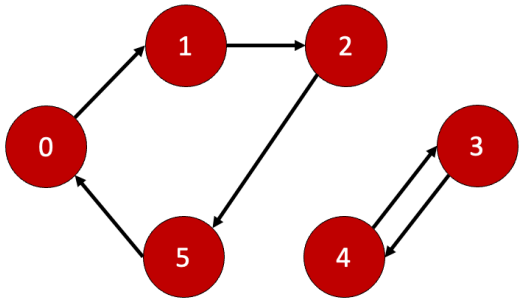

Figure 4.1: Example of sub tour

lower and upper bounds of the load can be written as equation 4.7.

$$\max(0, q_j) \leq \theta_j \leq min(C, C + q_j), j \in V \tag{4.7}$$

To ensure that truck reached all the nodes upon departure from the depot, we need to add continuity constraint (equation 4.8) to the model. Consider Figure 4.1 which contain example of a sub-tour. Suppose depot represent by node 0, it can be seen that in this tour, truck will never reach station 3 and 4 causing station 3 and 4 remain unbalanced at the end of night tour operation. In optimization terms, this continuity condition is also known as *sub-tour elimination constraint*.

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, S \subseteq V_s, S \neq \emptyset \tag{4.8}$$

Now that readers have the reasoning behind the model, we summarize objective function and constraint for

the Simple Night Tour Model.

$$
\begin{aligned}
\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \\
\sum_{i \in V} x_{ij} = 1, \forall j \in V \\
\sum_{j \in V} x_{ij} = 1, \forall i \in V \\
\sum_{j \in V_s} x_{0j} = \sum_{i \in V_s} x_{i0} \\
\sum_{j \in V} x_{0j} \leq m \\
0 \leq \theta_0 \leq \text{number of e-scooters in depot} \\
\theta_j \geq \theta_i - q_j - M(1 - x_{ij}), i \in V, j \in V_s \\
\theta_i \geq \theta_j + q_j - M(1 - x_{ij}), i \in V, j \in V_s. \\
\max(0, q_j) \leq \theta_j \leq min(C, C + q), j \in V \\
\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, S \subseteq V_s, S \neq \emptyset \\
x_{ij} \in \{0, 1\}
\end{aligned}
\tag{4.9}
$$

4.2 An Illustrative Example of Simple Night Tour Model

In this section, we will see how the model works by utilizing simple dummy data. Assume we are presented with a specific logistical challenge involving three stations, the distances between pairs of these stations, deviations from a standard route, and the availability of a single service vehicle (equation 4.10). The primary objective here is to identify the most efficient route that can adequately address the requests and demands of these stations. This task is not only about minimizing travel distance but also about planning the use of the service vehicle to fulfill station requirements.

To simplify the problem for illustrative purposes, we make the assumption that the distances between stations are symmetrical. In other words, if the distance between station A and station B is represented as $d(s1, s2)$, it is assumed to be equal in both directions, $d(s1, s2) = d(s2, s1)$. This symmetry simplifies the route calculation and aids in providing a clear example. However, it's essential to recognize that in real-world scenarios, various factors such as road constraints, traffic conditions, or road quality may result in different distances between stations depending on the direction of travel. These complexities are often considered in more advanced route optimization

models to ensure accurate and practical solutions.

$$
\begin{aligned}
S &= 1, 2, 3 \\
V &= 0, 1, 2, 3 \\
m &= 1 \\
d(0,1) &= d(1,0) = 2km \\
d(0,2) &= d(2,0) = 4.5km \\
d(0,3) &= d(3,0) = 3km \\
d(1,2) &= d(2,1) = 3.5km \\
d(1,3) &= d(3,1) = 2.5km \\
d(2,3) &= d(3,2) = 1km \\
q_1 &= 4 \\
q_2 &= -6 \\
q_3 &= -8 \\
C &= 20 \\
\theta_0 &\leq 10
\end{aligned}
\tag{4.10}
$$

By carefully considering all the different ways stations can be visited, we end up with a total of six potential routes that cover all the stations. For simplicity of demonstration purposes, we will restrict the model to always pick-up 10 e-scooters from the depot. Table 4.1 summarized the cost associated with each route. This thorough approach enables us to explore and assess every possible order in which the stations can be visited, giving us a complete picture of the available route choices.

This in-depth analysis of these routes not only helps us finding the most efficient and cost-effective options but also deepens our understanding of how stations interact and the best way to visit them. It provides us with valuable insights to make well-informed decisions and optimize our routes in this specific situation.

The computations presented reveal that among the considered routes, route1, route2, route4, and route6 emerge as feasible options. In contrast, route3 and route5 prove to be unfeasible since they result in the vehicle load exceeding the capacity threshold. Consequently, the optimal choice for the night route is route1 due to its exceptional cost-efficiency, boasting the minimum cost among all the feasible routes, represented by cost1, amounting to 9.5 kilometers.

It's worth noting that the complexity of manual calculation grow significantly as we are dealing with more stations, since the number of possible tours to visit n stations is $n!$. Therefore, we will see how to build an automation tool that can generate the solution, given the model (4.9) and its inputs (4.10), in Chapter 5.

Route Number	Route	θ	Cost (km)	Is feasible?
1	[0,1,2,3,0]	[10,6,12,20,20]	9.5	Yes
2	[0,1,3,2,0]	[10,6,14,20,20]	10	Yes
3	[0,2,3,1,0]	[10,16,24,20,20]	10	No, because $\theta_2 > C$
4	[0,2,1,3,0]	[10,16,12,20,20]	13.5	Yes
5	[0,3,2,1,0]	[10,18,24,20,20]	9.5	No, because $\theta_2 > C$
6	[0,3,1,2,0]	[10,18,14,20,20]	13.5	Yes

Table 4.1: Route permutation and its associated cost

4.3 Night Rebalancing Tour Problem with Battery Swaps (NRTP-BS)

The second model is formulated with two goals. The first one is to allow battery swap during night tour operation. Company found this option is necessary since it can save them operation cost. Furthermore, in practical situations, achieving a perfectly rebalanced distribution may not always be feasible. Instead of declaring a solution infeasible, we aim to offer the company "the next best" solution, which is the second goal of our enhanced model. Let s_j represent the number of e-scooters that are either in excess (if positive) or missing at station j after the night tour operation. We will refer to this variable as the imbalance variable. To encourage the model not to generate imbalanced solutions unnecessarily, even when a perfectly balanced solution is achievable, we will add the absolute value of the imbalance variable into the objective function, combined with a positive unit penalty $P_j \in [0, P_{MAX}]$ where P_{MAX} is the maximum penalty allowed. This penalty reflects our inclination to prioritize either achieving a well-balanced distribution as the tour outcome or opting for shorter routes, even if it means accepting an imbalanced result. As the unit penalty approaches zero, the model becomes more adaptable in accommodating imbalanced solutions. In contrast, by setting the penalty closer to P_{MAX}, the model places a stronger emphasis on attaining a balanced outcome, even at the cost of prolonging the night tour operation.

The objective function for this model can be written as equation 4.11.

$$\min \sum_{j \in V_s} P_j |s_j| + \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{4.11}$$

As for the operational assumptions, the same operational assumptions in the previous model will be re-used with slight modification:

- In assumption 3, we allow truck to carry more than just full-charge e-scooter, but also low-battery e-scooters, and full charge batteries;
- in assumption 5, we get more additional information:
  - the initial number of low-battery e-scooters on ground before rebalancing;
  - the final desired number of full-battery e-scooters on ground after rebalancing;

- in assumption 6, there are more variation to the operation:

    1. pick-up full-charge e-scooters;
    2. pick-up low-battery e-scooters;
    3. drop-off full-charge e-scooters;
    4. swap the battery of low-battery e-scooters on-board and drop them off;
    5. swap the battery of low-battery e-scooters on-site.

The first 4 model constraints introduced in the Simple Night Tour model (equation 4.2 - equation 4.4) and sub tour elimination constraint (4.8) will be added into the Battery Swap model constraints. Nonetheless, the battery swap model discussed in this chapter will be constrained to the use of only one truck. This constraint is introduced to maintain simplicity and avoid overwhelming readers with a potentially extensive growth in the number of variables. For instance, if we were to consider a scenario involving three trucks, the complexity would escalate significantly. Not only would there be an increase in the number of variables due to the different types of loads—full charge e-scooters, low battery e-scooters, and spare batteries—but we would also need additional variables to track the quantity of each load in each truck. Consequently, for the sake of simplicity and clarity, we have opted to restrict the model to a single truck.

Let q_j be the ideal number of full-battery e-scooters that needs to be dropped off from station j (picked up, if negative) to achieve the desired number after rebalancing. Moreover, let w_j^F be the number of full-battery e-scooters that operator needs to drop off (if positive, pick up if negative) in order to rebalance the station j. Let w_j^L be the number of low batter e-scooters to drop-off at station j after swapping their batteries (operation 4). Lastly, let w_j^B be the number of battery swaps for low battery e-scooters that are already at station j (operation 5). The night tour operation at station j then can be stated as equation 4.12.

$$\begin{aligned} q_j + s_j = w_j^F + w_j^L + w_j^B, \forall j \in V_s \\ w_j^L, w_j^B, y_j^L \geq 0, \forall j \in V_s \end{aligned} \tag{4.12}$$

In reality, operator can pick up as much as full battery e-scooters that exist in the station and drop off as much as full battery e-scooters that exists in the truck (equation 4.14). Let F_j and L_j be the number of full battery and low battery e-scooters that presents in station j. Furthermore, let θ_j^F, θ_j^L, and θ_j^B be the number of full-charge e-scooter, low-battery e-scooter, and full charge spare battery on board after visiting node j respectively. These variables will always have value greater than 0 (equation 4.13). Operator can also pick up as much as low battery e-scooters that exist in the station (equation 4.15). Another alternative to rebalance the station is to swap low battery e-scooters with full charged battery (operation 4 and 5). Operator can opt for this choice as much as long as we still have full-charged spare battery in the truck (equation 4.17). Moreover, to ensure that the station is not cluttered with low battery e-scooters, operator can pick up low battery e-scooters y_j^L at station j (operation 2).

$$\theta_j^F, \theta_j^L, \theta_j^B \geq 0, \forall j \in V \tag{4.13}$$

$$\begin{aligned} w_j^F &\geq -F_j, \forall j \in V_s \\ w_j^F &\leq \theta_i^F + M(1 - x_{ij}), \forall i \in V, j \in V_s \end{aligned} \tag{4.14}$$

$$w_j^L \leq \theta_i^L + M(1 - x_{ij}), \forall i \in V, j \in V_s \tag{4.15}$$

$$w_j^L + w_j^B \leq \theta_i^B + M(1 - x_{ij}), \forall i \in V, j \in V_s \tag{4.16}$$

$$y_j^L + w_j^L \leq L_j, \forall j \in V_s \tag{4.17}$$

To ensure that the truck is not overloaded, we need to keep track the truck load throughout the operation. When leaving the depot, a truck can load as much as full charged e-scooter, low charge e-scooter, and spare battery that exist in the depot (equation 4.18). The progression of full battery e-scooter, low battery e-scooter, and spare battery in the truck at every station can be described by equation 4.19, 4.20, and 4.21, respectively.

$$\theta_0^F \leq F_0, \theta_0^L \leq L_0, \theta_0^B \leq B_0 \tag{4.18}$$

$$\begin{aligned} \theta_j^F &\geq \theta_i^F - w_j^f - M(1 - x_{ij}), \forall i \in V, j \in V_s \\ \theta_j^F &\leq \theta_i^F - w_j^f + M(1 - x_{ij}), \forall i \in V, j \in V_s \end{aligned} \tag{4.19}$$

$$\begin{aligned} \theta_j^L &\geq \theta_i^F - w_j^L + y_j^L - M(1 - x_{ij}), \forall i \in V, j \in V_s \\ \theta_j^L &\leq \theta_i^F - w_j^f + y_j^L + M(1 - x_{ij}), \forall i \in V, j \in V_s \end{aligned} \tag{4.20}$$

$$\begin{aligned} \theta_j^B &\geq \theta_i^B - w_j^L - w_j^B - M(1 - x_{ij}), \forall i \in V, j \in V_s \\ \theta_j^B &\leq \theta_i^B - w_j^L - w_j^B + M(1 - x_{ij}), \forall i \in V, j \in V_s \end{aligned} \tag{4.21}$$

We are now managing two distinct load types in the truck: battery and e-scooter. Thus, it is essential to consider two distinct truck's maximum capacity. We will represent the maximum e-scooters capacity in the truck as C and the maximum battery capacity as C^B. Additionally, we must establish upper bounds for these variables to ensure that the truck is not overloaded (equation 4.22 and 4.23).

$$\theta_j^F + \theta_j^L \leq C, \forall j \in V \tag{4.22}$$

$$\theta_j^B \leq C^B, \forall j \in V \tag{4.23}$$

In conclusion, this model is defined by the objective function outlined in equation 4.11, along with a set of

constraints ranging from equation 4.2 - 4.4, and from Equation 4.12 - 4.23. These equations collectively shape the framework for addressing the intricate challenges of our problem, guiding us toward optimized solutions that balance the operation and battery requirements effectively.

Now that reader have the reasoning behind the model, we summarize objective function and constraint for the NRTP-BS model in Figure 4.2.

4.4 An Illustrative Example of Battery Swap Model

For better clarity of the NRTP-BS model, let us revisit the previous illustrative example (as represented in Equation 4.10) and apply the battery swap model. To achieve this, we will make a slight adjustment to the original situation by declaring demand of each station, the number of full and low battery e-scooters that exist in each station, also the the unit penalty on each station (4.24).

$$
\begin{aligned}
C &= 10 \\
C^{\mathcal{B}} &= 10 \\
F_0 &= 10 \\
B_0 &= 10 \\
q &= [0, 4, -6, -8] \\
\text{Full battery e-scooters in each station} &= [0, 6, 16, 18] \\
\text{Low battery e-scooters in each station} &= [0, 0, 0, 0] \\
\text{scneario 1} : P &= [0, 0, 0, 0] \\
\text{scneario 2} : P &= [10, 10, 10, 10]
\end{aligned}
\tag{4.24}
$$

In this specific case, battery swap model is using 46 unique variables, whereas the simple night tour is only using 21 variables. It is important to highlight that, both models dealing with the same number of nodes, they both employ 16 variables to depict the connections between these nodes. Notably, the basic model only requires 5 variables to characterize the station load at each location, while the battery swap model utilizes a more substantial 30 variables for this purpose. This example already demonstrates that the increased flexibility offered by the battery swap model comes at the cost of higher computational resources and complexity.

We will not delve into the laborious manual calculation of all possible combinations of variable values, that here significantly includes also the setting for node-operations related variables. Instead, we shed light on the significant impact of the penalty parameter. In this example of scenario 1, we have chosen to prioritize shorter distances even if it results in a slight imbalance in e-scooters distribution. Lets explore an additional scenario to illustrate this concept.

In the first scenario, where we prioritize minimizing the distance ($P_j = 0$), we opt for a route that starts at the depot, carrying 4 fully charged e-scooters, drop 4 fully charge e-scooters in station 1, then does nothing while visit station 2 and 3. This choice results in a night tour spanning 3.5 kilometers. From an operational perspective, at least in the analysed case, the proposed solution this approach makes sense. It does not compromise the customer

$$\min \sum_{j \in V_s} P_j |s_j| + \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij}$$

$$\sum_{i \in V} x_{ij} = 1, \forall j \in V$$

$$\sum_{j \in V} x_{ij} = 1, \forall i \in V$$

$$\sum_{j \in V_s} x_{0j} = \sum_{i \in V_s} x_{i0}$$

$$\sum_{j \in V} x_{0j} \leq m$$

$$Q_j + s_j = w_j^F + w_j^L + w_j^B, \forall j \in V_s$$

$$w_j^L, w_j^B, y_j^L \geq 0, \forall j \in V_s$$

$$\theta_j^F, \theta_j^L, \theta_j^B \geq 0, \forall j \in V w_j^F \geq -F_j, \forall j \in V_s$$

$$w_j^F \leq \theta_i^F + M(1 - x_{ij}), \forall i \in V, j \in V_s$$

$$w_j^L \leq \theta_i^L + M(1 - x_{ij}), \forall i \in V, j \in V_s$$

$$w_j^L + w_j^B \leq \theta_i^B + M(1 - x_{ij}), \forall i \in V, j \in V_s$$

$$y_j^L + w_j^L \leq L_j, \forall j \in V_s$$

$$\theta_0^F \leq F_0, \theta_0^L \leq L_0, \theta_0^B \leq B_0$$

$$\theta_j^F \geq \theta_i^F - w_j^f - M(1 - x_{ij}), \forall i \in V, j \in V_s$$

$$\theta_j^F \leq \theta_i^F - w_j^f + M(1 - x_{ij}), \forall i \in V, j \in V_s$$

$$\theta_j^L \geq \theta_i^F - w_j^L + y_j^L - M(1 - x_{ij}), \forall i \in V, j \in V_s$$

$$\theta_j^L \leq \theta_i^F - w_j^f + y_j^L + M(1 - x_{ij}), \forall i \in V, j \in V_s$$

$$\theta_j^B \geq \theta_i^B - w_j^L - w_j^B - M(1 - x_{ij}), \forall i \in V, j \in V_s$$

$$\theta_j^B \leq \theta_i^B - w_j^L - w_j^B + M(1 - x_{ij}), \forall i \in V, j \in V_s$$

$$\theta_j^F + \theta_j^L \leq C, \forall j \in V$$

$$\theta_j^B \leq C^B, \forall j \in V$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, S \subseteq V_s, S \neq \emptyset$$

$$x_{ij} \in \{0, 1\}$$

Figure 4.2: MILP formulation for the NRTP-BS

experience because users will find fully stocked e-scooters at the stations the next morning. This not only enhances the user experience but also bolsters marketing efforts by instilling trust through the visible availability of ample e-scooters. Nevertheless, we stress that this is a lucky situation and, moreover, having more scooters than needed may cause other operational issues from, e.g., the operator perspective (this is why we also penalize excess e-scooters).

Furthermore, consider a different perspective—favoring a balanced distribution, even if it means taking a longer night route. In scenario 2 where $P_j = 10$, the best solution involves a route that begins at the depot with 4 full e-scooters, stopping at station 1 to drop-off 4 full charged e-scooters, proceeding to station 2 to pick up 6 full charged e-scooters, and finally visiting station 3 to collect 8 e-scooters. This example proves that, even at the cost of traveling greater distances, we can achieve a perfectly balanced e-scooters distribution. This can be particularly valuable in situations where we need to place e-scooters in less secure areas, aiming to release as few e-scooters as possible while still meeting user demand. In essence, the unit penalty here plays a pivotal role and can be strategically employed based on the specific use case, providing a versatile tool for addressing varying operational needs.

5 Models Implementation in Docplex

The goal of this chapter is to tackle and resolve the optimization problems presented in Chapter 4: the "Simple Night Tour Model" and the "NRTP-BS." These models are designed to assist an e-scooter company in optimizing their night tour operations. To achieve this, we will leverage the capabilities of IBM Decision Optimization CPLEX, using the interface "Docplex". By using this API, we will write Python code to formulate and solve these complex optimization problems.

Docplex offers an efficient and accessible means to address a broad spectrum of optimization tasks, including linear programming, mixed-integer linear programming, quadratic programming, and more. Its power lies in its user-friendly Python interface, which allows for the straightforward creation of decision variables, the definition of objective functions, and the addition of constraints. As highlighted in Chapter 2, with Docplex' natural, algebraic syntax for expressing optimization models, both experts and those new to mathematical optimization can effectively define and solve real-world problems.

5.1 Implementation of the Simple Night Tour Model

This section presents the pseudo-code for implementing the simple night tour model. In Chapter 4, we have discussed that model presented in equations 4.9 and Figure 4.2 contains subtour elimination constraints whose number grows exponentially with the number of stations. This fact makes the problem not directly tractable. In order to overcome this issue, we will implement the row generation method presented in Chapter 3. The flow of this algorithm is summarized in Figure 5.1.

However, with an increasing number of stations, the cost of recalculating the solution rises when a subtour is detected within the current optimal solution. Under the hood, Docplex utilizes the branch and bound method while solving the problem, as elaborated in Section 3.5. By leveraging this observation, we can enhance the algorithm's efficiency to separate the infeasible solution earlier. The idea is to create a separation mechanism at each

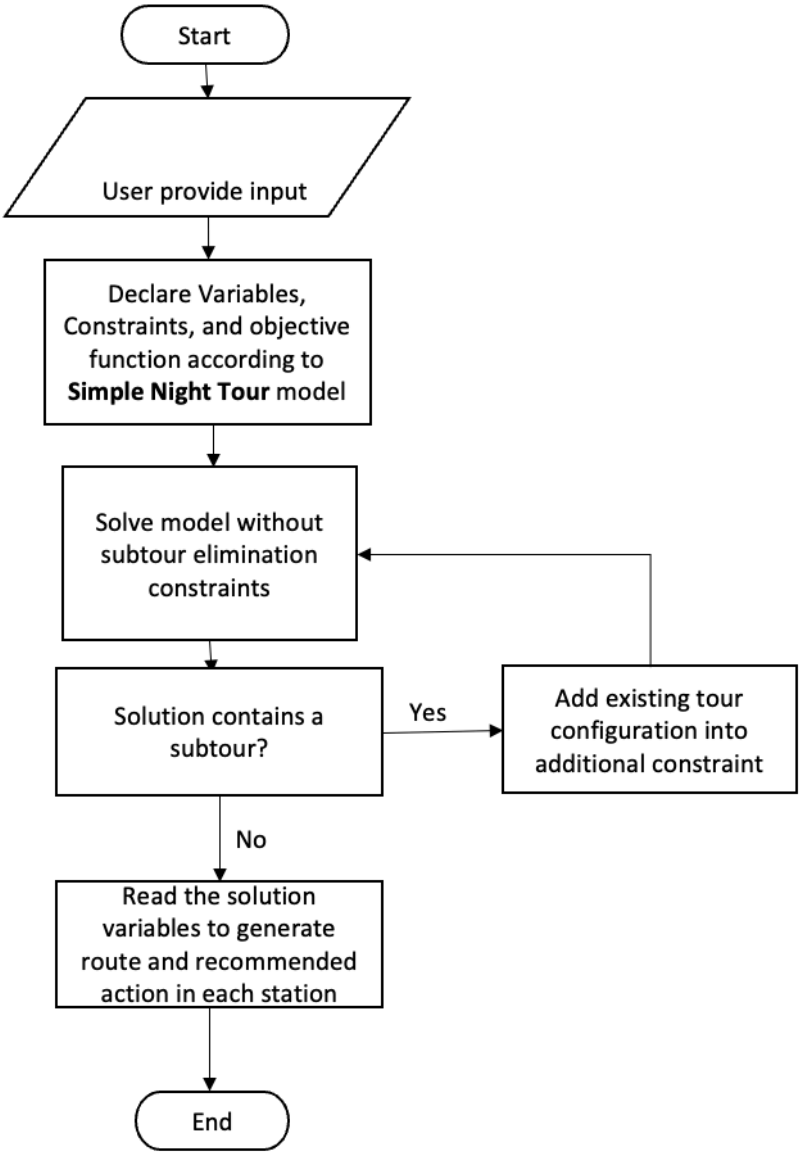

Figure 5.1: Simple Night Tour with row generation implementation flow

branch-and-bound node as soon as a new integer-feasible solution is found. In Docplex, this can be done by utilizing the callback class, namely *LazyConstraintCallback*. It is a subclass of the more general CplexCallback class, which provides a framework for implementing various types of callbacks in CPLEX. To ensure that constraint (4.8) is respected in the final solution in a branch-and-cut fashion, the callback invokes a *separation procedure* to identify the violated constraints. The lazy constraint callback is used to process solutions that satisfy the integrality requirement. We can take advantage of this property to devise a separation procedure that works on a graph where tours or subtours can be directly identified. First, the algorithm needs to translate the current solution into a readable tour. This process can be done by following the following steps:

1. Analyze the solution of variable x.
2. Build a route by examining the key of variable x where $x_{ij} = 1$.
3. Start by finding j where $x_{0j} = 1$. This j will be the next station the truck will head to.
4. Repeat the same step until $x_{j0} = 1$. This indicates the last station before the truck returns to the starting station.

After the algorithm obtained a readable cycle, it should identify weather this cycle violates the subtour elimination constraint or otherwise. In case of violation, we will tell the model to never generate this solution again by adding the current configuration solution as an additional model constraint. The code for the *subtour separation* step can be formulated as follows:

```
if len(route) < self.num_stations +2:
    print('Subtour exists. Recalculating the tour')
    bad_tour = 0
    for i in range(len(route)-1):
        bad_tour += self.x[route[i], route[i+1]]
        converted = self.linear_ct_to_cplex(bad_tour <= len(route)-2)
        self.add(converted[0] , converted[1],converted[2])
        self.lazy_ct+=1
```

When a cycle contains no subtour, the route's length will be equivalent to the number of stations plus 2. For instance, if there are 3 stations, and the generated route is $0- > 1- > 2- > 3- > 0$, this specific route doesn't encompass any subtours, as evidenced by its length being 5, which is equal to 3 (number of stations) plus 2. In the event of a subtour exists, we need to add a constraint into the lazy constraint pool to avoid model generate the same solution in the next iterations. The additional constraint will take the form described in equation 5.1. For example, if there are 3 stations and the generated route is only $0- > 1- > 0$, a subtour is identified. Therefore, we will include the constraint $x[0,1] + x[1,0] <= 1$ in the pool of lazy constraints.

$$\sum_{i=0,j=0}^{n} x_{ij} \leq \text{length of subtour} - 2 \tag{5.1}$$

In summary, the flow of implementation for the Simple Night Tour with "LazyConstraintCallback()" is illustrated in Figure 5.2.

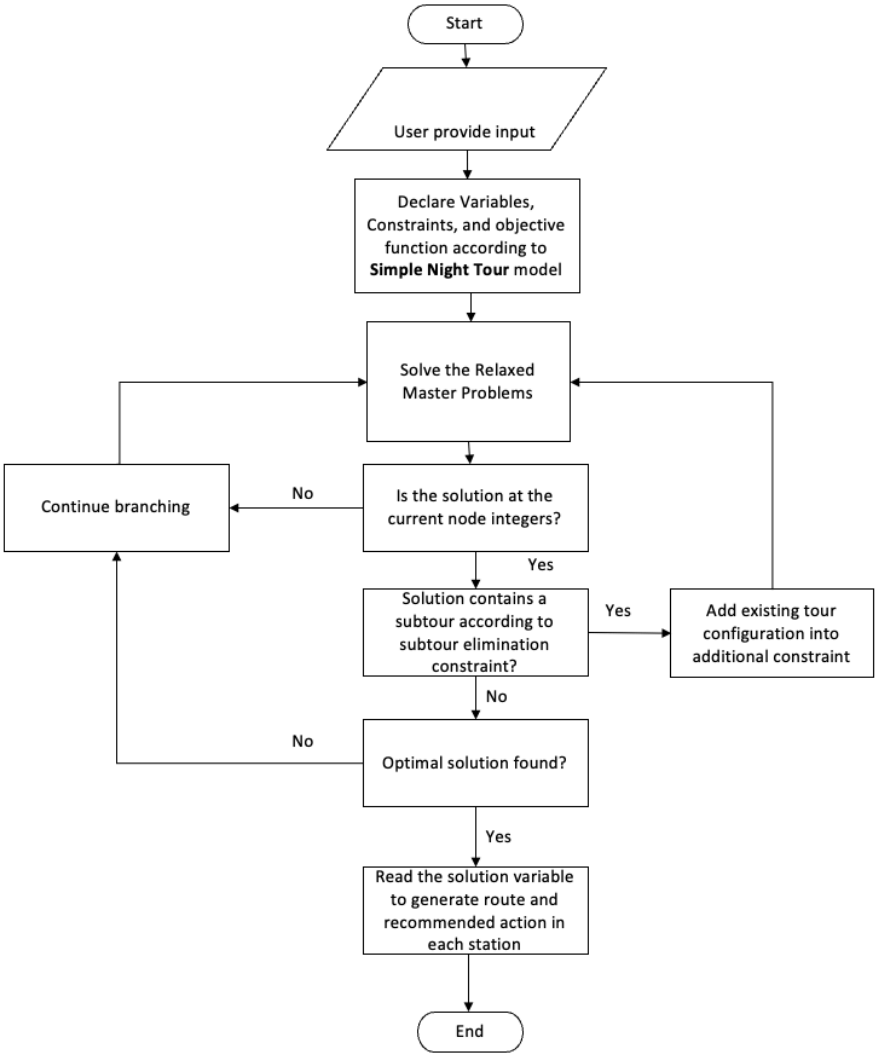

Figure 5.2: Simple Night Tour with LazyConstraintCallback implementation flow

```
      Nodes                                         Cuts/
 Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap         Variable B NodeID Parent
 Depth

    0     0        7.7500     4                      7.7500       10
    0     0        9.5000     8                     Cuts: 6       14
Lazy callback initiated
no subtour located. Presenting the solution

Implied bound cuts applied:  1
Mixed integer rounding cuts applied:  1
Zero-half cuts applied:  2
Lift and project cuts applied:  1
Gomory fractional cuts applied:  1

Root node processing (before b&c):
  Real time             =    0.01 sec. (0.61 ticks)
Sequential b&c:
  Real time             =    0.00 sec. (0.00 ticks)
                         ------------
Total (root+branch&cut) =    0.01 sec. (0.61 ticks)
Number of lazy constraint added = 0
The route for night rebalancing is: [0, 1, 2, 3, 0]
The total journey is 9.5 km
From the depot, bring 4 scooter and 0 battery
In station 1, drop off 4 full battery scooter, pick up 0 low battery scooter, drop 0 low battery scooter, and
exchange 0 battery
In station 2, pick up 6 full battery scooter, pick up 0 low battery scooter, drop 0 low battery scooter, and
exchange 0 battery
In station 3, pick up 8 full battery scooter, pick up 0 low battery scooter, drop 0 low battery scooter, and
exchange 0 battery
```

Figure 5.3: Snapshot of the generated result

To better understand the implementation process in Docplex, let us revisit the same example discussed in section 4.2. After declaring the variables, constraint, and objective function according to the model, we can solve the model by invoking the following command. The snapshot of the result is presented in figure 5.3.

During the optimization of mathematical programs by ILOG CPLEX, numerous underlying processes occur behind the scenes. Throughout the optimization process, CPLEX constructs the branch-and-bound tree with the linear relaxation of the original problem at the root and subproblems to optimize at the nodes of the tree. CPLEX updates its progress in optimizing the original problem in a node log file while building and traversing this tree. In the given example, CPLEX achieved the optimal objective function value of 9.5 km by exploring 0 nodes and performing 14 cumulative iterations to solve the generated subproblems. Notably, to reach the final solution, CPLEX autonomously added various general-purpose cuts without requiring user-specific configurations. In this instance, these six cuts include implied bound cuts, mixed-integer rounding cuts, zero-half cuts, lift and project cuts, and Gomory fractional cuts; no user cuts were defined. Beyond determining the route, we can also derive suggested actions at each station by considering the θ variable. For this example, the suggested action is to bring 4 full charge e-scooters from the depot, drop 4 off at station 1, pick pick-up 6 full-battery e-scooters at station 2, and finally pick-up 8 full-battery e-scooters at station 3. This recommendation aligns with the solution generated in chapter 4.2. The overall time taken to solve this example was 0.01 seconds.

5.2 Implementation of NRTP-BS

NRTP-BS model is formulated to allow more operational choice during the night tour operation. However, this flexibility comes with the caveat that this model requires many more variables and constraints in comparison to the Simple Night Tour model. To address this issue, we try to improve the efficiency of the branch-and-cut approach implemented by CPLEX by separating the subtour elimination not only when solutions are integral but even when the solution of the current Relaxed Master Problem is fractional. In this way, we can anticipate the addition of relevant constraints at the early stages of the process and, overall, save computational resources (see [34]). To this end, we use the "users cut callback" mechanism to invoke a specialized separation procedure able to identify violated subtour elimination constraints even if the current solution contains fractional values. Mirroring CPLEX, doplex provides a mechanism to seamlessly incorporate the cutting plane method into the algorithm through the UserCutCallback(). This callback will be used within the cut loop that CPLEX calls at each node of the branch and cut algorithm. It will be called once after CPLEX has ended its own cut generation loop so that the user can specify additional cuts to be added to the cut pool.

Upon calling the users cut callback mechanism, there is a possibility that the current solution contains fractional values. Therefore, no route can be identified by the subtour elimination procedure that has been explained in Section 5.1. To address this challenge, we can revisit the maxflow problem explained in Chapter 3. By considering the solution at the current node, we build a network where nodes correspond to stations (including depots), and each arc (i, j) has a capacity of x_{ij}. We then explore all possible pairs of source and sink among the number of stations. In each iteration, we check if the maxflow is less than 1. If maxflow is less than 1, it indicates the presence of a node that is not connected to other nodes.

We observe that subtour elimination can be seen as a connectivity requirement, asking for a flow of at least one between any two pairs of nodes in the graph. According to the max-flow min-cut property, this is equivalent to saying that, for any partition of the node set N into two subsets S and $N_s = N \setminus S$, the sum of the capcities on the arcs from any node in S to any node in N_S must be at least one. This in turn means that subtour elimination constraints (4.8) can be equivalently stated as follows:

$$\sum_{i \in S, j \in N \setminus S} x_{ij} \geq 1, \forall S \subset V_S, S \notin \emptyset \tag{5.2}$$

We can utilize the residual information from the maxflow to identify nodes that lack connections to others. Consider the example of Figure 4.1, a subtour in this example here includes nodes $0, 1, 2$, and 5, forming the set S, and hence N_S contains nodes 3 and 4. Consequently, the added constraint is essentially the inverse of how we identify the subtour. For the given example, the added constraint is $x[0,3] + x[0,4] + x[1,3] + x[1,4] + x[2,3] + x[2,4] + x[5,3] + x[5,4] \geq 1$. We want to emphasize that the provided separation procedure also works with fractional variables, in which case a subtour is not directly visible on the corresponding network.

When there is no (5.2) identified as violated, the solution at the current node is considered as a candidate. However, unlike in the row generation procedure depicted in Figure 5.2, we can not immediately infer that this solution is the final solution. The algorithm still needs to find the integer solution, and furthermore, this integer solution needs to be checked to determine if the generated tour contains a subtour or not, by invoking the LazyConstraintCallback. In general, since we have conducted an extensive separation process while solving the

relaxed integer problem, we usually converge to the final solution by the time LazyConstraintCallback is invoked. Nevertheless, this process does not completely eliminate the possibility of a subtour existing in this situation. It is important to note that the implementation of LazyConstraintCallback in this problem can stand alone, while the implementation of UserCutCallback requires LazyConstraintCallback to ensure that no subtour is contained in the final solution.

The implementation flow that processes a branch-and-bound node with use of LazyConstraintCallback and UserCutCallback for Battery Swap model is summarized in Figure 5.4.

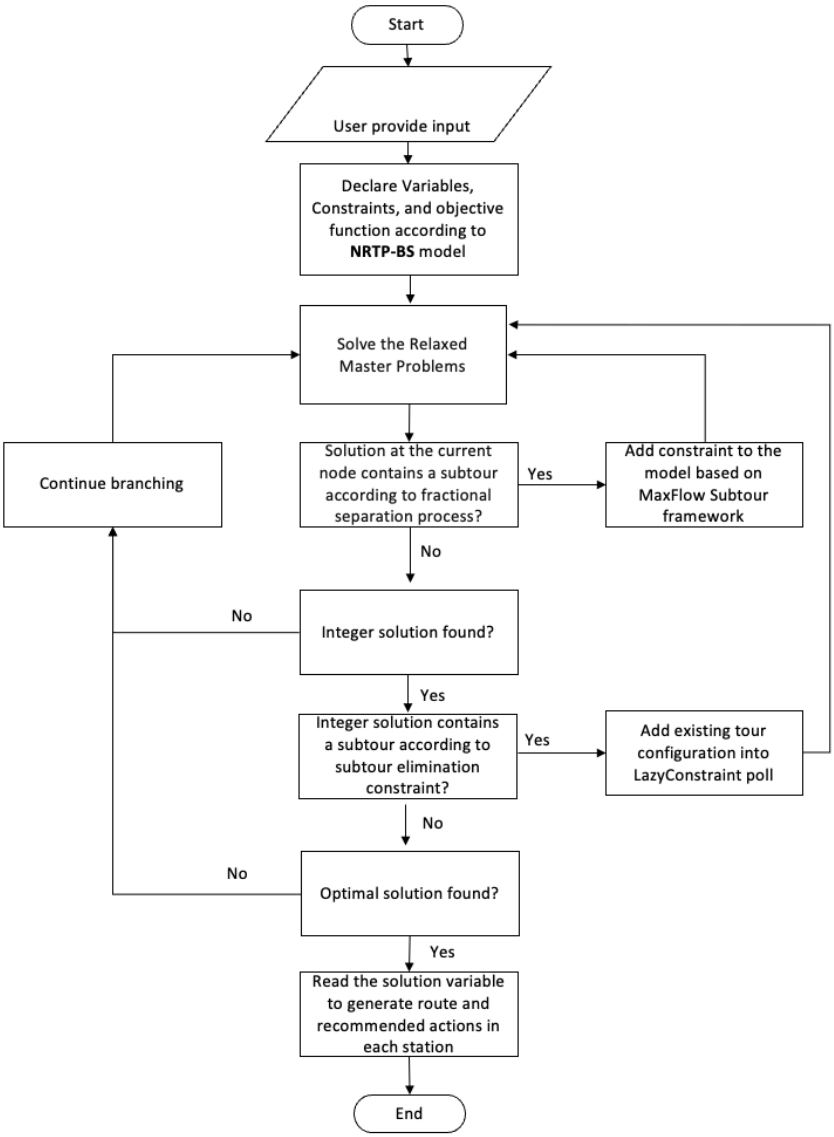

Figure 5.4: CutCallback+LazyConstraintCallback flow in NRTP-BS

6 Computational Experiments

To identify the critical factors in our model, we will perform sensitivity analysis on the docplex implementations of the Simple Night Tour and the NRTP-BS presented in Chapter 5. To this end, we have tested our models on synthetic instances and we analyze the computational results in terms of achieved optimal solution and required running times. In the chapter, we describe how we created the test benchmark and the results of our analysis.

6.1 Instance benchmark

In order to generate the instances, we will build a generic function that can create a randomized dataset given a number of stations. The key components of the dataset include a distance table (d), representing the distances between stations, and simulated data specifying the number of fully charged scooters, the demand for scooters, and the required rebalancing quantity at each station. The distance table is populated symmetrically with random distances between stations, excluding self-distances (i.e., from a station to itself). A large value (10^8) is used to represent self-distances, thus eliminating the possibility of self-visits. Subsequently, the function generates prerequisite data for each station. It computes the initial number of fully charged scooters F at each station, sets a uniform demand of 8 scooters for all stations, and calculates the required rebalancing quantity Q to meet the demand. The final step involves creating an object of a data class using the generated distance table d, simulated demand data sim, F and Q. The explained steps are summarized in the following algorithm:

```
def dummy_data_gen_new(num_stations,print_log=0):
    ```
    Function to generate dummy data
    Input: Number of station
    Output: A class of data consist of distance table, number of full,low,
```
```

```
needs and penalty in each station.
```
```
d= pd.DataFrame(np.zeros([num_stations+1,num_stations+1]))
for i in range(0,num_stations+1):
    for j in range(i,num_stations+1):
        if i != j:
            temp=random.uniform(0,1)*5
            d[i][j]=temp
            d[j][i]=temp
        else:
            d[i][j]=100000000 #this is a trick to eliminate self visit (0,0)

#Generating simulation random data
Q=[0] #ideal number of full battery escooter that should be picked up
      #to achieve the desired number after rebalancing
F=[0] #number of full battery scooters at node j BEFORE rebalancing
demand = [0] #need for every station

for i in range(1,num_stations+1):
    F.append(int(random.uniform(0,1)*5))
    demand.append(8) #for the moment the needs for all station are the same
    Q.append(demand[i]-F[i])


sim = pd.DataFrame(list(zip(F,demand,Q)),
                   columns=['F','demand','Q'])

data = data_class(d,sim,F,Q)
return data
```

Examining the generated data, $d[i][i]$ for all station i represents the self distance. In reality, this value is 0. However, considering the objective functions in the Simple Night Tour and NRTP-BS (equations 4.1 and 4.11), setting $d[i][i] = 0$ may prompt the algorithm to prefer self-visit rather than visiting other node. Therefore, rather than assigning a distance of 0 to the same station, opting for a substantially larger value compared to other distances discourages the algorithm from selecting $x[i][i] = 1$. Moreover, concerning the distance between other stations, we will generate a random number between 0 and 5, following a normal distribution. The normal distribution is selected to replicate the distance of docking station around the city. Some of the stations will be close to each other, due to more popular demand, and some of the stations are quite far apart with each other. Regarding the e-scooter situation at each station, the variables F and L will denote the number of full and low-battery e-scooters present in a station, respectively. The values for these variables will vary randomly within the ranges from 1 to 5 and 1 to 8, respectively.

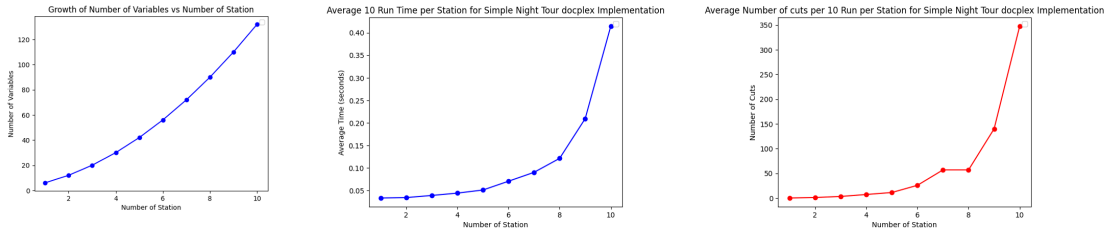

Figure 6.1: [Left] Growth of Number of Variables vs Number of Station. [Middle] Average 10 Run Time per Station for Simple Night Tour docplex implementation. [Right] Average Number of cuts per 10 run per station for Simple Night Tour docplex implementation.

6.1.1 Results on the Simple Night Tour Experiment

By analyzing the mathematical model for the Simple Night Tour, it can be inferred that the growth of binary variables accelerates significantly as the number of stations increases ($O(n^2)$). This main growth is driven by the binary variables x_{ij} whereas the variable θ grows linearly ($O(n)$). To investigate the computational implications of this phenomenon, a sensitivity analysis will be performed by varying the number of stations in relation to the docplex implementation with integer separation method incorporated through the LazyConstraintCallback, according to the schema presented in Figure 5.2. Throughout this analysis, we will monitor the growth of variables, runtime, and the number of added subtour elimination constraints as we increment the number of stations. To account for the inherent randomness in the dataset, the experiment will be repeated 10 times for each number of stations, ensuring a comprehensive understanding of the model's natural behavior. The sensitivity analysis ranges from the number of stations being 1 to 10. The experiment was carried out on a MacBook Pro equipped with an Apple M1 Pro chip and 32 GB of RAM.

As illustrated in Figure 6.1 (left), our analysis is aligned with the evidence indicating that an increase in the number of stations corresponds to quadratic increment in the number of variables. This observation is further reflected in Figure 6.1 (middle), where the average runtime for solving the Simple Night Tour model exhibits an exponential growth. While this poses a scalability concern for the e-scooter company, a closer examination reveals that the runtime for an individual problem remains relatively fast, clocking in at less than 1 second per iteration, at least in the instances that we tested up to 10 stations. Furthermore, we are interested to figure out what is the contributing operation that halt the run time as the number of station grows. Evidently, with the expanding number of stations, we need to do more separation operation while docplex conducting its branch-and-cut operation. Taking a closer look, when the number of station equal to 10, docplex runs the integer separation method almost 350 times.

This sensitivity analysis serves as a cautionary note, indicating that we must carefully manage runtime expectations when dealing with a substantial number of stations. In this experiment, the number of e-scooters in question does not exert an influence on the runtime. Additionally, as previously highlighted in the model formulation, the Simple Night Tour model is designed to generate only a perfectly balanced solution, lacking variables to address system imbalances. Consequently, we need to augment the code to handle instances where the model produces infeasible solutions. This occurrence could arise from insufficient e-scooters to rebalance the distribution or a truck capacity insufficient to accommodate the required load.

6.1.2 Results on NRTP-BS Experiment

To enable the model to give us the next-best-solution, we will pivot our experiment into the NRTP-BS. As detailed in Chapter 4, in addition to allow for demand imbalances, this model introduces the flexibility for operators to exchange low-battery scooters with extra batteries transported from the depot, rather than physically swapping the low-battery e-scooters for fully charged ones. However, this flexibility comes at a cost, as the number of variables to manage is higher compared to the Simple Night Tour model, given the same number of stations.

In this experiment, we explore two different methods to programmatically solve the model: one utilizing only integer separation method implemented using LazyConstraintCallback, and the other combining the integer separation with fractional separation implemented using LazyConstraintCallback and CutCallback, respectively. As depicted in Figure 6.2, the overall performance of the method that employs both CutCallback and LazyConstraintCallback outperforms the method that only employs LazyConstraintCallback as the number of stations increases. This can be attributed to the nature of the problem: as the number of stations grows, the solution space becomes larger. CutCallback facilitates faster convergence by adding more guardrails to the model at earlier stages, even when integer solutions are not yet available. Without CutCallback, the model needs to wait for integer solutions coming from the linear relaxation, which, in the worst case, may require solving to integrality at the current Master Problem, before validating the solution for subtour elimination constraints. Implementing CutCallback allows us to identify infeasible solutions early, and add some relevant constraints earlier, resulting in an overall faster runtime.

Furthermore, Figure 6.3 reveals that when utilizing only LazyConstraintCallback in the model, fewer separation problem is conducted. This is because we need to wait for the model to find the integer solution first. For instance, when the number of stations is 100, solving the NRTP-BS with only LazyConstraintCallback performed around 580 integer separation problem on average. However, when combining LazyConstraintCallback and CutCallback, the number of integer and fractional separation problem combined increases to more than 1000. Therefore, combining this observation with the runtime analysis, we argue that the abundance of fractional separation problem help the algorithm to reduce the solution space by cutting infeasible early from the search tree. Since there is less solution to explore, the algorithm can find the optimal solution quicker.

Diving deeper, in the combined method, since fractional separation procedure has been conducted, algorithm does not have to perform many integer separation anymore. However, as evident in the cases where the number of stations is 6 and 19, sometimes the integer-generated solution still contains a subtour. This examples confirm that while fractional separation problem can aid the algorithm in converging to the solution faster, it does not guarantee that the generated route satisfies all necessary constraints without deploying the integer separation problem.

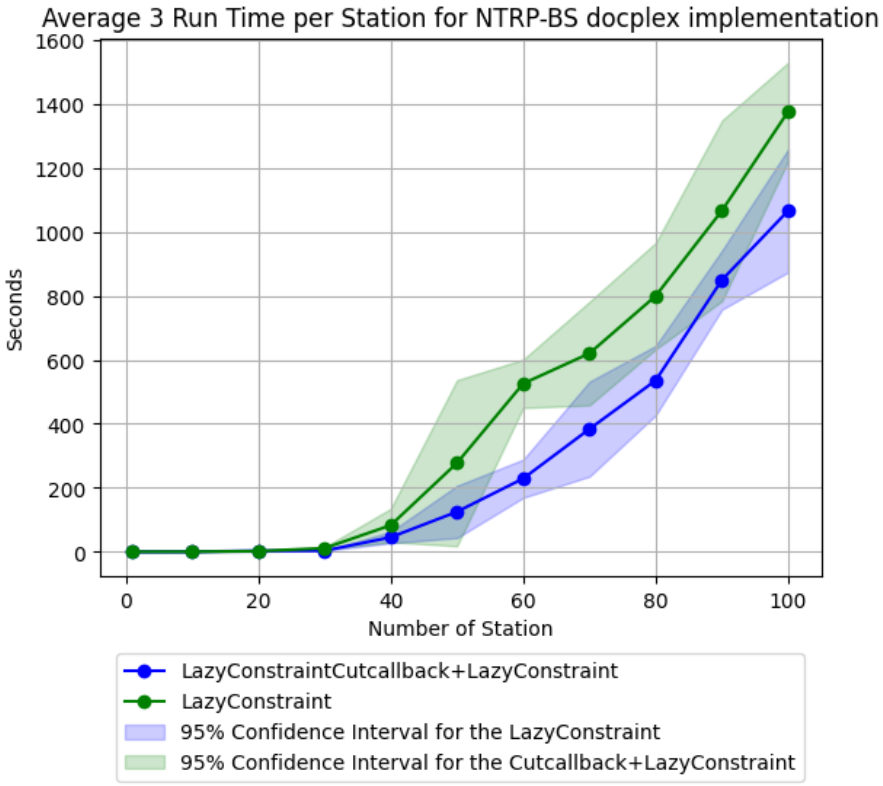

Figure 6.2: Average 3 run time per station for NRTP-BS

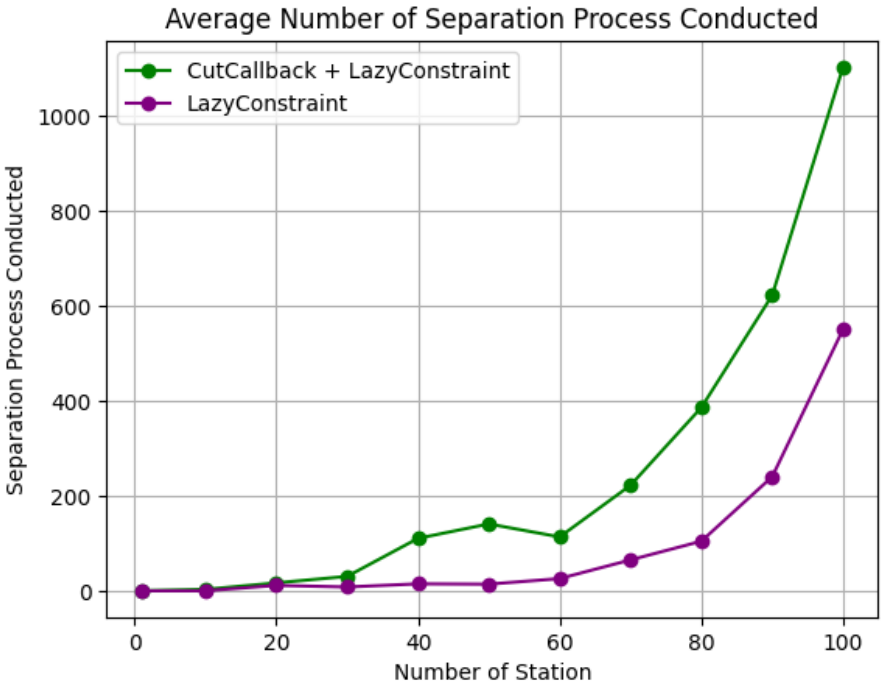

Figure 6.3: Average number of Separation Process for NRTP-BS

7 Conclusions

In this thesis, we have reviewed the existing literature on the implementation of e-scooter sharing systems in society. Through this review, a prominent challenge in the field has been identified: the issue of over-saturation and under-utilization of e-scooters. One proposed solution involves implementing a night tour operation to rebalance e-scooter distribution. However, a notable gap in the literature lies in the limited discussion of night tour operations that allow swapping e-scooter batteries rather than the entire units. This thesis aims to address this gap by introducing a novel model that facilitates battery swaps during night tour operations. The anticipated outcome of this model is to provide actionable insights for night tour operators, guiding them on where to drive and what actions to take at each station during the operation.

To model this complex problem, we present two distinct mathematical programming models: the Simple Night Tour model and the NRTP-BS model. The first model draws inspiration from the work of Dell'Amico [11] on bicycle rebalancing problems, functioning as a benchmark model. The second model represents an original enhancement, introducing the capability for night tour operators to swap batteries during operations. Both models contain subtour elimination constraints, the quantity of which becomes significant with an increasing number of stations. This characteristic makes the problem not directly tractable. Although the row generation method offers a solution to dynamically add this constraint, the computational cost rises significantly when recalculating the solution if a subtour is detected at the conclusion. To enhance the efficiency of the model algorithm, we propose a separation method that can identify early whether the solution at the current node within the branch-and-cut process contains a subtour or not, instead of waiting until the end of the tree to check if the solution contains a subtour.

During the separation procedure, we have two options at a node level: conducting it once the integral solution has been found or conducting it once the fractional solution has been found. In Docplex, the former method can be implemented by utilizing the LazyConstraintCallback whereas the latter can be implemented by utilizing CutCallback. Both methods provide flexibility by dealing with subtour constraints dynamically without generating them in advance. However, using CutCallback alone does not fully guarantee the absence of subtours in the final

solution since the solution might be fractional. Therefore, CutCallback needs to be paired with LazyConstraint-Callback to check if the integer solution found contains a subtour or otherwise.

To further analyze the proposed model, we have done a sensitivity analysis on both of the models against a systetic dataset ranging from a 1 number of stations to 100. The experiment is repeated three times at each station to ensure that we capture the general behavior of the model and not just a lucky coincidence because of the generated dataset. With these numerical experiments, we have observed that both models can finish the run in under 2 seconds for 25 stations. However, as the number of stations grows, the NRTP-BS needs up to 23 minutes to find the optimal solution. When both callbacks are incorporated into the model, CutCallback effectively addresses the separation problem, resulting in the faster addition of the necessary LazyConstraint. Deploying both types of cuts has proven to expedite algorithm runtime. This is especially true as the number of stations grows. At the largest number of stations in this experiment, 100, the runtime gap is as wide as 5 minutes during the sensitivity analysis.

In order to enhance this research, a thorough comprehension of the inner mechanisms of Docplex is required. Beside user-defined cuts, the solver dynamically introduces various cuts to the model during the solving process. A deeper insight into these processes enables us to strategically introduce additional cuts to tighten the formulation to the integer convex hull, thereby accelerating the algorithm's convergence. A critical takeaway from this experience underscores the significance of diligently reviewing the documentation provided by the solver's developer. Different versions may demand distinct implementation approaches, highlighting the need for careful consideration and adaptation to specific solver specifications. Another enhancement that can be made is to apply the NRTP-BS model against real e-scooter data to further validate the model.

References

[1] Mohammed Almannaa, Huthaifa Ashqar, Mohammed Elhenawy, Mahmoud Masoud, and A. Rakotonirainy. A Comparative Analysis of E-Scooter and E-Bike Usage Patterns: Findings from the City of Austin, TX. 06 2020.

[2] Steve Annear. A new bike-share company, OFO, is rolling into cities near Boston. https://www.bostonglobe.com/metro/2017/09/27/new-bike-share-company-ofo-rolling-into-cities-near-boston/nRnVY7Tdz0PYc1gSFWtNhL, September 27 2017. Boston Globe. Retrieved December 3rd, 2023.

[3] David L. Applegate, Robert E. Bixby, Vasek Chvátal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2007.

[4] Gulay Barbarosoglu and Demet Ozgur. A Tabu Search Algorithm for the Vehicle Routing Problem. *Computers and Operations Research*, 26(3):255–270, 1999.

[5] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1997.

[6] Kris Braekers, Katrien Ramaekers, and Inneke Nieuwenhuyse. The Vehicle Routing Problem: State of the Art Classification and Review. *Computers and Industrial Engineering*, 99, 12 2015.

[7] Teobaldo Bulhões, Anand Subramanian, Güneş Erdoğan, and Gilbert Laporte. The static bike relocation problem with multiple vehicles and visits. *European Journal of Operational Research*, 264(2):508–523, 2018.

[8] Nick Carey and Paul Lienert. E-scooters Fall Head Over Wheels with Battery Swapping. https://www.reuters.com/business/autos-transportation/e-scooters-fall-head-over-wheels-battery-swapping-2022-03-24/, March 24 2022. Retrieved December 3rd, 2023.

[9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, second edition, 2001.

[10] Vladimir Deineko and Alexander Tiskin. One-sided monge TSP is NP-Hard. pages 793–801, 11 2006.

[11] Mauro Dell'Amico, Eleni Hadjicostantinou, Manuel Iori, and Stefano Novellani. The bike sharing rebalancing problem: Mathematical formulations and benchmark instances. *European Journal of Operational Research*, 237(1):252–263, 2014.

[12] Paul DeMaio. Bike-sharing: History, impacts, models of provision, and future. *Journal of Public Transportation*, 12(4):3, 2009.

[13] Ahmadreza Faghih-Imani, Naveen Eluru, Ahmed M. El-Geneidy, Michael Rabbat, and Usama Haq. How land-use and urban form impact bicycle flows: evidence from the bicycle-sharing system (BIXI) in Montreal. *Journal of Transport Geography*, 41:306–314, 2014.

[14] Christian Fries. *Mathematical Finance: Theory, Modeling, Implementation*. Wiley, 2007.

[15] David Garrick. Dockless bike sharing arrives in San Diego on Thursday. https://www.sandiegouniontribune.com/business/growth-development/sd-fi-bikeshare-20180215-story.html, February 15 2018. San Diego Union-Tribune. Retrieved December 3rd, 2023.

[16] Mauro Giacon. Padova. Monopattini, stop al parcheggio selvaggio: sanzione a chi li lascia dove non si può. https://www.ilgazzettino.it/nordest/padova/monopattini_sharing_parcheggio_selvaggio_multa_app-7321764.html, 2023. Retrieved November 19th, 2023.

[17] Nicholas I. M. Gould. *An Introduction to Algorithms for Continuous Optimization*. 2000.

[18] Adriana Heldiz. San Diego Explained: Why Dockless Bikes Are Everywhere, year = 2018. https://www.voiceofsandiego.org/topics/news/san-diego-explained-why-dockless-bikes-are-everywhere/, March 8. Voice of San Diego. Retrieved December 3rd, 2023.

[19] Jason Holdsworth. The Nature of Breadth-First Search. *Technical Report 99-1, James Cook University, Australia*, 02 1999.

[20] IBM. CPLEXLazyConstraintCallback Documentation. https://www.ibm.com/docs/en/icos/12.10.0?topic=classes-cplexcallbackslazyconstraintcallback. Retrieved December 3rd, 2023.

[21] IBM. Cut Callback. https://www.ibm.com/docs/en/icos/20.1.0?topic=legacy-cut-callback. Retrieved December 3rd, 2023.

[22] IBM. docplex.mp.model Module. https://ibmdecisionoptimization.github.io/docplex-doc/mp/docplex.mp.model.html. Retrieved December 3rd, 2023.

[23] IBM. Beyond Linear Programming - IBM Decision Optimization Tutorials. https://ibmdecisionoptimization.github.io/tutorials/html/Beyond_Linear_Programming.html, Retrieved December 3rd, 2023.

[24] IBM. IBM Decision Optimization CPLEX Modeling for Python Documentation. https://ibmdecisionoptimization.github.io/docplex-doc/getting_started_python.html, Retrieved December 3rd, 2023.

[25] K. Kim. Investigation on the effects of weather and calendar events on bike-sharing according to the trip patterns of bike rentals of stations. *Journal of Transport Geography*, 66:309–320, 2018.

[26] KnotCity. KnotCity - Docking is the new black. https://www.knotcity.com/en/, 2023. Retrieved November 19th, 2023.

[27] Y. Li and Y. Zheng. Citywide Bike Usage Prediction in a Bike-Sharing System. *IEEE Transactions on Knowledge and Data Engineering*, 32(6):1079–1091, 2019.

[28] Li, Yexin and Zheng, Yu and Yang, Qiang. Dynamic bike reposition: A spatio-temporal reinforcement learning approach. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 1724–1733, New York, NY, USA, 2018. Association for Computing Machinery.

[29] Tom Magnanti and Jim Orlin. Applied Mathematical Programming. `https://web.mit.edu/15.053/www/AppliedMathematicalProgramming.pdf`. Retrieved December 3rd, 2023.

[30] Microsoft. Define and solve a problem by using Solver. `https://support.microsoft.com/en-us/office/define-and-solve-a-problem-by-using-solver-5d1a388f-079d-43ac-a7eb-f63e45925040`. Retrieved December 3rd, 2023.

[31] David R. Morrison, Sheldon H. Jacobson, Jason J. Sauppe, and Edward C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016.

[32] United Nations. Resolution adopted by the General Assembly on 25 September 2015. United Nations General Assembly, 2015. A/RES/70/1.

[33] Jesus Osorio, Chao Lei, and Yanfeng Ouyang. Optimal rebalancing and on-board charging of shared electric scooters. *Transportation Research Part B: Methodological*, 147:197–219, 2021.

[34] M. W. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33:60–100, 1991.

[35] A. Burak Paç. Row generation techniques for approximate solution of linear programming problems. 2010.

[36] Julius Pfrommer, Joseph Warrington, Georg Schildbach, and Manfred Morari. Dynamic Vehicle Redistribution and Online Price Incentives in Shared Mobility Systems. *IEEE Transactions on Intelligent Transportation Systems*, 15(4):1567–1578, August 2014.

[37] Thomas Pogiatzis. *Application of mixed-integer programming in chemical engineering*. PhD thesis, 2013.

[38] R. Radharamanan and L.I. Choi. A branch and bound algorithm for the travelling salesman and the transportation routing problems. *Computers and Industrial Engineering*, 11(1):236–240, 1986.

[39] Leif Reigstad. The Rise and Fall of Dockless Bike-Sharing in Dallas. `https://www.texasmonthly.com/news-politics/rise-fall-dockless-bike-sharing-dallas/`, 2018. Retrieved December 3rd, 2023.

[40] Ridemovi. Wishing You an Electric 2022. `https://www.ridemovi.com/whishing-you-an-electric-2022/`. Retrieved December 3rd, 2023.

[41] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley and Sons, 1998.

[42] Susan Shaheen et al. Bikesharing in Europe, the Americas, and Asia: Past, Present, and Future. *Transportation Research Record*, 2143(1):159–167, 2010.

[43] Chih-Kang Lin Shangyao Yan and Zong-Qi Kuo. Optimally locating electric scooter battery swapping stations and battery deployment. *Engineering Optimization*, 53(5):754–769, 2021.

[44] Andrew Small. What Doomed Seattle's Pronto Bike-Share Program. `https://www.bloomberg.com/news/articles/2017-01-31/what-doomed-seattle-s-pronto-bike-share-program`, January 31 2017. Retrieved December 3rd, 2023.

[45] Robert Tarjan. Depth-first search and linear graph algorithms. In *12th Annual Symposium on Switching and Automata Theory (swat 1971)*, number 12, pages 114–121, 1971.

[46] Qiang Wei. China's bike-sharing industry braces for explosive growth. `http://en.people.cn/n3/2017/0305/c90000-9186008.html`, 2017. Retrieved December 3rd, 2023.

[47] Z. Yang, J. Hu, Y. Shu, P. Cheng, J. Chen, and T. Moscibroda. Mobility Modeling and Prediction in Bike-Sharing Systems. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, pages 165–178, 2016.

[48] Resty Woro Yuniar. Are China's bike-sharing services oversharing? `https://www.scmp.com/week-asia/business/article/2113028/are-chinas-bike-sharing-services-oversharing`. Retrieved December 3rd, 2023.

[49] Marco Zanetti. Optimization Models for Bike Sharing Systems. Master's thesis, Padova University, 2015.

[50] Jiawei Zhang. Gradient Descent based Optimization Algorithms for Deep Learning Models Training. *IFM Lab Tutorial Series*, 1, 2019.

[51] Yaoming Zhou, Zeyu Lin, Rui Guan, and Jiuh-Biing Sheu. Dynamic battery swapping and rebalancing strategies for e-bike sharing systems. *Transportation Research Part C: Emerging Technologies*, 111:328–347, 2020.

[52] Yaoming Zhou, Zeyu Lin, Rui Guan, and Jiuh-Biing Sheu. Dynamic battery swapping and rebalancing strategies for e-bike sharing systems. *Transportation Research Part B: Methodological*, 177:102820, 2023.

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Prof. Luigi De Giovanni, for his continuous support, guidance, and patience throughout my thesis journey.

Heartfelt appreciation goes to my family, the KDaheng Team, and the Spenard Family, whose unwavering support and boundless love have made me feel that I am never alone in this journey. It's truly a privilege to have a well-rounded, unconditional support.

To all my friends around the world—Padova, Luxembourg, Indonesia, everywhere—thank you for always believing in me, so it was not hard to convince myself that I could finish school while working at the same time. I acknowledge this is borderline impossible. I would only recommend this to everyone with a brave soul and a strong desire to always keep learning.

To Italy, thank you for giving me a chance and welcoming me with an open arm. Thank you to the Ministry of Foreign Affairs (MAECI) for the scholarship which enable me to go to this prestigious school and contribute my thoughts on this great topic.

At this moment, I extend my gratitude to everyone who has contributed to my growth. Each of you holds a unique place in my heart, and I am humbled by the connections we share. Thank you.

For God, the Nations, and our beloved almamater.