View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Clemson University: TigerPrints

Clemson University
TigerPrints

All Dissertations | Dissertations

12-2016

A Study of the Static Bicycle Reposition Problem with a Single Vehicle

Ling Zu
Clemson University, lzu@g.clemson.edu

Follow this and additional works at: https://tigerprints.clemson.edu/all_dissertations

Recommended Citation
Zu, Ling, "A Study of the Static Bicycle Reposition Problem with a Single Vehicle" (2016). *All Dissertations*. 1808.
https://tigerprints.clemson.edu/all_dissertations/1808

This Dissertation is brought to you for free and open access by the Dissertations at TigerPrints. It has been accepted for inclusion in All Dissertations by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.

A STUDY OF THE STATIC BICYCLE REPOSITION PROBLEM WITH A SINGLE VEHICLE

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Industrial Engineering

---

by
Ling Zu
December 2016

Accepted by:
Mary E. Kurz, Committee Chair
B. Rae Cho
William G. Ferrell
David M. Neyens

ABSTRACT

The Bicycle Sharing System (BSS), a public service system operated by the government or a private company, provides the convenient use of a bicycle as a temporary method of transportation. More specifically, this system allows people to rent a bike from one location, use it for a short time period and then return it to either to the same or a different location for an inexpensive fee. With the development of IT technology in the 1990s, it became possible to balance the bicycle inventory among the various destinations. In fact, a critical aspect to maintaining a satisfactory BSS is effectively rebalancing bicycle inventory across the various stations. In this research, we focus on the static bicycle repositioning problem with a single vehicle which is abstracted from the operation issue in the bicycle sharing system. The mathematical model for the static bicycle reposition problem had been created and several variations had been analyzed. This research starts to solve the problem from a very restrictive and constrained model and relaxes the constraints step by step to approach the real world case scenario. Several realistic assumptions have been considered in our research, such as a limited working time horizon, multiple visit limitation for the same station, multiple trips used for the vehicle, etc. In this research, we use the variable neighborhood search heuristic algorithm as the basic structure to find the solution for the static bicycle reposition problem. The numeric results indicate that our algorithms can provide good quality result within short solving time. By solving such a problem well, in comparison to benchmark algorithms, this research provides a starting place for dynamic bicycle repositioning and multiple vehicle repositioning.

DEDICATION

To my parents for all their love and support and for giving me the best education possible. I appreciate their sacrifice. Without them, I would not have been able to accomplish all that I have.

To my husband Wennian for his never-ending support and help. His love, intelligence, and encouragement have given me the motivation to continue my studies.

To my lovely son ZuXiao (little Sammy), he is my sunshine. Every time when I felt desperate and tired, he saved me and brightened my life with sunshine.

ACKNOWLEDGMENTS

I would like to express my deep appreciation and thanks to my advisor, Dr. Mary Kurz. You have been a great mentor to me. I would like to thank you for all your encouragement and help both for my studies and for my life.

TABLE OF CONTENTS

Page

Table of Contents (Continued)

Table of Contents (Continued) Page

LIST OF TABLES

LIST OF FIGURES

List of Figures (Continued)

Figure Page

CHAPTER ONE

INTRODUCTION

The Bicycle Sharing System (BSS), a public service system operated by the government or a private company, provides the convenient use of a bicycle as a temporary method of transportation. More specifically, this system allows people to rent a bike from one location, use it for a short time period and then return it to either to the same or a different location for an inexpensive fee. It has been in use for several decades, the earliest on record in Amsterdam in 1965 (Shaheen and Guzman, 2011) where approximately fifty white bicycles were placed around the inner city for use for free. Because many of these bicycles were stolen or became damaged, this bicycle sharing system, called the White Bikes, was terminated shortly after it was initiated. This free BSS, referred to as the first generation bicycle sharing system, was replaced with a second generation which implemented changes to prevent theft and damage. The first organized large-scale BBS, the Bycykler København, which involved one thousand bicycles and began in Copenhagen, Denmark, in 1995 (Shaheen et al., 2010), represents a typical 2nd generation BSS. With a refundable deposit, a specially designed bike with non-standard parts, and fixed stations and lockers, the Bycykler København reduced the theft of and damage to the bicycles and is still in operation today.

With the development of IT technology in the 1990s, the 3rd generation BSS integrated the smart card and other technology into the system, offering such new options as the collection of real-time information about the operator and the station. In addition, using this technology, it became possible to balance the bicycle inventory among the

various destinations. The latest generation, the fourth, of the BSS system, integrated advanced information system and network technology as well as GPS tracking and real-time mobile communication technology. As a result, the centralized control center has real-time data on the status of the station as well as the capability to track the location of each bicycle and to send this information to an interested customer through an internet connection or a mobile device. All of this new technology integrates the bicycle sharing system more fully, enhancing its usability in today's society.

In addition to this low-cost, short-distance transportation service, the BSS brings other benefits to the public. Daily commuters can save the time and the stress of traveling through congested traffic and avoid the pressure and cost of finding parking. In addition, those using this service for short-distance travel can enjoy the benefits of physical exercise. Further, tourists can enjoy the city without having to deal with multiple bus transfers, taxi fares and sore feet. Finally, the public is subjected to fewer traffic jams, less pollution and improved air quality.

Since the BSS not only provides individual users with a convenient, affordable mode of transportation but also can benefit the city and the public, this system is becoming increasingly more popular in modern cities as evidenced by the number of such systems that have been implemented around the world. According to Larsen (2013), in April 2013, more than 500 cities in 49 countries have BSS's. Even though the BSS is based on self-service, it requires significant routine maintenance for the system to run smoothly, including regular equipment and bicycle checks and repairs. Of these various maintenance jobs, perhaps the most important is to balance the available bicycles among

the different rental stations, especially critical as it impacts customer satisfaction. An empty station prevents a customer from renting a bike, while at the same time; a full station blocks a customer from returning one. According to the research conducted by Shaheen and Guzman (2011), most BSS complaints are triggered by the unavailability of bicycles and/or the unavailability of vacant lockers at a destination.

The number of bikes at each station should be maintained at a certain level. Usually, the process of rebalancing the number of bicycles is done using a fleet of vehicles to move bicycles among stations. In general, this bike repositioning problem can be classified as either static or dynamic. The dynamic balancing problem refers to the balancing process that occurs when the system is in operation and the number of bicycles at any given station may change significantly, affecting the need for and the result of a repositioning process. This type of problem, referred to as the *dynamic bicycle repositioning problem* (DBRP). The static balancing problem refers to the night repositioning operation. Since during the night, the number of bikes at each station either remains the same or experiences only small changes, it does not affect the result of the repositioning event. This type of repositioning problem, referred to as the *static bicycle repositioning problem* (SBRP), is the focus of this research.

CHAPTER TWO

LITERATURE REVIEW

The static bicycle rebalancing problem, the topic of this research, is an aspect of the vehicle routing problem with pickup and delivery (VRPPD). An extension of the classic vehicle routing problem (VRP), VRPPD has been investigated from many perspectives; review papers, such as those by Berbeglia et al. (2007) and Parragh, Doerner, and Hartl (2008), provide summaries of this research. The VRPPD involves three types, the first one being the One-to-Many-to-One (1–M–1) problem. In this type, the commodities are delivered from one depot to many customers and then are collected from the customers and delivered back to the depot, a problem similar to the classic VRP. Real-world scenarios exemplifying this problem include the soft drink delivery problem, new and used appliances delivery / collection problem, and the full and empty pallets delivery problem. The second type of the VRPPD is the One-to-One (1–1) problem in which each commodity has a specified origination and destination. The situations researched concerning this type include the courier service problem, the less than a truckload transportation problem, the maritime shipping problem, and the dial-a-ride problem. In the third type, the Many-to-Many (M-M) problem, each commodity may have multiple originations and destinations, each location in the system can be the origination or the destination, or both situations can be present simultaneously. This type includes several variants such as the SWAP problem, the K-delivery problem, and the 1-commondity pickup and delivery problem. The static bicycle rebalancing problem (SBRP) investigated is the latter, a 1-commondity pickup and delivery problem.

Previous research primarily used two general approaches (models) to address the SBRP problem, the first solving it with the classic traveling salesman problem with pickup and delivery, an approach that includes a visiting limitation in the model. The key feature of this approach is that the entire pickup / delivery event for each station is limited to at most one visit. The second approach, an extended model of the first with more realistic assumptions, relaxes the visiting limitation by allowing the same station to be visited multiple times throughout the route. When one or more stations are large and the number of bikes requiring delivery or pickup exceeds the vehicle capacity, a station's inventory cannot be repositioned to the target station in only one visit. If repositioning each station to its target inventory level is a hard constraint, it may not even be possible to provide a feasible solution using the first approach. While the second approach is more realistic than the first, the realistic assumption makes it more complex from both the modeling and resolving perspective. On the other hand, even though the first approach includes unrealistic assumptions, it has the advantage of being well researched and many inequalities and methodologies can be applied directly.

Hernández-Pérez & Salazar-González (2004a) first proposed the one-commodity pickup-and-delivery traveling salesman problem (1-PDTSP), extending the classic TSP problem by considering both pickup and delivery customers. Their objective was to determine the most cost-effect solution by visiting the depot and each customer once and once only while at the same time collecting all commodities from the pickup customer and satisfying all requests from the delivery customers. They proposed a branch-and-cut algorithm to solve this problem. In a subsequent study, they (2004b) refined their

research, developing two heuristic algorithms to address this problem with up to 500 customers in the system. Extending this research further, Hernández-Pérez & Salazar-González (2010) found a close relationship between the 1-PDTSP and the Capacitated Vehicle Routing Problem and applied the inequalities recently developed for the 1-PDTSP problem with the branch-and-cut framework, successfully solving this 1-PDTSP problem with more than 100 customers optimally.

Because of the lack of realism in the first approach to the SBRP problem, Benchimol et al. (2011) proposed a second approach in their research, providing an integer programming model that defined a static rebalancing problem referred to as the single vehicle one-commodity capacitated pickup and delivery problem. This problem considers the network as built on one complete graph, with the depot being a special vertex representing the garage or parking lots of the operation vehicles. All routes start and end at this location, with every other vertex in the network being a bike rental station where consumers can rent or return a bike. Only one capacitated vehicle is used to redistribute the bikes among the various stations, each having a target number. The objective is to find the most cost-effect route for achieving the target number at all stations. However, unlike for the classic TSP problem, the vehicle route can visit the same station multiple times.

Chemla, Meunier, and Calvo (2013) investigated the problem proposed in Benchimol et al. (2011)'s research, providing an exact mathematical model including the relaxations for the algorithm. An upper bound of the optimal solution for the problem is obtained through a Tabu search that only considers the visiting order in the solution and

obtains the loading instructions using an auxiliary algorithm which reduces the search space significantly. The research reported here provides an auxiliary algorithm based on the max flow problem to find the optimal bicycle loading / unloading quantity for each station of a given routing sequence. It appears to be the first research to implement a heuristic method to solve the SBRP problem.

More recently, Rainer-Harbach, Papazek, Hu, and Raidl (2013) extended the model proposed by Chemla et al. (2013). However, in contrast to the solution proposed by Benchimol et al. (2011) and Chemla et al. (2013), multiple vehicles with different capacities are used to balance bikes among the various stations, with the vehicles in the fleet beginning and ending at separate locations with no storage space for bikes. Each vehicle has a fixed capacity and a total time limitation for the operation, e.g. work shift length. An additional improvement included in this research was the relaxation of the system balancing constraint. Unlike in previous work, the system balancing was not a hard constraint in this paper; rather any deviation from the target number was considered as an input for a penalty function, its objective being to minimize the combination of these 3 aspects: (1) the total deviation from the target number at each location, (2) the total number of handled bikes (total loaded/unloaded), and (3) the total operation time which is linear related to vehicle operational cost. This relaxation of the station target status constraint expands the solution space for the SBRP problem, bringing the solution closer to the real-world situation. Further, it can also help the first approach provide a feasible solution when the SBRP problem includes such special cases as the station's pickup / delivery quantity is larger than the vehicle capacity. Furthermore, this research

provided a general structure for solving the SBRP problem, proposing a two-step strategy to decompose its complexity of the problem. It first creates the vehicle routing schedule, then uses the integer programming model to solve the loading / unloading plan for each station visited based on the vehicle routing schedule generated in the first step. This method addresses the complexity of the overall problem by solving two smaller ones in sequence.

In further research, Raidl, Hu, Rainer-Harbach, Papazek (2013) improved the second step of their initial strategy, which was based on the integer programming model, a time-consuming process. In this more recent research, they provided a new, more efficient method for calculating the optimal loading operations based on two maximum flow computations. The result of their computations supported their new algorithm, reducing the time needed significantly.

Raviv, Tzur, and Forma (2013) used a general model approach, proposing a two mixed integer programming (MIP) formulation. Both MIP formulations use the total operation cost as the objective. The first MIP formulation, the *arc-index* formulation, was constrained by the number to times a station could be visited per trip as in the first approach, while the second MIP formulation, the *time-index* formulation, was constructed without any visiting limitations as in the second approach. Several inequalities and dominance rules were applied in these 2 models, ones that solved both MIP models with CPLEX. The computational results found that typically the *arc-index* formulation can yield a solution with better solution (i.e. smaller objective) than *time-index* formulation in 2 hours running time even though the *arc-index* formulation has smaller feasible set of

solution; however, the *time-index* formulation was found to have a better solution than the *arc-index* formulation when given a longer running time.

In more recent research, Li et al. (2016) developed a model considering multiple types of bicycles in the system. In their research, each station had specific lockers for the different types of bicycles, whereas other studies did not include this constraint, allowing any type of bicycle to occupy any empty locker. In addition, they introduced two types of strategies, substitution and occupancy. The substitution strategy allowed users to rent a substitute type of bicycle when the type they requested was out of stock, while the occupancy strategy allowed the users to return the bicycle to a substitutable locker type. Their model, based on the first SBRP model with a station visit limitation constraint, includes a traveling and penalty costs for each station. They also used the 2-step method to solve the problem, first generating the vehicle route through a hybrid generic search, then using a greed heuristic algorithm to determine the loading operation at each station.

Ho and Szeto (2014) implemented the station target status constraint relaxation in the Traveling Salesman Pickup and Delivery model (first structure model) and proposed an Integer Programming model and a heuristic algorithm for solving the problem for a single vehicle scenario, using the classical Travel Salesman problem with delivery to solve it. Applying the findings from Chemla et al. (2013), it explicitly defined the pickup and drop-off location based on the target number to reduce the solving time. In addition, while it used the penalty cost to replace the station target status constraint, in this research, this cost is the only component in the objective function, with neither routing cost nor total traveling time being included. By doing so, this problem attempts to

only find a feasible routing schedule and related loading / unloading plan at each station to meet the target inventory level without considering any operational costs. This means that the proposed model cannot tell the difference between two solutions giving the same bicycle inventory level at each station even if their routing costs differ by a large margin.

This research reported here focused on the static bicycle reposition problem with a single vehicle. According to Chemla et al. (2013), usually one district is covered by only one vehicle in the real world. However, the multiple vehicle problems can be decomposed into a single vehicle problem through clustering. Furthermore, this research considered the SBRP problem using both the first and second approaches.

The next chapter, Chapter 3, provides both the integer programming model and the heuristic algorithm for the problem proposed by Ho and Szeto (2014). In contrast to previous research, this research included both the routing and penalty costs in the objective function to enable finding the solution with the minimal operational cost. Furthermore, a new heuristic algorithm was developed to solve the problem. Even through this algorithm uses the two-step (routing first, loading assignment second) method to obtain the heuristic solution, it improves the method for solving the second step by using an auxiliary algorithm to find the loading / unloading plan for the routing schedule under consideration. This improved auxiliary algorithm constructs a special graph and finds the shortest distance from its beginning to its end point, thus, resulting in determining the optimal operation plan for a given routing schedule.

Chapter 4 uses the same basic single vehicle SBRP model but relaxes the station visiting constraint and vehicle routing trip limitation. In this new research, the vehicle can

use multiple trips (i.e. visit the depot multiple times) to complete the reposition event. Although the limitation that each station can be visited at most once each trip is kept, the same station is allowed to be visited multiple times in different trips. In other words, the station visiting limitation is partially relaxed, and each can be visited multiple times in one solution. A VNS-based 1-step heuristic algorithm is proposed to solve this problem. In contrast to the 2-step method, the 1-step algorithm can modify the vehicle routing schedule and the loading / unloading plan at the same time.

In Chapter 5, the visiting limit constraint is further relaxed by being removed from the model. The vehicle can visit any station any number of times without any limitation. In addition, unlike previous research which allowed multiple station visits, this research also allowed multiple trips in the solution, meaning that the vehicle also can visit the depot as well as each station multiple times. Furthermore, it used the 1-step method to solve the problem rather than the 2-step method. Based on our knowledge, this research is the first using a 1-step method for the multiple station visit SBRP problem.

CHAPTER THREE

STATIC BICYCLE REPOSITION PROBLEM WITH SINGLE VEHICLE AND SINGLE TRIP

One of the critical issues in BSS operation is balancing the bicycle inventory level among the various stations in the system. This static bicycle repositioning problem is an extension of the classic VRP problem with one commodity pickup and delivery. This chapter investigates this problem with a single vehicle with three restriction assumptions: the vehicle only can use one *trip* for the repositioning, meaning means it can visit the depot only twice, at the beginning and at the end; all the repositioning must be finished within the given time horizon, meaning no overtime is allowed, and each station can be visited at most once in the repositioning event to balance its bicycle inventory level. To solve the problem, this research provides a mathematical model for the abstracted problem and includes a variable neighborhood search algorithm that has been created to solve it.

Introduction

A bicycle sharing system in a city allows consumers to rent a bicycle from the system, use it for a short time period, and then return it to the system. All the bicycles used in the system are kept in stations at various locations across the city. Each of these stations includes a centralized self-service machine for the renting and return of the

bicycles. Real-time information for each station, which is uploaded into the data center through this self-service machine, includes the detailed records for each bicycle and the stations, such as the number of available bicycles, the empty lockers, and the bicycle usage at each. The key to the success of this system is to ensure customers can rent / return a bicycle to the station when they want to. In other words, the bicycle inventory level at each station should keep a certain level, neither too full nor too empty, which can satisfy both the rent and return needs of the customers. Because of the unbalanced demand for rent and return at each station as well as other factors, the BSS system operator needs to manually rebalance the bicycle inventory level among the various stations to meet that target. This is the problem addressed in this research.

Problem Description

The BSS considered here refers to a self-service rental and return system for bicycles, one that allows consumers to rent a bicycle at any station in the system, use it for a short time or distance, and return it to any station in the system. These stations, which are located at various places in the city, have a constant number of fixed lockers for storing a specified number of bicycles at any given time. The number of bicycles at each station is limited to the number of lockers, and a customer can rent a bicycle if there is at least one available in one of the lockers. Similarly, they also can return a bicycle when there is at least one vacant locker. Based on past research, two critical issues challenging the BSS system are (1) no bicycle is available at the station when the

customer wants to rent one and (2) no vacant locker is available when the customer wants to return one. Both of these issues generate customer dissatisfaction, and a few such disappointments might result in losing customers. Thus, for this system to run effectively, the operator needs to rebalance / reposition the number of bicycles at each station to avoid these two issues, the focus of the SBRP problem considered in this research. This repositioning process occurs at night when there is little or no activity to affect the repositioning process. The entire repositioning process needs to be finished within a given time horizon (e.g. 8 hours' work schedule). For the purposes of this study, three additional constraints have been added to reduce the complexity of the problem: (1) during the repositioning process, each station can be visited no more than once; (2) only one vehicle is used for the repositioning event; (3) there is only one depot in the system, and it has unlimited bicycle inventory and storage space.

Notation and Integer Programming Model

This section abstracts the SBRP problem with mathematical notations. For clarification, *vehicle* is defined here as the transporter used to reposition bicycles among the various stations. The *depot* is defined as the parking lot or distribution center where the vehicle will be parked when it is not in operation. Based on this definition, this research specifies that all vehicle trips start or end at the depot. A *station* is the location where customers can rent or return bicycles. Even though the depot and the stations are

separated by definition, the depot location may be the same as one of the stations. The *station capacity* is defined as the total number of fixed lockers at that station.

In contrast to the research conducted by Ho and Szeto (2014), this study incorporates several realistic considerations in the model. First, Ho and Szeto's objective function considers only the penalty cost, which is the cost related to the difference between the numbers of bicycles after repositioning to the target value at each station. However, the daily operational costs, such as for fuel and labor, are not considered in their objective function. This research includes these operational costs in the objective function in order to obtain a more accurate estimate of the total cost. Second, Ho and Szeto (2014) use a Tabu search to solve the problem. While within the algorithm, the routing schedule for the vehicle is controlled by this search, the associated loading / unloading plan for each routing schedule are reassigned by a group of simple heuristics to adjust the previous existing loading / unloading plan to create a feasible one for the current routing schedule. To improve the second step of the heuristic algorithm, this research uses an auxiliary algorithm to find the optimal loading / unloading plan for each station for any given routing schedule.

Based on the number of vehicles and the number of trips used in solving the problem, Ho and Szeto's (2014) and this research can be defined as the one vehicle, one trip case (SBRP-11). Subsequent work may consider a one vehicle, multiple trip case (SBRP-1M), a multiple vehicle, one trip case (SBRP-M1), and / or a multiple vehicle, multiple trip case (SBRP-MM).

Below are the notations used to describe the SBRP-11 problem.

Sets:

N : the set of all stations. $N = \{1, 2, ..., n\}$.

N_0 : the set of all nodes, including both the stations and the depot. Since the routing both starts and ends at the depot 0, we define 0^+ as start point, and 0^- as end point, meaning $N_0 = \{0^+, 0^-, 1, 2, ..., n\}$

Parameters:

l_i^b : the number of bicycles at station *i* before the repositioning event.

s_i : the number of lockers installed at station *i*, a.k.a. the capacity of station *i*.

t_i : the target number of bicycles planned to be located at station *i*.

c : the capacity of the vehicle.

$g_i(I_i^a)$: the convex penalty function at station *i* with I_i^a bicycles remaining at the station after the repositioning event.

d_{ij} : the distance between node *i* and *j*.

e_{ij} : the total travel time from node *i* to node j.

f_{ij} : the total cost to travel from node *i* to node *j*.

h : the time horizon length for the whole repositioning event.

α : the weight of the penalty cost in the objective function.

β : the weight of the regular operational cost in the objective function.

Decision variables:

X_{ij} : 1 if the *vehicle* visits station *j* immediately after visiting station *i*, otherwise 0.

Q_{ij} : the number of bicycles carried on the *vehicle* when it travels from station *i* to station *j*.

Q_i^L : the number of bicycles loaded into the *vehicle* at station *i*.

Q_i^U : the number of bicycles unloaded from the *vehicle* at station *i*.

W_i : the sub-tour elimination variable for station *i*.

I_i^a : the number of bicycles at station *i* after the repositioning process.

Objective:

Minimize $$\alpha \cdot \sum_{i \in N} g_i(I_i^a) + \beta \cdot \sum_{i \in N_0} \sum_{j \in N_0} f_{ij} \cdot X_{ij} \quad (3.1)$$

Subject to:

$$I_i^a = l_i^b + Q_i^U - Q_i^L \qquad \forall i \in N_0 \quad (3.2)$$

$$Q_i^L - Q_i^U = \sum_{j \in N_0} Q_{ij} - \sum_{j \in N_0} Q_{ji} \qquad \forall i \in N_0 \quad (3.3)$$

$$Q_{0^+}^L = \sum_{j \in N} Q_{0^+ j} \quad (3.4a)$$

$$Q_{0^+}^U = 0 \quad (3.4b)$$

$$Q_{0^-}^U = \sum_{i \in N} Q_{i0^-} \quad (3.5a)$$

$$Q_{0^-}^L = 0 \quad (3.5b)$$

$$Q_{ij} \le c \cdot X_{ij} \qquad \forall i \in N_0, \forall j \in N_0 \qquad (3.6)$$

$$\sum_{j \in N} X_{0^+ j} = 1 \qquad (3.7)$$

$$\sum_{i \in N} X_{i 0^-} = 1 \qquad (3.8)$$

$$\sum_{i \in N_0} X_{ij} = \sum_{l \in N_0} X_{jl} \qquad \forall j \in N \qquad (3.9)$$

$$\sum_{j \in N_0} x_{ij} \le 1 \qquad \forall i \in N \qquad (3.10)$$

$$\sum_{i \in N_0} Q_i^L = \sum_{i \in N_0} Q_i^U \qquad (3.11)$$

$$\sum_{i \in N_0} \sum_{j \in N_0} e_{ij} \cdot X_{ij} \le h \qquad (3.12)$$

$$W_i - W_j + (n+1) \cdot X_{ij} \le n \qquad \forall i, j \in N_0, i \ne j \qquad (3.13)$$

$$X_{ij} \in \{0,1\} \qquad \forall i, j \in N_0 \qquad (3.14)$$

$$Q_{ij} \ge 0, \quad integer \qquad \forall i, j \in N_0 \qquad (3.15)$$

$$Q_i^L \ge 0 \quad integer \qquad \forall i \in N_0 \qquad (3.16)$$

$$Q_i^U \ge 0 \quad integer \qquad \forall i \in N_0 \qquad (3.17)$$

$$W_i \ge 0 \quad integer \qquad \forall i \in N_0 \qquad (3.18)$$

$$I_i^A \ge 0 \quad integer \qquad \forall i \in N_0 \qquad (3.19)$$

The objective function (3.1) is defined as the sum of the penalty cost and regular operational cost for the SBRP repositioning event. Each category of cost is associated with a weight which can be scaled based on the priority between these two cost categories.

Constraint set (3.2) defines the bicycle inventory level for each node after the repositioning event. The inventory level for each station node visited during a trip is equal to the initial inventory minus the number of bicycles picked up or the initial inventory plus the number of bicycles delivered. For each station (except for the depot node), the vehicle stops at most once. Thus, the pickup and drop off event are exclusive, meaning only one event happens at a time. Based on the definition used here, the depot is divided into 2 points, depot start 0^+ and depot end 0^-, meaning constraint 3.2 is also applicable for these split depot points.

Constraint sets (3.3~3.5) define the balancing of the flow of the delivery. For each station in the trip, the total loading/unloading bicycle number at the station equals the difference between the number of bicycles on the vehicle before entering and after leaving the station. As depot start 0^+ is the beginning of the route, it will only load bicycles. On the same principle, depot end 0^- will only unload bicycles.

Constraint set (3.6) ensures that at any time during the repositioning event, the vehicle does not carry more bicycles than its capacity.

Constraint sets (3.7~3.9) form the connection constraint, which ensures the trip is linked. The trip must have the outflow from the depot, the inflow back to the depot and all other visits to the stations connected by the trip.

Constraint set (3.10) ensures that each station will be visited at most once during the trip.

Constraint set (3.11) defines the balancing of the numbers of bicycles loaded and unloaded. During the repositioning event, the total number of bicycles loaded into the vehicle equals the total number of bicycles unloaded from the vehicle.

Constraint set (3.12) defines the time limit for the total repositioning event. The total repositioning event should take no longer than h.

The constraint set (3.13) eliminates any sub-tours in each trip, ensuring every trip includes the depot as the starting and ending point.

The constraints (3.14~3.19) are the sign restrictions for the decision variables in this model.

Limitation of the Integer Programming Model

The SBRP problem is a NP hard problem, meaning solving it with Integer Programming models with large datasets is time-consuming. This section explores determining the capacity or tolerable limit for solving this problem with the proposed IP model. The formulated model is implemented in ILOG OPL find solutions within a specified time frame by applying the IP to a small set of data. Due to the complexity of the problem, it was anticipated that the IP could not find the explicit solution given 329 stations and one depot, the typical size of a city BSS.

The testing instances are solved by a Dell notebook with an Intel Core i5-2520M CPU @ 2.5 GHz. The solution time using ILOG with 7, 8, 9 and 10 stations are shown in Table 3.1 :

Table 3.1: Solving Time Using ILOG for Different Numbers of Stations

	# of Stations							
	7		8		9		10	
Vehicle Capacity	8	10	10	20	10	20	10	20
Time for solution (in seconds)	88	103	2865	1858	1773	714	>24hrs	>24hrs

As this table shows, an increase in the number of stations results in a longer time needed to find the optimal solution. When the number of stations is more than 10, the running times are longer than 24 hours. In these cases, a different approach such as a new heuristic algorithm to determine the optimal routing solution is needed.

Heuristic Algorithm

As the previous analysis indicated, it is time-consuming to use optimization software such as CPLEX or GUROBI to solve the IP model for the SBRP problem involving more than 15 stations as in these cases, a feasible solution cannot be found within a reasonable amount of time (e.g. several hours). To address this issue, it is necessary to develop an efficient heuristic method to obtain the solutions.

In this research, each station can be visited at most once during the entire repositioning process; however, fulfilling the station's request was not maintained as a hard constraint, with the total number of bicycles picked up/dropped off being driven by

balancing the penalty function and the routing cost. For example, if one station exhibited a low penalty cost but was located far from the depot, the optimal solution might allow this station's request to remain unfulfilled to reduce the total routing cost rather than reducing the total penalty cost. Because of such issues, using the routing schedule may not represent the solution to this problem as it not only includes the routing schedule for the vehicle but also the loading / unloading plan for each station visited. The heuristic algorithm for this research used the "routing first, loading assignment second" method to find the heuristic solution. Based on the VNS algorithm, one random routing schedule was generated by the algorithm, and then based on this schedule, the auxiliary algorithm generated a loading / unloading plan for each station visited.

Similar to Ho and Szeto's (2014) work, each solution in this research consisted of two parts: (1) a routing sequence, and (2) a loading / unloading plan based on the routing sequence generated. For clarification, the solution for this problem is defined as $x =< r, a >$, where r represents the routing schedule and a the loading / unloading plan for each station. For the routing sequence, $r = (r_1, r_2, ..., r_\rho)$, where r_i is the station ID for the i^{th} stop in the vehicle routing sequence, $r_i \in \{0,1,...,n\}$, based on the definition, $r_1 = r_\rho = 0$, the routing sequence starts and ends at the depot. Since repositioning every station's bicycle inventory to its target was not a hard constraint, the routing sequence does not have to cover every station in the network, meaning the length of the routing sequence, ρ, is not fixed. The loading / unloading plan for each station, referred to as the assignment sequence and applied only to the those having this event, is defined as

$a = (a_1, a_2, ..., a_\rho)$, where a_i is the loading / unloading bicycle quantity at station r_i in the i^{th} stop position in the vehicle routing sequence and $a_i \in \{-c, ..., c\}$, where c is the capacity of the vehicle. The positive sign of a_i represents the loading of bicycles from the station into the vehicle while the negative sign represents the unloading of bicycles from the vehicle to the station. By definition, the assignment sequence is highly bonded with the routing sequence, meaning the combination of routing sequence and assignment sequence can be used to represent the solution $x =< r, a >= (< r_1, a_1 >, ..., < r_\rho, a_\rho >)$. The tuple $< r_i, a_i >, i \in \{1, ..., \rho\}$ means the i^{th} stop of vehicle route is station r_i, and its loading or unloading bicycle number is a_i at this station.

This research proposes a Variable Neighborhood Search (VNS) based heuristic algorithm in conjunction with an auxiliary algorithm to determine the solution for the SBRP problem. The algorithm will first determine the vehicle routing schedule, and then generate an assignment plan for each station visited in this schedule. The following sections detail both of these algorithms.

Initial Solution Construction

The initial solution is the starting point for the VNS algorithm. A good initial solution, one close to the optimal solution, can help the algorithm reduce the solving time, meaning the quality of the one selected will affect the performance of the algorithm. Since this research is looking for the optimal global solution with no knowledge of where

it is in the solution space, finding a good quality initial solution is a challenge. In general, two basic rules guide the selection of the initial solution: (1) randomness, which ensures that the initial solution is scattered across the solution space. (2) A better objective value which results in an initial solution close to the optimal one.

In this research, the total cost is composed of two parts: (1) the vehicle routing cost and (2) the station inventory penalty cost. As it is difficult to control the former in the construction solution, the initial solution is generated by minimizing the total penalty cost without considering the routing cost. For each station, this research assumes its loading / unloading quantity satisfies its request, meaning that its inventory will be adjusted to its target inventory level after the repositioning event, resulting in a minimal penalty cost. Based on this assumption, we can determine the loading / unloading quantity for every station. For example, at station *i*, the delivery quantity is $l_i^b - t_i$. If $l_i^b > t_i$, station i is considered to be a pickup station, but if $l_i^b < t_i$, it is categorized as a drop off station. All stations with $l_i^b = t_i$ will be considered as ignorable stations and excluded from the initial solution. In this way, all stations are categorized into 2 groups: pickup stations or drop off stations.

The following sections propose two methods for constructing the initial solutions: Random Selection and Penalty Cost Selection.

Random Selection Method

The first, the random selection method, alternatively selects the pickup and drop off process in creating the initial solution. For each trip, it includes one pickup and one drop off process. Initially, it begins with an empty route. The pickup process randomly selects stations from the pickup group to add to the end of the route until the total pickup quantity is accumulated. The pickup process ends when the newly added station, for example station *g*, violates the vehicle capacity, meaning it is not included in the route. The drop off process repeats the same process, replacing pickup stations with drop off ones, the only difference being that the vehicle bicycle inventory decreases as new stations are added. When the next drop off station added violates the vehicle inventory constraint, the drop off process stops and the pickup process resumes. This entire process repeats until the route fills the total time horizon limit or all stations have been covered. Once this process stops, the vehicle routing schedule is determined, and the order in which the stations will be visited is assigned.

Penalty Cost Selection Method

The penalty cost based selection method uses the same procedure as the random selection method to create the initial solution, the only difference being that the selection of the stations during the pickup / drop off process is based on their penalty rather than being randomly done. The stations with higher penalty costs are selected earlier than those with lower penalty costs.

Both methods will be applied to get candidate initial solutions for the VNS algorithm, one candidate initial solution will be randomly selected to be passed to VNS algorithm as the initial solution. But all candidate initial solutions' objective value will be recorded and the best one will be saved as the current best solution to the VNS algorithm.

Variable Neighborhood Search

The VNS algorithm, a recent heuristic algorithm proposed by Mladenović and Hansen (1997), has been used to solve several combinational optimization and global optimization problems efficiently. Specific to the research here, it has been used to solve both multi-depot (Polacek, Hartl, Doerner, and Reimann, 2004; Polacek, Benkner, Doerner, and Hartl, 2008; and Kuo and Wang, 2012) and periodic vehicle routing problems (Pirkwieser and Raidl, 2008; Hemmelmayr, Doerner, and Hartl, 2009; Pirkwieser and Raidl, 2009; and Pirkwieser and Raidl, 2010).

The VNS algorithm is generally constructed based on the local search principle, which uses an efficient algorithm to find a local optimum. Based on its search rule, the local search only makes a change when the new solution is better than the current one. This search criterion helps the algorithm find the local optimum efficiently but at the same time, creates the flaw that it may become stuck in a local valley and not able to find the global optimum. To avoid this flaw, the VNS algorithm uses the neighborhood function to create an incumbent solution in order to escape the local valley. For a given solution x, its neighbor solution, $x' = N(x)$, is the new solution created based on x with

some simple modification. The transformation function $N(\cdot)$, which generates the neighbor solution *x'* from solution *x*, is called the neighborhood function. The simple modification, for example a swap station A with station B delivery sequence in the routing sequence, is a neighborhood function. In the iteration, the algorithm will apply different neighborhood functions to explore different incumbent solutions. The local search method is then applied to each of these to find its local optimum. Once the improved solution has been found, the incumbent is updated by the better solution, and the algorithm begins the next iteration. In this structure, the neighborhood exploration helps VNS avoid being stuck in the local optimum valley and the local search helps it to find a better solution.

The following pseudo code provides the steps of the general VNS algorithm:

Repeat following sequence until the stopping condition is met:

(1) Set $k \leftarrow 1$;

(2) Repeat the following steps until $k = k_{max}$

(a) Shaking. Generate a solution *x'* at random from the k^{th} Neighborhood function $x' = N_k^S(x)$

(b) Local search

(b1) Set $l \leftarrow 1$;

(b2) Repeat following steps until $l = l_{max}$

- Exploration of neighborhood. Find the best neighbor $x'' = N_l^L(x')$
- Move or not. If $f(x'') < f(x')$, set $x' \leftarrow x''$ and $l \leftarrow 1$; otherwise set $l \leftarrow l+1$

(c) Move or not. If this local optimum is better than the incumbent, move there ($x \leftarrow x''$), and continue the search with $N_1 (k \leftarrow 1)$; otherwise, set $k \leftarrow k + 1$

There are 2 types of neighborhood functions used in the VNS algorithm. The first, $N_k^S(x)$, $k = 1,...,k_{\max}$, is used in the shaking phase, which can help the solution escape the local valley, while the second, $N_l^L(x), l = 1,...,l_{\max}$, is used in the local search phase to

find the local optimum. As seen in the pseudo code of the VNS algorithm, these neighborhood functions are important elements of this method.

Neighborhood Functions

The following sections detail the shaking neighborhood functions and the local search neighborhood functions in the VNS pseudo code. The function $N(x)$ creates the new solution by including a small modification made to solution x. To differentiate between the two types of neighborhood functions, more details about the solution for the problem are needed. In general, the solution for this research includes the following information:

1. The stations that are visited in the vehicle routing schedule
2. The station visiting sequence in the vehicle routing schedule
3. The loading / unloading quantity at each station in the vehicle routing schedule

The first point limits the structure of the solution which constrains the outcome range of the solution. For example, let's assume the problem involves 10 stations with only one optimal solution, and its routing sequence covers 8 stations. Solutions with routing sequences that do not cover these 8 stations have no chance to transform to the optimal solution if the neighborhood function changes only the delivery sequence and related delivery assignment. The second and third points affect the routing cost and penalty cost under certain solution structures. When the structure of the solution is fixed, changing elements in points 2 and 3 will provide changes that may reach the best solution.

This research incorporates neighborhood functions which change the structure of the solution into the shaking neighborhood function set (i.e. point 1), while neighborhood functions which change the performance of the solution are classified as the local search neighborhood function (points 2 and 3).

The shaking neighborhood function will change the total number of stations or the stations visited in each trip. The following neighborhoods were created for this research:

1. (D) Delete one station from the routing sequence
2. (A) Add one station to the routing sequence
3. (R) Replace one station in the routing sequence with another station.
4. (D2) Perform the deleting one station neighbor function twice.
5. (A2) Perform the adding one station neighbor function twice.
6. (R2) Perform the replacing one station neighbor function twice.
7. (D3) Perform the deleting one station neighbor function three times.
8. (A3) Perform the adding one station neighbor function three times.
9. (R3) Perform the replacing one station neighbor function three times.

The local search neighborhood function will not improve the performance of the current solution without changing the current solution structure. We construct the following local search neighborhoods:

1. (Swap) Swap 2 stations in a trip
2. (Move) In one trip, move one station to another visiting schedule
3. (2Opt) Perform the 2-Opt cross for the visiting schedule

4. (Swap2) Perform Swap twice
5. (Swap3) Perform Swap three times
6. (Swap4) Perform Swap four times
7. (Swap5) Perform Swap five times
8. (Move2) Perform Move twice
9. (Move3) Perform Move three times
10. (Move4) Perform Move four times
11. (Move5) Perform Move five times
10. (2Opt2) Perform 2Opt twice.
12. (2Opt3) Perform 2Opt three times
13. (2Opt4) Perform 2Opt four times
14. (2Opt5) Perform 2Opt five times

Processing the two types of neighborhoods can modify the key content points 1 and 2 for the solution, i.e. the stations covered in routing sequence and the delivery schedule. However, the VNS method provided here does not include the method for changing the number of bicycles loaded / unloaded at each station, meaning that the VNS algorithm only provides a solution for the routing sequence $r =< r_1, r_2, ..., r_\rho >$, but makes no contribution towards finding an assignment sequence $a =< a_1, a_2, ..., a_\rho >$. Without changing the loading / unloading number at each station, it is impossible to obtain the optimal solution. To address this issue, this research introduces an auxiliary algorithm to generate the assignment sequence based on the vehicle routing sequence created in the

previous step. By embedding this auxiliary algorithm in the current VNS algorithm, all aspects of the solution can be fully modified.

Auxiliary Algorithm for Assignment Sequence

The first step in the heuristic uses the VNS algorithm to generate a vehicle routing schedule for the problem. Based on this vehicle routing schedule, the vehicle routing cost and total delivery time can be determined. However, there is no loading / unloading plan for each station visited. As this research uses the penalty cost for each station, satisfying the request for each (i.e. repositioning each station inventory to the target level) becomes an optional constraint. As a result, the number of bicycles loaded / unloaded at each station visited cannot be determined uniquely, meaning many loading / unloading plans can be associated with the same vehicle routing schedule. The auxiliary algorithm proposed here can provide the optimal loading / unloading plan for a given vehicle routing. Even though it is not the optimal solution for the entire problem, it will guarantee an optimal solution when the VNS algorithm determines the optimal vehicle routing schedule.

As mentioned previously, the solution can be represented by $x =< r, a >= (< r_1, a_1 >,..., < r_\rho, a_\rho >)$. The VNS neighborhood functions change only the vehicle routing sequence $r =< r_1, r_2, ..., r_\rho >$, but not the associated assignment sequence $a =< a_1, a_2, ..., a_\rho >$. In order to know the penalty cost for each station, we must know the status of each station after repositioning. Since the stations not included in the vehicle

routing schedule will not change their status, their penalty cost can be easily calculated, meaning, only the stations in the vehicle routing schedule need to be considered.

Before introducing the auxiliary algorithm, the following are defined for clarity: for the vehicle routing sequence $r =< r_1, r_2, ..., r_\rho >$, the status of the station after repositioning is defined by $< i, p_i, q_i >$, where p_i is the inventory level at station r_i after the repositioning event and q_i is the number of bicycles on the vehicle after it leaves the station. By definition, it is known that the number of bicycles loaded / unloaded at each station visited is equal to its initial inventory minus the inventory after repositioning, i.e. $a_i = l^b_{r_i} - I^a_{r_i} = l^b_{r_i} - p_i$. Because each station is visited only once in the routing schedule, its inventory status changes only once. Considering the balancing of bicycle on the vehicle leads to the equation, $q_i + l^b_{r_{i+1}} = p_{i+1} + q_{i+1}$, which is used to generate all possible status options for the next station visited. For instance, assume a routing schedule $r =< 0, 3, 2, 1, 0 >$, a vehicle capacity *c*=3, an initial inventory at station 2 of 4, $l^b_2 = l^b_{r_3} = 4$, and a capacity at station 2 of 5, $s_2 = s_{r_3} = 5$. In addition, suppose currently we have one status $< i, p_i, q_i >=< 2, 5, 2 >$, meaning that after the repositioning for station r_2=3, there are 5 bicycles left at station 3 and 2 bicycles on the vehicle when it leaves this station. Using the equation $q_i + l^b_{r_{i+1}} = p_{i+1} + q_{i+1}$, we know that $2 + 4 = p_3 + q_3$, meaning all possible status options for station r_3=2, are <3,0,6>, <3,1,5>, <3,2,4>, <3,3,3>, <3,4,2>, <3,5,1>, <3,6,0>. Because of the station capacity limitation, it is impossible to obtain status <3,6,0>. The vehicle capacity constraint excludes status options <3,0,6>, <3,1,5>,

and <3,2,4>. Therefore, all possible status options which begin from <2,5,2> are <3,3,3>, <3,4,2> and <3,5,1>.

The auxiliary algorithm creates an optimal assignment sequence $a =< a_1, a_2, ..., a_\rho >$ for a given vehicle routing schedule $r =< r_1, r_2, ..., r_\rho >$. We define the graph $G_{sp} = (V_{sp}, A_{sp})$ with given routing sequence $r = (r_1, r_2, ..., r_\rho)$. The node set $V_{sp} = V_{start} + V_{end} + V_r$ includes two dummy nodes for the starting and ending points $V_r = \{< i, p_i, q_i > | \; p_i = 1, ..., s_{r_i}, q_i = 1, ..., c, i = 1, ..., \rho\}$ where p_i is the inventory level at station r_i after the repositioning event and q_i is the number of bicycles on the vehicle after its leaves station r_i. Let $V_{start} =< -1, 0, 0 >$ and $V_{end} =< \rho + 1, 0, 0 >$. For depot 0, since we assume it has enough capacity and inventory, its inventory always shows infinity. Each node in the graph represents a transit status of the system during the repositioning process. For clarification, the nodes are divided into different groups based on their visiting sequence. For instance, node $< i, p_i, q_i >$ is categorized into group *i*. Based on this definition, it is known that V_{start} belongs to group -1 and V_{end} belongs to group $\rho + 1$. In the graph designed, only the nodes in adjacent groups have an arc connection. By checking the vehicle capacity constraint, the station capacity constraint, the flow balancing constraints and the time horizon constraint, we can determine whether an arc exists between nodes in adjacent groups. If one exists between node $< i, a, b >$ and $< i + 1, c, d >$, its weight is the penalty cost for station r_{i+1}. In general, the graph looks like following:

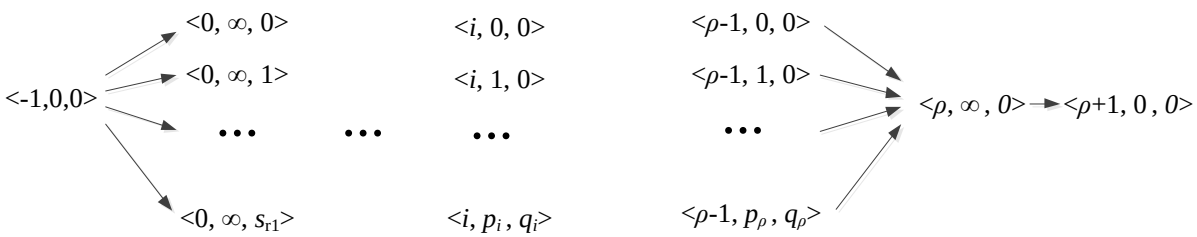

Figure 3.1: General Graph Constructed by a Given Vehicle Routing Schedule

More details about this method can be seen in Example 3.1. Consider the problem with 2 stations. The capacity for the vehicle is $c = 2$. The locker capacity and initial bicycle inventory for station 1 and station 2 are $s_1 = 3$, $s_2 = 4$ and. $l_1^b = 1$, $l_2^b = 2$. Its related inventory target and penalty cost coefficient are $t_1 = 2$, $t_2 = 3$ and $g_1(x) = |x-2|$, $g_2(x) = |x-3|$. A given routing sequence $r_1 = <0,1,2,0>$ results in the following graph:

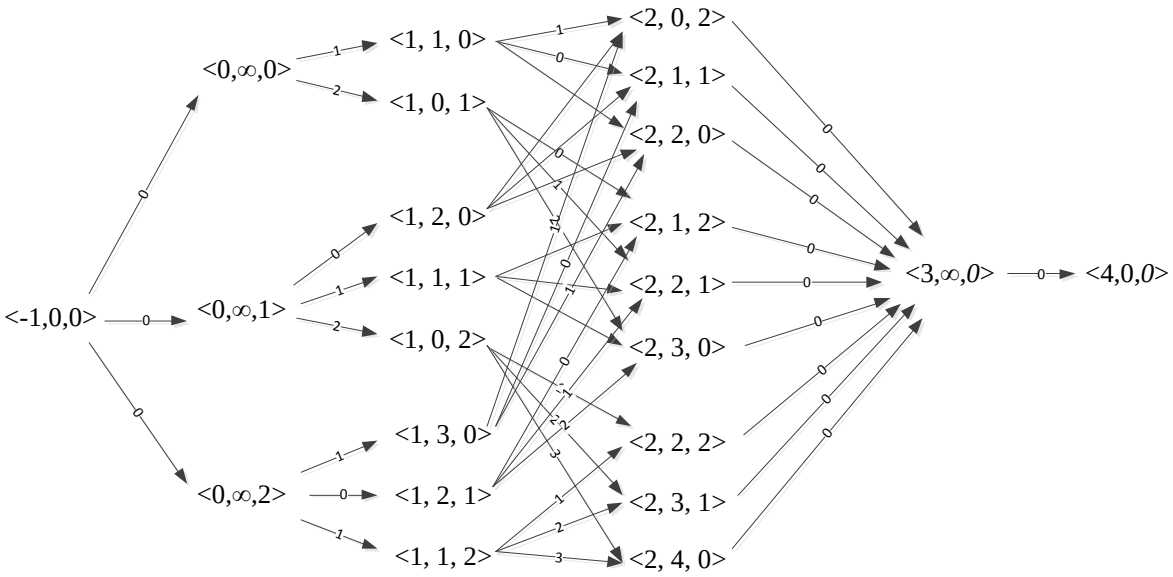

Figure 3.2: Graph Constructed for Vehicle Routing Schedule in Example 3.1

By find the shortest path from the beginning to the end point, we could get the assignment sequence for each station. The shortest path from start point to end point is <-1,0,0> → <0,∞,2> → <1,2,1> → <2,3,0> → <3,∞,0> → <4,0,0> with the total value of 0. The vehicle will pick up 2 bicycles at the depot and drop 1 bicycle at station 1 and drop 1 bicycle at station 2.

Numerical Result

This section evaluates the performance of the algorithm proposed in this research by solving the SBRP problem using datasets of different sizes. In general, two sizes are considered: (1) a small size dataset for which an optimal solution can be found using the IP model, for which we compare the final results provided by the proposed heuristic

algorithm to determine if the proposed heuristic can provide an optimal solution or one close to it. (2) a large dataset for which there is no guarantee that an optimal solution can be found, so we instead compare to the results presented in Ho and Szeto (2014). All tests were conducted using a Dell notebook with an Intel Core i5-2520M CPU @ 2.5 GHz.

1. Small dataset group testing

For the testing of the small datasets, both the heuristic algorithm and the IP model were used to solve the test cases from this set. The IP model was solved using ILOG OPL with CPLEX as the solver. The heuristic algorithm was coded in C++. All the datasets in this group were randomly generated, with the stations being randomly scattered through an area of 100 x 100 and the depot located at (50, 50). The vehicle capacity was fixed at 10. Each station's capacity and target values were also randomly generated. The parameters are listed in Table 1 below:

Table 3.2: Parameters for Datasets Used in the First Testing Group

Parameters	Values
Station location	X ~ U(0,100), Y ~ U(0,100)
Depot location	(50, 50)
Vehicle capacity	10
Station capacity and Target value	U (0, 10)

Table 3.2 shows the results from this group, a total of 12 test cases being listed. The number of stations used in the datasets ranged from 5 to 8. The time horizon used in these datasets ranged from 200 to 350 units. All instances were solved with both the IP model and H1 heuristic algorithms. As the H1 algorithm includes the randomness, each testing instance will be run 30 replications to get the median value as the result for the H1 algorithm.

Table 3.3: All Testing Instances Major Parameters and Results in 1st Testing Group

No.	$\|N\|$	*H*	IP	Gap	IP Time (s)	H1	H1 Time (s)	STD H1	Min H1	Max H1	OPT Time
1	5	200	229.87	0	23	229.87	2	0	229.87	229.87	30/30
2	5	200	255.88	0	25	255.88	2	0	255.88	255.88	30/30
3	5	200	107.47	0	21	107.47	2	0	107.47	107.47	30/30
4	6	250	403.08	0	30	403.08	6	0	403.08	403.08	30/30
5	6	250	637.37	0	34	637.37	7	0	637.37	637.37	30/30
6	6	250	246.26	0	29	246.26	6	0	246.26	246.26	30/30
7	7	300	403.11	0	95	403.11	10	0	403.11	403.11	30/30
8	7	300	691.50	0	102	691.50	11	0	691.50	691.50	30/30
9	7	300	723.56	0	105	723.56	10	0	723.56	723.56	30/30
10	8	350	671.87	0	1856	671.87	12	0	671.87	683.45	30/30
11	8	350	577.91	0	2048	577.91	13	0	577.91	577.91	30/30
12	8	350	681.97	0	1778	681.97	13	0.65	681.97	689.51	29/30

The "|*N*|", "*H*", "IP", "GAP", "IP Time" represent number of stations, time horizon, integer programming model solved result, gap between result and Lower bound, the solving time for integer programming model. The "H1", "H1 Time", "STD H1", "MIN H1", "Max H1", "OPT Time" represent median of H1 result of 30 replications, standard deviation of 30 replication results, minimal H1 result of 30 replications, maximal H1 result of 30 replications, time to get optimal solution in 30 replications.

Comparing the best results found by both the IP model and the heuristic algorithm indicates that for all datasets in the first group, the heuristic algorithm found the optimal solutions for the problems in a solving time much shorter than that for the IP model. Thus, it appears, based on these results, that the heuristic algorithm proposed here performs well when the dataset is small (e.g. the dataset is less than or equal to 8 stations).

2. Large dataset group testing

The dataset used in the second testing group are all large size datasets. According to the results earlier in this chapter, it is unlikely that any particular data set will yield to the IP model within reasonable time (i.e. within in 1 day). So, all test cases are tested by 2 heuristic algorithms: the heuristic algorithm proposed in this research (H1), and Ho and Szeto's algorithm (H0). All the test cases come from the Ho and Szeto (2014) research.

The Ho and Szeto (2014) research's datasets includes 13 instances with different number of stations in it with the range from 100 to 400. In their research, they use 2 levels of time horizon: 9000 and 18000, and 2 levels of vehicle capacity: 10 and 20. By the combination of the time horizon, vehicle capacity and number of stations in the instance, they have 52 testing scenarios.

In this analysis, we run these 52 testing scenarios to get the results for comparison. Because of the randomness in the heuristic algorithm, it is possible to get different outputs result with the same input and testing scenario. It brings the uncertainty for the comparison. To reduce that uncertainty, we run H1 m=30 replications for each testing scenario. Five measure criteria are achieved from these m iterations: Maximal Objective Value (MaxObj), Minimal Objective Value (MinOjb), Average Objective Value (AvgObj), Standard Deviation of Objective Value (StdObj), and Average Solving Time (AvgTime). Ho and Szeto (2014) research does not report whether the objective value shown in their results is the mean value, or the best value of their testing, so we assume it is the average value.

All the testing instances are categorized into 4 sets shown in table 3.4~3.7.

Table 3.4: Test Cases with Time horizon = 9000, and vehicle capacity = 10

$\|N\|$	H0 Obj	H0 Time (s)	H1 MinObj	H1 AvgObj	H1 MaxObj	H1 StdObj	GAP	H1 AvgTime (s)
100	772.20	0.759	749.55	752.93	754.57	1.32	2.5%	153
125	1027.86	1.444	1004.22	1004.40	1007.33	0.93	2.3%	142
150	1254.57	1.438	1222.93	1223.36	1229.67	1.94	2.5%	169
175	1416.83	1.986	1372.59	1375.04	1378.88	1.74	2.9%	184
200	1640.84	2.334	1615.07	1616.67	1617.58	0.71	1.5%	150
225	1897.30	2.989	1874.07	1884.95	1890.18	4.97	0.7%	192
250	2124.02	3.281	2102.83	2112.81	2119.14	4.15	0.5%	160
275	2286.06	3.764	2236.80	2239.73	2243.39	1.89	2.0%	195
300	2513.06	3.703	2497.61	2507.68	2508.53	2.93	0.2%	169
325	2777.69	4.195	2740.33	2741.55	2742.82	0.95	1.3%	202
350	2996.27	3.577	2972.82	2973.85	2976.69	1.05	0.7%	264
375	3161.19	6.195	3118.88	3121.26	3126.68	1.89	1.3%	258
400	3397.68	8.186	3385.16	3385.82	3397.86	3.97	0.3%	289

* These objective values do not include the transportation cost.

Table 3.5: Test Cases with Time horizon = 18000 and vehicle capacity = 10

$\|N\|$	H0 Obj	H0 Time (s)	H1 MinObj	H1 AvgObj	H1 MaxObj	H1 StdObj	GAP	H1 AvgTime (s)
100	688.12	1.250	680.87	666.62	684.14	0.63	3.1%	331
125	940.66	1.675	920.47	891.05	923.55	1.72	5.3%	304
150	1155.42	2.213	1136.79	1116.36	1149.17	0.58	3.4%	207
175	1315.08	3.330	1283.49	1245.74	1287.44	5.05	5.3%	317
200	1536.21	2.737	1511.91	1502.54	1524.32	1.33	2.2%	285
225	1795.81	3.917	1790.20	1732.37	1790.68	2.12	3.5%	293
250	2016.59	6.995	2014.33	1962.53	2020.62	2.48	2.7%	306
275	2178.06	5.947	2118.69	2150.98	2127.75	1.98	1.2%	376
300	2410.37	5.899	2403.37	2376.20	2408.72	4.72	1.4%	325
325	2663.54	9.975	2628.67	2616.55	2643.30	6.45	1.8%	362
350	2890.89	6.619	2864.07	2853.08	2868.33	0.20	1.3%	343
375	3056.93	7.876	2993.41	3028.46	3006.80	1.91	0.9%	382
400	3287.34	10.052	3262.27	3265.17	3270.82	4.02	0.7%	421

* These objective values do not include the transportation cost.

Table 3.6: Test Cases with Time horizon = 9000 and vehicle capacity = 20

$\|N\|$	H0 Obj	H0 Time (s)	H1 MinObj	H1 AvgObj	H1 MaxObj	H1 StdObj	GAP	H1 AvgTime (s)
100	764.13	0.476	680.87	682.53	684.14	0.83	10.7%	684
125	1022.72	0.968	920.47	923.45	923.55	0.95	9.7%	613
150	1248.85	1.249	1136.79	1144.94	1149.17	3.35	8.3%	723
175	1409.79	1.557	1283.49	1285.51	1287.44	1.00	8.8%	789
200	1634.81	1.791	1511.91	1515.11	1524.32	2.95	7.3%	898
225	1892.47	2.865	1790.20	1790.21	1790.68	0.11	5.4%	921
250	2115.55	2.708	2014.33	2019.06	2020.62	1.58	4.6%	969
275	2280.87	2.652	2118.69	2127.05	2127.75	2.80	6.7%	1123
300	2505.99	3.948	2403.37	2406.82	2408.72	1.39	4.0%	1266
325	2763.36	3.315	2628.67	2632.90	2643.30	5.10	4.7%	1607
350	2992.52	2.871	2864.07	2866.38	2868.33	1.22	4.2%	1772
375	3149.95	4.851	2993.41	3005.69	3006.80	4.41	4.6%	1764
400	3393.38	3.642	3262.27	3280.16	3270.82	2.50	3.3%	2215

* These objective values do not include the transportation cost.

Table 3.7: Test Cases with Time horizon = 18000 and vehicle capacity = 20

$\|N\|$	H0 Obj	H0 Time (s)	H1 MinObj	H1 AvgObj	H1 MaxObj	H1 StdObj	GAP	H1 AvgTime (s)
100	667.51	0.961	578.71	580.85	584.60	1.64	13.0%	2029
125	924.07	1.273	790.03	790.33	795.29	1.57	14.5%	1903
150	1143.04	2.797	1000.75	1001.73	1003.05	0.73	12.4%	2362
175	1300.92	1.570	1115.40	1118.4	1119.34	1.17	14.0%	2851
200	1523.18	2.877	1361.36	1368.17	1373.38	3.08	10.2%	2938
225	1779.22	2.443	1584.76	1585.32	1595.78	3.37	10.9%	2186
250	1999.98	3.872	1799.85	1801.90	1808.87	2.69	9.9%	2293
275	2167.81	4.134	1990.56	1991.34	1992.27	0.55	8.1%	2153
300	2393.41	6.962	2192.24	2207.07	2209.12	4.47	7.8%	2694
325	2644.56	6.417	2427.51	2431.69	2431.73	1.27	8.0%	3270
350	2869.10	4.612	2677.72	2678.84	2681.58	1.17	6.6%	3565
375	3028.17	6.594	2849.78	2855.03	2856.35	1.79	5.7%	3503
400	3261.30	5.554	3071.32	3074.26	3085.55	3.91	5.7%	3009

* These objective values do not include the transportation cost.

In these tables, the "|*N*|", "H0 Obj" and "H0 Time" columns were copied from Ho and Szeto (2014)'s research, representing the total station number, Ho and Szeto's heuristic result, and Ho and Szeto's heuristic running time, respectively. When we use the H1 algorithm to solve the same problem, each testing scenario had been run 30 duplications. The "H1 MinObj", "H1 AvgObj", "H1 MaxObj", "H1 AvgTime" columns represent the minimal objective result, the average objective result, the maximal objective result, and the average running time calculated from the 30 duplication results. The "GAP" shows the improvement gap between the H1 and H0 algorithms using the formula GAP = (H0 – H1 AvgObj) / H0. Since in Ho and Szeto (2014)'s research, the vehicle distribution cost was not included in the objective, this research set $\alpha = 1$, $\beta = 0$ to exclude the vehicle distribution cost in the objective when running the test cases.

From a glance view of the result shown in the tables above, the new heuristic we proposed in this research provides better solution than Ho and Szeto (2014)'s research. To support that finding, we will use statistical testing. At first, we use the Anderson-Darling test to check the normality of the raw data. Based on the test result which is illustrated in the probably plots of Tables A.1, A.2, A.3 and A4, the GAP data does not follow a normal distribution. As such, nonparametric statistical tests are required. In this research, the 1-Sample Wilcoxon test was selected to test if the medians of GAP are equal to zero. The hypotheses tested are defined as following:

H_0: median of the GAP is equal to zero

H_1: median of the objective value is greater than zero

Minitab was used for testing and the results are shown in Figures A.5, A.6, A.7 and A.8. The p-values of all tests are less than 0.001 which means that there is sufficient evidence to reject the null hypothesis and conclude that the median of the GAP is greater than 0. By the definition of GAP = (H0 – H1 AvgObj) / H0 and conclusion of the 1-Sample Wilcoxon test, we conclude that the H0 algorithm always provide larger objective value than the H1 algorithm. Since we prefer the minimal objective value, the H1 algorithm can provide better quality solution than the H0 algorithm.

On the other hand, the running time for H1 algorithm is much longer than Ho and Szeto's algorithm. Because we proposed auxiliary algorithm to provide the optimal solution for the given vehicle routing schedule, it consumes a lots of time when the number of station and vehicle capacity increased in the testing scenario. Even through the running time increased a lot, but total solving time is still within a reasonable range (the max running time for a scenario with 400 stations is within 1 hour). The Ho and Szeto (2014)'s research has much shorter running time (less than 10 seconds).

These results suggest, in general, the new heuristic algorithm can provide better solution but will take longer time.

Conclusion and Future Work

This research presented a VNS-based heuristic algorithm with an auxiliary algorithm to solve the static bike repositioning problem, one using the routing first, loading assignment second approach to find the heuristic solution for the problem.

Computational results show that this heuristic performs well when the dataset is small. It also gives a good solution when the dataset size is large but takes a long time to solve the problem. The contributions of this research are the following: (1) It includes the operation cost in the objective function; (2) It proposes an auxiliary algorithm to find the optimal assignment plan for a given vehicle routing. Future work will extend this research by developing a heuristic to allow the vehicle to visit the station more than once and developing one for multiple vehicles. Furthermore, we will try to improve the efficiency of the auxiliary algorithm with new technic, such as using the mixed integer programming model to solve the assignment plan for the algorithm.

CHAPTER FOUR

STATIC BICYCLE REPOSITIONING PROBLEM WITH A SINGLE VEHICLE AND MULTIPLE TRIPS

This chapter approaches the bicycling repositioning problem by relaxing the constraint concerning the number of trips made by the vehicle, investigating the realistic assumption that it can visit both the depot and each station multiple times to complete the repositioning process. A VNS based algorithm was developed to solve the problem. As opposed to the previous chapter, we proposed a 1 step strategy to construct the routing schedule and loading assignment at the same time rather than using the 2 step "routing first, loading assignment second" strategy.

Introduction

The previous chapter solved a static bicycle rebalancing problem with a single vehicle, developing both an MIP model and a VNS algorithm to do so. The research presented in this chapter extends previous studies in several aspects, relaxing the constraints to make the research problem more realistic. As before, the static bicycle rebalance problem with a single vehicle for a bicycle sharing system is used here. In general, the bicycle sharing system allows customers to rent a bicycle, use it for a short time period, and then return it to the system. These bicycles are kept in stations located throughout the city, which use a centralized self-service machine that, in addition to

facilitating the business transaction, uploads real-time, detailed information to the data center on the number of available bicycles, the number of empty lockers, and the bicycle usage at each. One of the most important elements of this system is ensuring that customers can rent / return a bicycle to a station, meaning the bicycle inventory level at each should be kept at a level that is neither too full nor too empty to satisfy the needs of the users. Because of the unbalanced demand for rentals and returns as well as other factors, the BSS system operator manually rebalances the bicycle inventory level among the stations to meet this target level. This situation is the focus of this research.

Problem Descriptions and Terms

This research investigates repositioning the bicycle inventory level among various stations to their target values using a single vehicle within a specified working time horizon. If a station has not been repositioned to its target inventory level by the end of this horizon, it will be assessed a penalty cost. The operation cost, which is composed of the fuel and labor expenses, is highly related to the vehicle traveling time. The entire repositioning event should be completed within a given time horizon, meaning no overtime is allowed. The objective for this problem is to create a solution for repositioning the bicycle sharing system with minimal total operation and penalty costs.

To clarify this problem, we define the terms and delimitations for this research. The *Vehicle* is defined as the mode of transportation used to carry the bicycles among the various stations with the capacity to carry at most c units. A *station* is defined as the place

where the customers can rent or return bicycles, and the total number of lockers at a station is defined as the *station capacity*. Each *station* has a finite bicycle inventory and *station capacity*. The *depot* is the distribution center and warehouse for both the *vehicle* and the bicycles; the *vehicle* begins and ends delivery from the *depot* which has an unlimited bicycle inventory and *station capacity*. In addition, a *trip* is defined as the vehicle routing sequence beginning at the *depot* and continuing through several *stations* before returning to the *depot*. The *visit point* represents the *station* or *depot* visited in the vehicle routing sequence. The *target* value is the designated inventory level for each *station* where the bicycles are repositioned. A *visit point* is a *pickup station* when the current inventory on at the time of visit is greater than its target value, while a *visit point* is a *drop off station* when its current inventory is less than its *target* value. If the station receives multiple visits, it could be both a *pickup station* (its inventory is greater than the *target*) and a *drop off station* (its inventory is less than the *target*) a different times in the solution. Furthermore, if a *station* is visited multiple times in the solution, it is defined as a *complex station* in the solution; if not, it is a *simple station* in the solution.

In previous research, the SBRP solved included several constraints. It allowed only one trip, and further, each station could be visited no more than once. With these maximal one-time visit constraints, it was impossible to remove the penalty cost completely when stations experienced a large number drop off or pick up requests even when the time and inventory were available. In this research, the *vehicle* is allowed multiple *trips* to reposition the bicycles among the *stations*; however, for each *trip*, the maximal one-time visit constraint for each *station* remains, but the same *station* can be

visited multiple times in different *trips*. With this relaxation, the *station* is allowed multiple visits in the solution.

Heuristic Algorithm

As pointed out by the Ting and Liao (2013) and Ho and Szeto (2014) research, the SBRP problem is a NP hard problem. It is unreasonable to solve the large scale, realistic SBRP problem with exact algorithm, such as the mixed integer programming model. These exact algorithms will cause the solving time increased exponentially as the station size increases, meaning it is difficult to find an optimal solution within a reasonable time for an SBRP problem involving a large number of stations using this model. This situation is addressed by using a heuristic algorithm; this research proposes using a VNS algorithm to find the solution for a SBRP problem. This chapter first analyzes the structure of the solution for the SBRP problem, including defining the appropriate symbols; then it provides the pseudo code for the VNS algorithm to introduce the structure of the heuristic algorithm, and finally it discusses the details of the algorithm and related terms such as initial solution and neighborhood functions, among others.

Analysis of the SBRP Problem Solution

Before the heuristic algorithm is introduced, it is necessary to understand the solution structure of the SBRP problem. In general, this solution contains the vehicle

routing sequence and the loading / unloading plan for each station visited. In this research, the solution is composed of two parts: (1) a qualified vehicle routing sequence, the travel time of which does not violate the time horizon limitation, and (2) the associated loading / unloading plan for each station for this routing sequence. For consistency, the same symbols $x =< r, a >$ used in the previous chapter to define a solution are also used here. The r represents the routing schedule; and a the loading / unloading plan (i.e. the assignment plan) for each station for routing sequence r. The routing sequence is defined as $r = (r_{0^+,1}, r_{11}, ..., r_{\rho_1 1}, r_{0^-,1}, ..., r_{0^+,k}, r_{1k}, ..., r_{\rho_k k}, r_{0^-,k})$, where the $r_{p,k}$ is the station ID for the p th stop in the k th *trip*, and ρ_k is the total number of stations visited in k th *trip*. The r_{0^+} and r_{0^-} represent the depot start point and end point, respectively. Since no limitation is set for the max number of trips used in one solution, the variable k, i.e. the total number of *trips* used in the solution, is uncertain; however, we could get the upper bound for the number of trips in the solution by the time horizon limitation. Suppose μ is the travel time from the depot to the closest station, the shortest travel time for a trip will be 2μ, i.e. the trip just visit one station. By the time horizon limitation, we can get upper bound for trip number in solution, $k^{\max} = \lceil T / (2\mu) \rceil$, where T is the time limit horizon.

The loading / unloading plan associated for routing sequence r, referred to as the assignment plan, is $a = (a_{0^+,1}, a_{11}, ..., a_{\rho_1 1}, a_{0^-,1}, ..., a_{0^+,k}, a_{1k}, ..., a_{\rho_k k}, a_{0^-,k})$, where $a_{p,k}$ is the number of bicycles loaded at station $r_{p,k}$ at the p th stop in the k th *trip*. By using the routing sequence r and its related assignment plan a, it is easy to calculate the inventory

level on the vehicle / at the station when the vehicle departs from each location. We define $v_{p,k}$ as the inventory level on the vehicle when it leaves station $r_{p,k}$ on *p* th stop in the *k* th *trip* and $I_{r_{p,k},k}$ as the inventory level at station $r_{p,k}$ when vehicle has left. All values are calculated with the formulas $v_{p,k} = v_{p-1,k} + a_{p,k}$, and $I_{r_{p,k},k} = I_{r_{p,k},k-1} - a_{p,k}$, while special cases use $v_{0^+,1} = a_{0^+,1}$, $I_{i,0} = l_i^b$, *i*=0,1,2,…,n;

The routing sequence *r* and the assignment plan *a* are critical elements in the solution as well as being highly connected. Although the inventory level on the vehicle and at the station can be calculated from the solution $x =< r, a >$, it is easier to check the feasibility of the solution with these 2 variables. Thus, the inventory level on the vehicle and at the station is included in the solution, resulting in a new formula, called the full solution:

$$x =< r, a, v, I >= (< r_{0^+,1}, a_{0^+,1}, v_{0^+,1}, I_{r_{0^+,1},1} >, ..., < r_{0^-,k}, a_{0^-,k}, v_{0^-,k}, I_{r_{0^-,k},k} >)$$

The tuple $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ indicates that the *p* th stop on the *k* th *trip* is *station* $r_{p,k}$, and the number of bicycles for loading / unloading is $a_{p,k}$; the inventory level on the vehicle after this repositioning is $v_{p,k}$, and the inventory level left at the station is $I_{r_{p,k},k}$.

Variable Neighborhood Search Algorithm

The variable neighborhood search algorithm, developed by Mladenovic in 1997, is a meta-heuristic algorithm based on a local search algorithm used to solve global optimization problems. Below are the pseudo code steps of the general VNS algorithm:

Repeat following sequence until the stopping condition is met:
Set $k \leftarrow 1$;
Repeat the following steps until $k = k_{max}$
Shaking. Generate a solution x' at random from the k^{th} Neighborhood function $x' = N_k(x)$
Local search
(b1) Set $l \leftarrow 1$;
(b2) Repeat following steps until $l = l_{max}$

- Exploration of neighborhood. Find the best neighbor $x'' = N_l(x')$
- Move or not. If $f(x'') < f(x')$, set $x' \leftarrow x''$ and $l \leftarrow 1$; otherwise set $l \leftarrow l+1$

Move or not. If this local optimum is better than the incumbent, move there ($x \leftarrow x''$), and continue the search with $N_1 (k \leftarrow 1)$; otherwise, set $k \leftarrow k + 1$

In this research, we use the combination of the insert point function $I(x)$ and the delete function $D(x)$ to generate the shaking neighborhood function, while the improvement function $P(x)$ is used to create the local search neighborhood function. All three functions will be discussed in detail in later sections.

The Initial Solution

Analyzing the pseudo code indicates that the initial solution is a good starting point the start point for the VNS algorithm. Even using the same search algorithm, a good initial solution can help the algorithm reduce the total search time. However, without sophisticated knowledge of the system and a deep understanding of the research issue, it

is very difficult to determine good suggestions for these starting points. Furthermore, this research is providing a general method for all SBRP problems rather than only for a certain case. Thus, two basic principles are used here to generate the initial solutions: (1) Randomness principles. The randomness property makes the initial solution randomly scattered throughout the solution space, thereby increasing the robustness of the algorithm. (2) Quality principle. In general, good solutions (i.e. those with better objective values) should share some property that makes them able to obtain better objective values. An initial solution with better objective values should be close to the global optimum. In this research, the stations are divided into 2 categories: (1) Pickup Station, which reduces the penalty cost because bicycles are picked up here (i.e. the current inventory level is greater than its target inventory level). (2) Drop off Station, which reduces the penalty cost because bicycles are dropped off here (i.e. the current inventory level is less than its target inventory level).

In this research, the following five methods are used to generate the candidate initial solutions, and one of them (selected at random in each run) are selected to pass to the VNS algorithm as the initial solutions, also, the best of all these candidate solution are saved as the current best solution for VNS algorithm:

1. Randomly Single Alternative Selection:

This method selects the pickup and drop off stations alternately. Initially, a pickup station $r_{1,1}$ is selected and added to the routing sequence. The loading quality at the selected station is defined as $a_{1,1} = \min\{c, l^b_{r_{1,1}}\}$. If the $l^b_{r_{1,1}} > c$, then $v_{1,1} = c$, and the station

$r_{1,1}$ remains on the selection list and its current inventory level is updated. If $l^b_{r_{1,1}} \le c$, then $v_{1,1} = l^b_{r_{1,1}}$, and the station $r_{1,1}$ is removed from the selection list. Next, a drop off station $r_{2,1}$ is randomly selected and added to the routing sequence. The unloading quality at station $r_{2,1}$ is defined as $a_{2,1} = \min\{v_{1,1}, s_{r_{2,1}} - l^b_{r_{2,1}}\}$. The inventory level on vehicle $v_{2,1}$ is updated after $a_{2,1}$ has been determined. If the inventory level meets the target level, station $r_{2,1}$ is removed from the selection list; otherwise, it remains, and its current inventory level is updated. Repeating these two steps generates a routing sequence and a related loading / unloading plan for the trip. If the newly selected station $r_{1,2}$ remains for current trip, *depot* $r_{0^-,1}, r_{0^+,2}$ is added to the current routing sequence. All inventories on the vehicle are unloaded at the *depot*, and the next trip is started following the process just described. However, the $r_{1,2}$ is not added into routing sequence but is used as the sign to add the *depot*. The entire process ends when the total travel time violates the time horizon limitation.

2. Randomly Full Load Alternative Selection:

This method selects full load pickup and drop off stations alternately using a strategy similar to the Randomly Single Alternative Selection method. The only difference is that pickup stations are continually added until the inventory level on the vehicle is full. Then drop off stations are selected until the entire inventory on vehicle is dropped off.

3. Penalty Cost Priority Selection:

This method selects and adds the station which gives the best penalty cost reduction to the routing sequence. It calculates this value for each station based on the current inventory level on the vehicle $v_{p,k}$, the vehicle capacity *c*, the station capacities s_i and their current inventory level $I_{i,k-1}$. If station *i* is a drop off station, its best reduction is $\delta = \min_{q=1,\ldots,\min\{v_{\rho_k,k}, s_i - I_{i,k-1}\}} \{g_i(I_{i,k-1}+q) - g_i(I_{i,k-1})\}$. If station *i* is a pickup station, its best reduction is $\delta = \min_{q=1,\ldots,\min\{c - v_{\rho_k,k}, I_{i,k-1}\}} \{g_i(I_{i,k-1}-q) - g_i(I_{i,k-1})\}$. The station with the largest penalty cost reduction is added to the routing sequence. The station first selected for each *trip* is a special case. Since infinite capacity and inventory are assumed at the depot, either a pickup or drop off station can be selected as the first station in the *trip* route. If it is a pickup station, no action is done at the depot; if a drop off station, the required number will be picked up at the depot starting point. Similar to the strategy used in previous method, once the selected station is added to the current *trip*, the depot is also added, and the process is repeated until the time horizon limitation is reached.

4. Travel Cost Priority Selection:

This method selects the station which adds the smallest travelling cost to the routing sequence. It uses the same steps as the Penalty Cost Priority Selection, with one difference: the criterion for choosing the station changes from the largest penalty cost reduction to the smallest travelling cost increase.

5. Cost Ratio Priority Selection:

This method selects and adds the station with the best ratio between the penalty cost reduction and the travelling cost increase to the routing sequence. If the current status is $< r_{\rho_k,k}, a_{\rho_k,k}, v_{\rho_k,k} >$, the cost ratio for each qualified station is obtained by dividing the best penalty cost reduction by the travel time. If station *i* is a drop off station, its cost ratio is $ratio_i = \min_{q=1,\ldots,\min\{v_{\rho_k,k}, s_i - I_{i,k-1}\}} \{g_i(I_{i,k-1}+q) - g_i(I_{i,k-1})\} / e_{r_{\rho_k,k},i}$. If it is a pickup station, then it is represented by $ratio_i = \min_{q=1,\ldots,\min\{c - v_{\rho_k,k}, I_{i,k-1}\}} \{g_i(I_{i,k-1}-q) - g_i(I_{i,k-1})\} / e_{r_{\rho_k,k},i}$. The station with the largest ratio will be selected and added to the routing sequence. Except for this selection criterion (i.e. best ratio rather than the best penalty cost reduction), the remaining steps in this method are the same as for the Penalty Cost Priority Selection method.

The Removal Neighborhood Function

The removal neighborhood function *R*(*x*), which is the set of the feasible neighbor solutions obtained by applying removal moves to the current solution *x*=<*r*,*a*,*v*,*I*>, removes points visited (either station or depot) from the vehicle routing sequence *r*, as well as its loading / unloading plan from *a* at the same time. The remaining routing sequence *r*' and loading / unloading plan *a*' are the new generated removal neighborhood *x*'=*R*(*x*). However, the new on vehicle inventory level *v*' may not be feasible because it

violates the vehicle capacity limitation (remove a drop off station or the depot) or non-negative sign limitation (remove a pickup station or the depot). This function only changes the loading / unloading plan for the predecessor station, the successor station and relative depot of the station removed to generate a new feasible solution $x'' = <r',a'',v''>$. If the removed point is a station, a maximum of four feasible solutions are generated with the same routing sequence r' but with different loading / unloading plan a''. If the removed point is a depot, a maximum of three feasible solutions are generated. A more detailed description of this method can be found in the next section. To be consistent, the removed point is assumed to be $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ in solution x.

In this research, even though each station can be visited only once in each *trip*, the same station can be visited multiple times in different *trips*. So the station can be visited multiple times across the solution level. In order to distinguish the multiple visited stations and once visited station in the solution, we define *simple station* as one that is visited only once in the solution, while a *complex station* is one visited more than once.

Neighborhood function for simple stations

This neighborhood function is applied when the stations affected are all *simple stations*. This section discusses the simple remove function *SR*(x) and simple insert function *SI*(x).

Removed station is a pickup station

If the removed point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ is a pickup station, a maximum of three feasible solutions can be generated. The number of bicycles picked up at the station can be instead picked up and carried from the depot, the previous station or the successor station as long as the total vehicle capacity is not violated.

1. Pickup from the Depot - Adjust $a_{0^+,k}$:

Denote η_1 as the residual vehicle capacity for the *k* th *trip* before visiting station $r_{p,k}$. Thus, $\eta_1 = \min_{t=0^+,1,\ldots,p-1}\{c - v_{t,k}\}$. For $a_{p,k} \le \eta_1$, the $a_{p,k}$ bicycle can be picked at the depot at the beginning of *k* th *trip*, $a'_{0^+,k} = a_{0^+,k} + a_{p,k}$ as shown in Example 1 below.

Example 1: Suppose one solution includes only one *trip* which visits 3 stations, $s_1 = s_2 = s_2 = 8$, $l_1^b = l_2^b = l_2^b = 3$. The vehicle *c*=10, and the depot is assumed to have infinite capacity. This feasible solution *x* includes the routing sequence $r =< 0^+, 3, 2, 1, 0^- >$ and loading / unloading plan $a =< +1, +2, +2, -5, 0 >$. *Station* 2 (the third entry) in this solution is removed. The new generated solution *x*' is shown in Figure 4.1.

Example 1

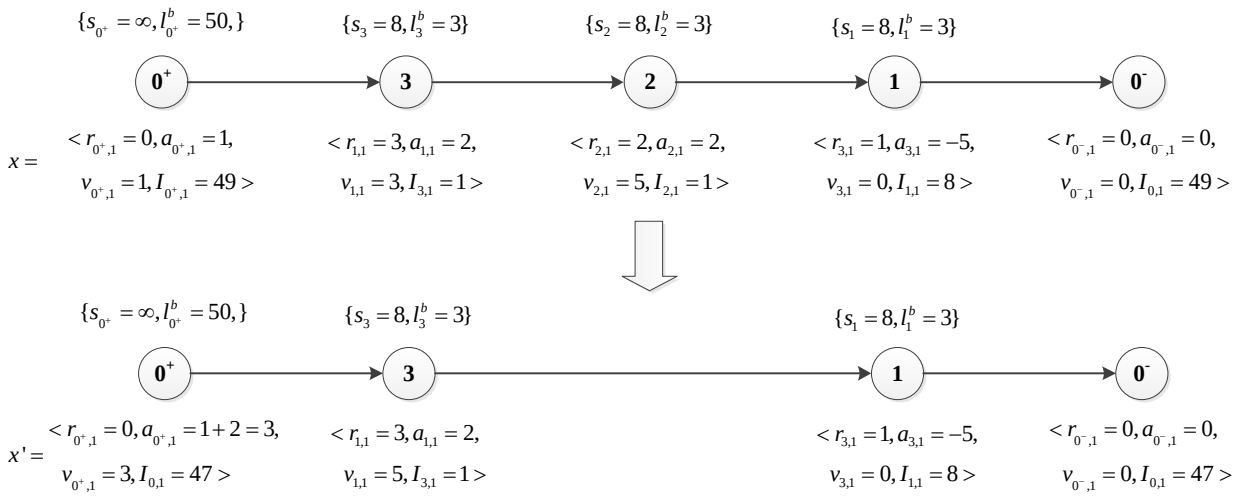

Figure 4.1: Removal neighborhood function, remove pickup station and adjust at depot start point

2. Adjust the amount from the previous station - Adjust $a_{p-1,k}$

If the predecessor station $r_{p-1,k}$ has enough vehicle capacity to pick up the quantity removed (i.e. $v_{p-1,k} + a_{p,k} \le c$) and has enough bicycles stored at the station $r_{p-1,k}$ (i.e. $a_{p-1,k} + a_{p,k} \le I_{p-1,k-1}$), then the vehicle can pick up more / drop off fewer bicycles at station $r_{p-1,k}$ to cover the removed number (i.e. $a'_{p-1,k} = a_{p-1,k} + a_{p,k}$). Examples 2 and 3 illustrate the case when the predecessor is a pickup and drop off station.

Example 2: Suppose one solution includes one *trip* that visits 3 stations, $s_1 = s_2 = s_2 = 8$, $l^b_1 = l^b_2 = l^b_2 = 3$. The vehicle c=10. This feasible solution x includes routing sequence

$r =< 0^+, 3, 2, 1, 0^- >$ and loading / unloading plan $a =< 0, +1, +2, -3, 0 >$. Station 2 is removed from this solution. The new generated solution *x*' is shown in Figure 4.2.

Example 2

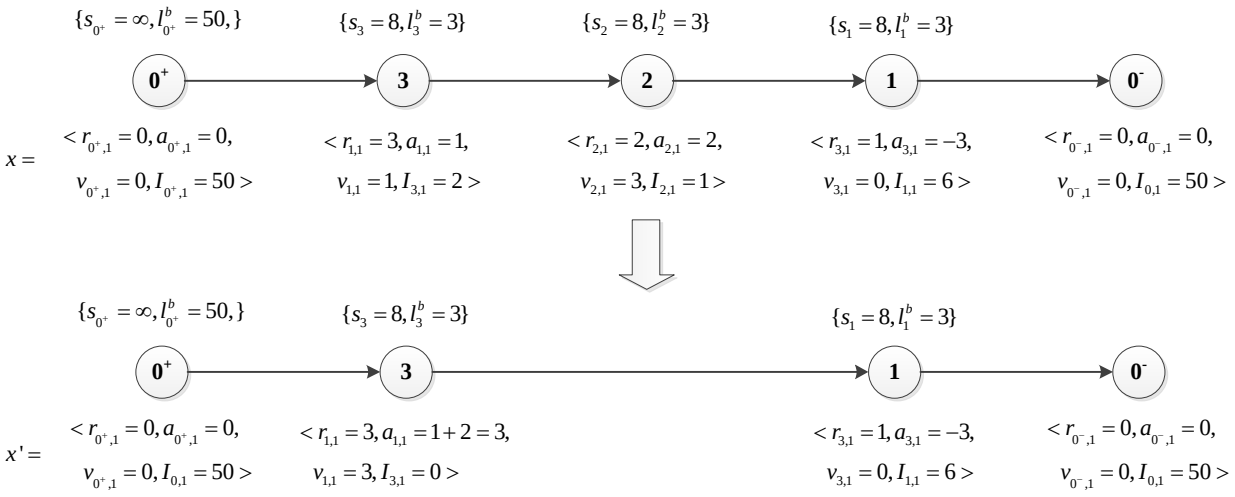

Figure 4.2: Removal neighborhood function, remove pickup station and adjust predecessor pickup station

Example 3: Suppose one solution includes one trip visiting 3 stations, $s_1 = s_2 = s_2 = 8$, $l^b_1 = l^b_2 = l^b_2 = 3$. The vehicle *c*=10. This feasible solution x includes routing sequence $r =< 0^+, 3, 2, 1, 0^- >$ and loading / unloading plan $a =< +3, -1, +2, -4, 0 >$. Station 2 is removed from this solution. The new generated solution *x*' is shown in Figure 4.3.

Example 3

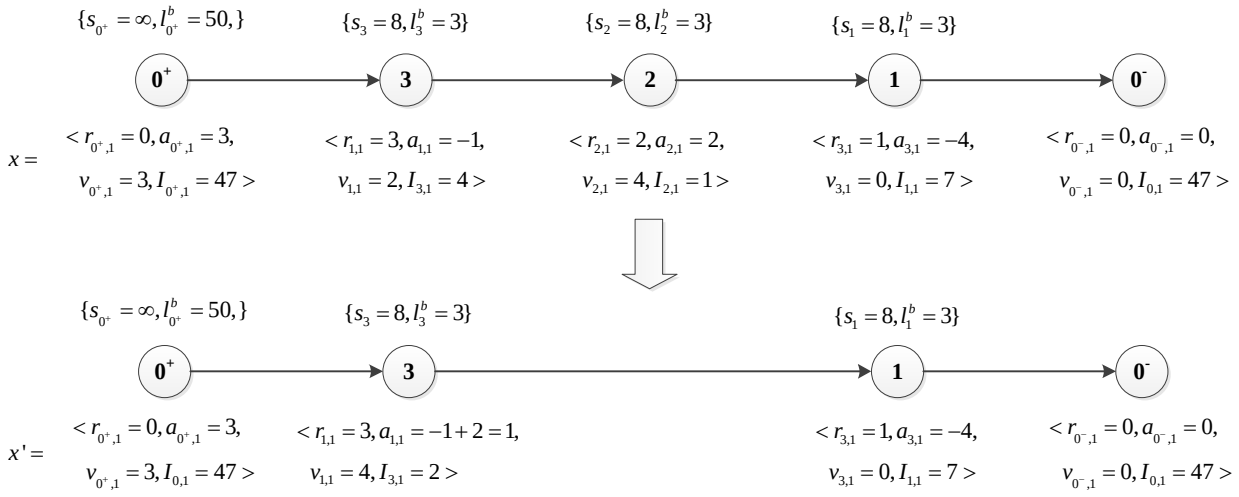

Figure 4.3: Removal neighborhood function, remove pickup station and adjust predecessor drop off station

3. Adjust the amount from the successor station - Adjust $a_{p+1,k}$:

Because we know that the previous solution is feasible, the vehicle capacity limitation should be automatically qualified if station $r_{p-1,k}$ picks up the quantity removed. If the successor station has enough inventory capacity to cover this number (i.e. $a_{p,k} + a_{p+1,k} \le I_{p+1,k-1}$), then station $r_{p+1,k}$ can pick up fewer to cover the number removed (i.e. $a'_{p+1,k} = a_{p+1,k} + a_{p,k}$). Examples 4 and 5 illustrate the case when the successor is a pickup and drop off station.

Example 4: Suppose one solution includes one *trip* visiting 3 stations, $s_1 = s_2 = s_2 = 8$, $l^b_1 = l^b_2 = l^b_2 = 3$. The vehicle c=10. The feasible solution x includes routing sequence

$r =< 0^+, 3, 2, 1, 0^- >$ and loading / unloading plan $a =< 0, +1, +2, +1, -4 >$ Station 2 is removed from this solution. The new generated solution *x*' is shown in Figure 4.4.

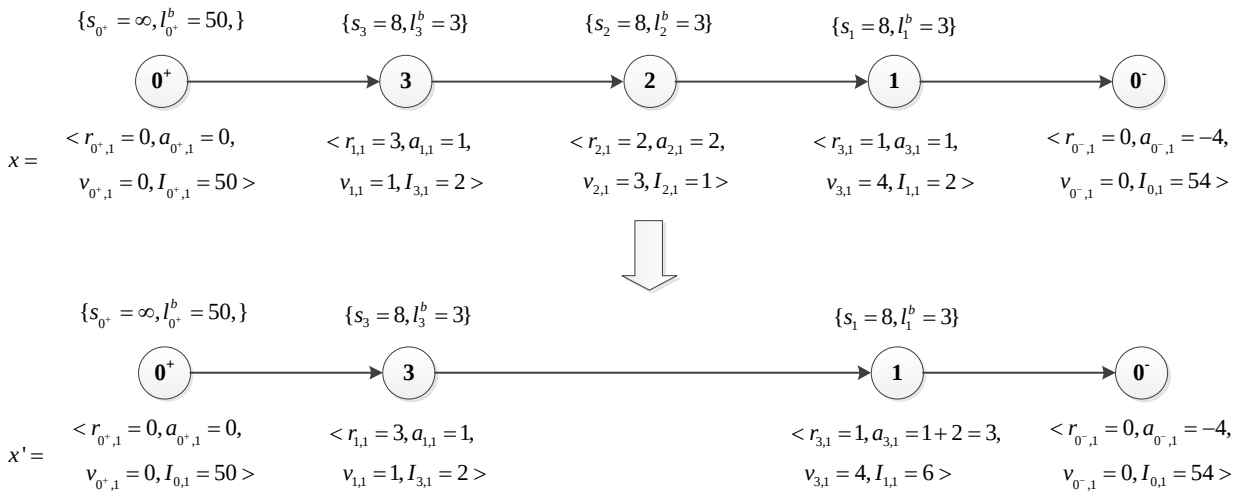

Figure 4.4: Removal neighborhood function, remove pickup station and adjust successor pickup station

Example 5: Suppose one solution includes one trip visiting 3 stations, $s_1 = s_2 = s_2 = 8$, $l^b_1 = l^b_2 = l^b_2 = 3$. The vehicle *c*=10. This feasible solution *x* includes routing sequence $r =< 0^+, 3, 2, 1, 0^- >$ and loading / unloading plan $a =< 0, +1, +2, -1, -2 >$. Station 2 is removed from this solution. The new generated solution *x*' is shown in Figure 4.5.

Example 5

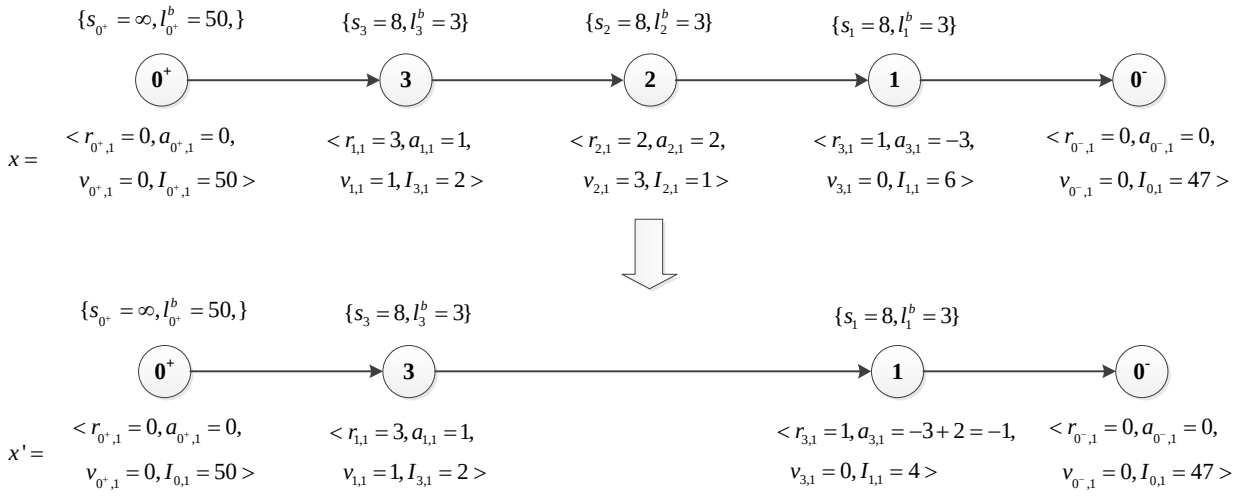

Figure 4.5: Removal neighborhood function, remove pickup station and adjust successor drop off station

Removed point is a drop off station

If the removed point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ is a drop off station, a maximum of three feasible solutions may be generated. The number of bicycles dropped off up at the station can be instead dropped off at the depot, the previous station or the successor station as long as the total vehicle capacity is not violated.

1. Adjust the amount at the Depot - Adjust $a_{0^-,k}$:

Denote η_2 as the residual vehicle capacity for the *k* th *trip* after visiting station $r_{p,k}$. Thus, $\eta_2 = \min_{t=p+1,\ldots,\rho_k}\{c - v_{t,k}\}$. If $\left|a_{p,k}\right| \le \eta_2$, then the $\left|a_{p,k}\right|$ extra bicycles can be

dropped at the depot at the end of *k* th *trip*, $a'_{0^-,k} = a_{0^-,k} + a_{\rho_k k}$. Example 6 below shows this case.

Example 6: Suppose one solution includes one trip visiting 3 stations, $s_1 = s_2 = s_2 = 8$, $l_1^b = l_2^b = l_2^b = 3$. The vehicle *c*=10, and the depot have enough capacity, for example $l_{0^+}^b = 50$. This feasible solution *x* includes routing sequence $r = < 0^+, 3, 2, 1, 0^- >$ and loading / unloading plan $a = < +1, +2, -2, -1, 0 >$. Station 2 is removed from this solution. The new generated solution *x*' is shown in Figure 4.6.

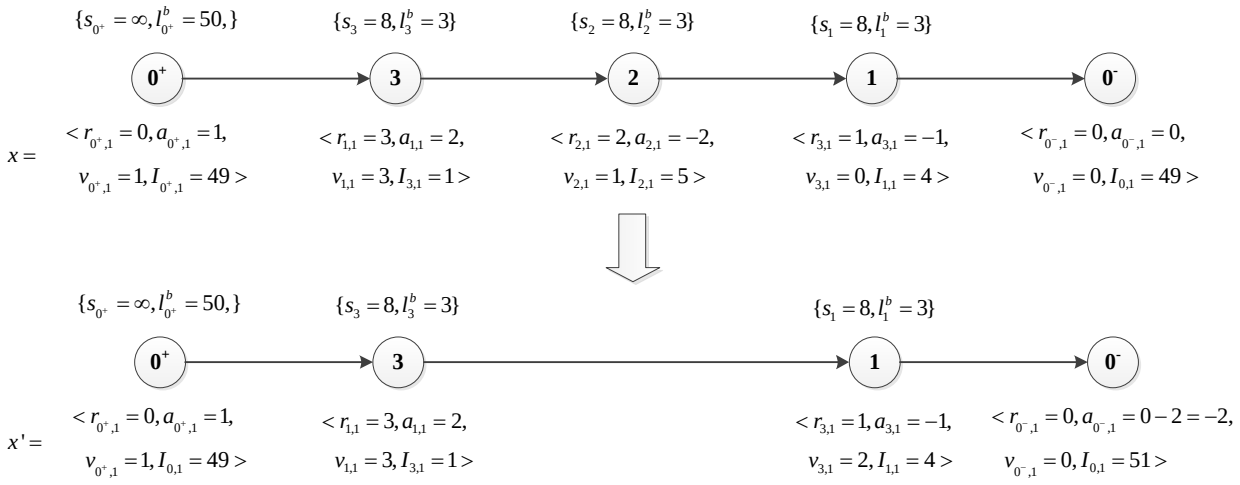

Figure 4.6: Removal neighborhood function, remove drop off station and adjust depot end point

2. Adjust the amount at previous station - Adjust $a_{p-1,k}$:

If the predecessor station $r_{p-1,k}$ has enough bicycles stored at station $r_{p-1,k}$ (i.e. $a_{p-1,k} + a_{p,k} \le I_{p-1,k-1}$), then station $r_{p-1,k}$ can drop off more / pick up fewer to cover the number removed (i.e. $a'_{p-1,k} = a_{p-1,k} + a_{p,k}$). Examples 7 and 8 illustrate the case when the predecessor is a pickup and drop off station.

Example 7: Suppose one solution includes one trip visiting 3 stations, $s_1 = s_2 = s_2 = 8$, $l_1^b = l_2^b = l_2^b = 3$. The vehicle c=10 and the depot have enough capacity, for example $l_{0^+}^b = 50$. This feasible solution x includes routing sequence $r =< 0^+, 3, 2, 1, 0^- >$ and loading / unloading plan $a =< 0, +3, -2, -1, 0 >$. Station 2 is removed from this solution. The new generated solution x' is shown in Figure 4.7.

Example 7

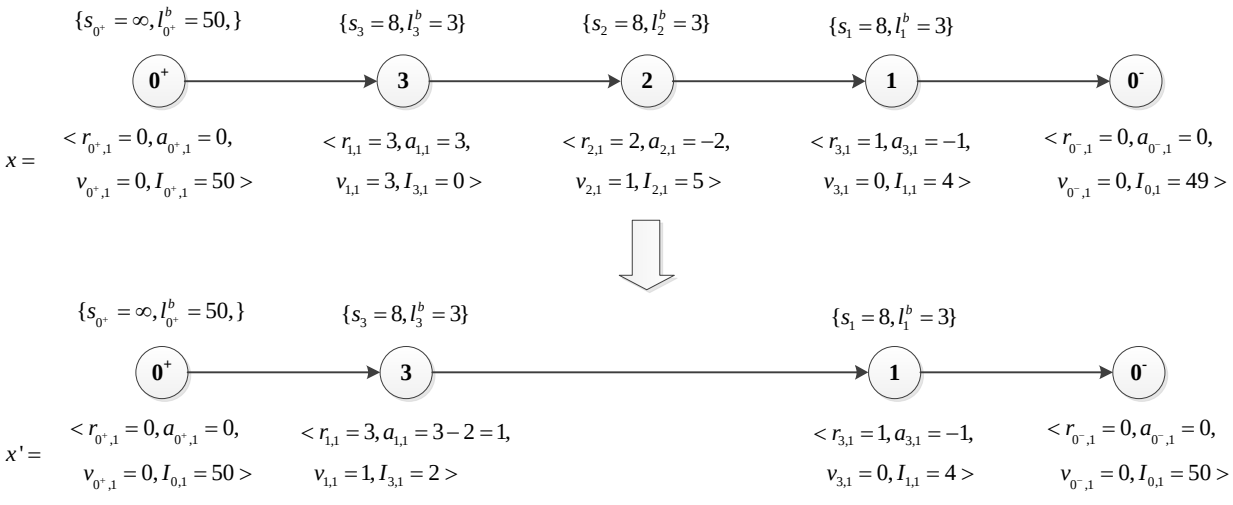

Figure 4.7: Removal neighborhood function, remove drop off station and adjust predecessor pickup station

Example 8: Suppose one solution includes one trip visiting 3 stations, $s_1 = s_2 = s_2 = 8$, $l_1^b = l_2^b = l_2^b = 3$. The vehicle *c*=10.. This feasible solution *x* includes routing sequence $r =< 0^+, 3, 2, 1, 0^- >$ and loading / unloading plan $a =< +4, -1, -2, -1, 0 >$. Station 2 is removed from this solution. The new generated solution *x*' is shown in Figure 4.8.

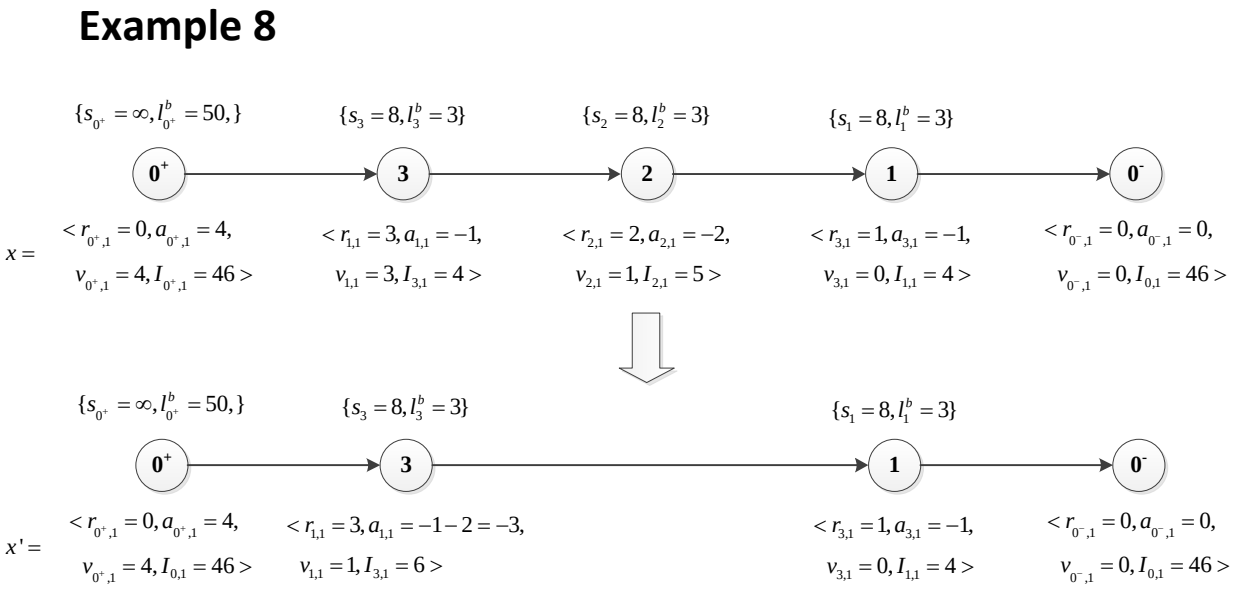

Figure 4.8: Removal neighborhood function, remove drop off station and adjust predecessor drop off station

3. Adjust the amount at the succeeding station - Adjust $a_{p+1,k}$:

If the successor station has enough inventory to cover the number removed (i.e. $a_{p,k} + a_{p+1,k} \le I_{p+1,k-1}$), station $r_{p+1,k}$ can pick up fewer bicycles to cover the number

removed (i.e. $a'_{p+1,k} = a_{p+1,k} + a_{p,k}$). Examples 9 and 10 illustrate the case when the successor is a pickup and drop off station.

Example 9: Suppose one solution includes one trip visiting 3 stations, $s_1 = s_2 = s_2 = 8$, $l_1^b = l_2^b = l_2^b = 3$. The vehicle *c*=10. This feasible solution x includes routing sequence $r =< 0^+, 3, 2, 1, 0^- >$ and loading / unloading plan $a =< 0, +3, -2, +1, -2 >$. Station 2 is removed from this solution. The new generated solution *x*' is shown in Figure 4.9.

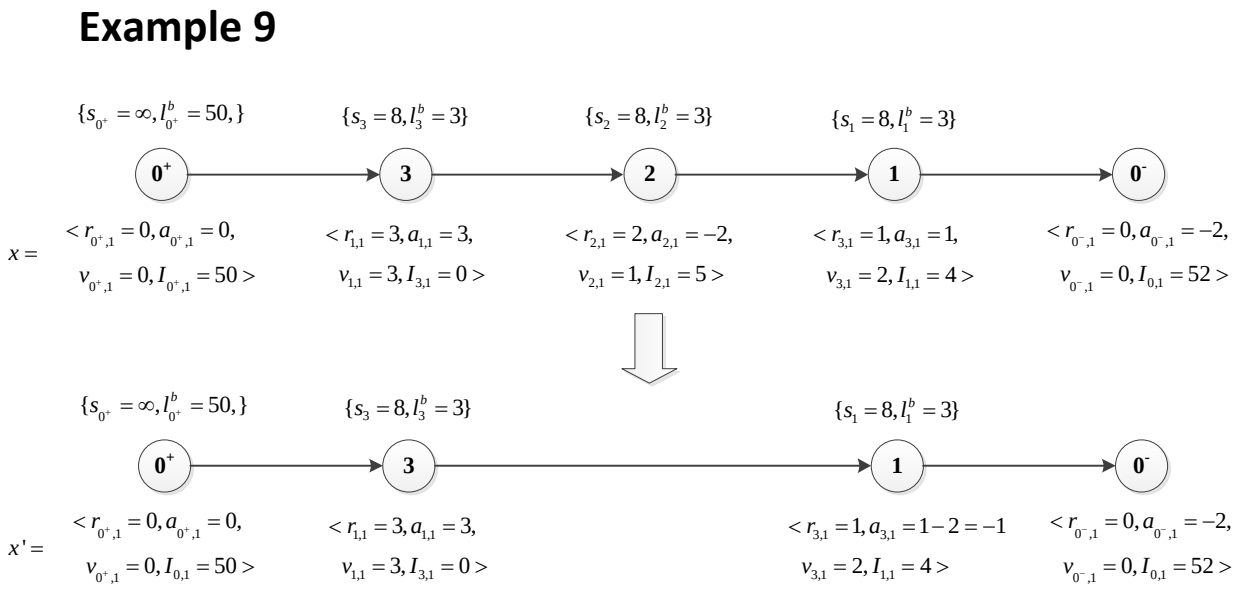

Figure 4.9: Removal neighborhood function, remove drop off station and adjust successor pickup station

Example 10: Suppose one solution includes one trip visiting 3 stations, $s_1 = s_2 = s_2 = 8$, $l_1^b = l_2^b = l_2^b = 3$. The vehicle c=10. This feasible solution x includes routing sequence $r =< 0^+, 3, 2, 1, 0^- >$ and loading / unloading plan $a =< 0, +3, -2, -1, 0 >$. Station 2 is removed from this solution. The new generated solution x' is shown in Figure 4.10.

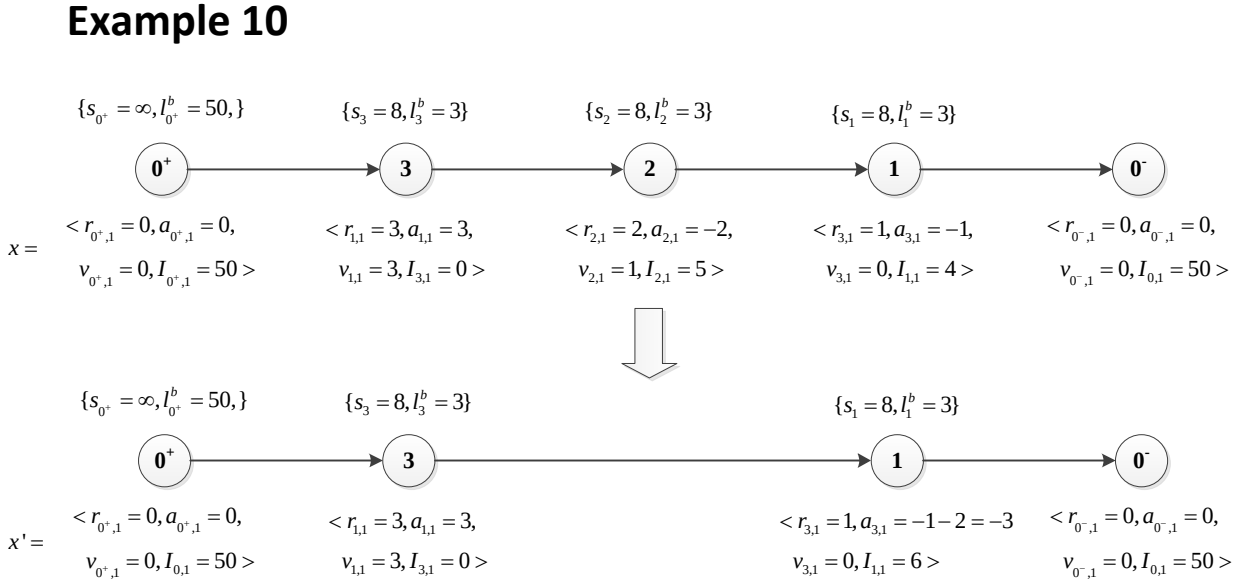

Figure 4.10: Removal neighborhood function, remove drop off station and adjust successor drop off station

Removed point is a depot

In this research, any point in the vehicle routing schedule except for the beginning and ending point can be removed, meaning that not only a station but also the depot within a routing sequence can be removed. According to the definition of the vehicle routing sequence, the depot points visited during the vehicle routing sequence are used as

the delimiter to separate trips in the routing. Once the depot point is removed, the adjacent two trips in the routing sequence will be merged into one large trip. Since each station can be visited at most once in each trip, it is necessary to check whether this change violates this constraint, reversing the remove depot action if needed. If the new trip does not violate the visit constraint, the depot visited is considered as a station, and the remove station method is used to generate 3 solutions. The only difference between a station and a depot visit is that the latter generates 2 visit points in the vehicle routing sequence ($r_{0^-,k}, r_{0^+,k+1}$). By merging these 2 depot points in assignment plan $a'_k = a_{0^-,k} + a_{0^+,k+1}$, the depot can be removed using the same method as for stations.

The Insert Neighborhood Function

The insert neighborhood function *I*(x) is the set of feasible neighbor solutions obtained by applying the insert moves to the current solution *x*=<*r*,*a*,*v*,*I*>. This move inserts a target station into the current vehicle routing sequence *r* and creates a related loading / unloading quantity for this inserted station. The new generated routing sequence *r*" and assignment plan *a*" become the inserted neighborhood *x*"=*R*(x). Similar to the remove neighborhood function, the insert move can also result in a solution that is not feasible. Using a strategy similar to the one in the remove neighborhood function, a maximum of 3 feasible solutions can be generated by adjusting the loading / unloading quantity for the depot and the predecessor and successor stations. To be consistent, it is

assumed that the insert point is $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ and that its position is before point $< r_{p+1,k}, a_{p+1,k}, v_{p+1,k}, I_{r_{p+1,k},k} >$ in solution *x*.

Insert point is a pickup station

If the inserted point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ is a pickup station, a maximum of 3 feasible solutions can be generated.

1. Adjust the amount at the ending depot visit - Adjust $a_{0^-,k}$:

First, the residual vehicle capacity $\eta_2 = \min_{t=p,\ldots,\rho_k}\{c - v_{t,k}\}$ is calculated. If $a_{p,k} \le \eta_2$, then the $a_{p,k}$ extra bicycles can be dropped at the depot at the end of the *k* th trip, $a'_{0^-,k} = a_{0^-,k} - a_{\rho_k k}$. Example 11 below shows this case.

Example 11: Suppose one solution includes one trip visiting 2 stations, $s_1 = s_2 = s_3 = 8$, $l_1^b = l_2^b = l_3^b = 3$. The vehicle c=10. This feasible solution x includes routing sequence $r =< 0^+, 3, 1, 0^- >$ and assignment plan $a =< 0, +3, -3, 0 >$. The point $< r_{2,1} = 2, a_{2,1} = +2 >$ is inserted into the current solution (i.e. insert station 2 into the second position of the first trip, picking up 2 bicycles). The new generated solution *x*' is shown in Figure 4.11.

Example 11

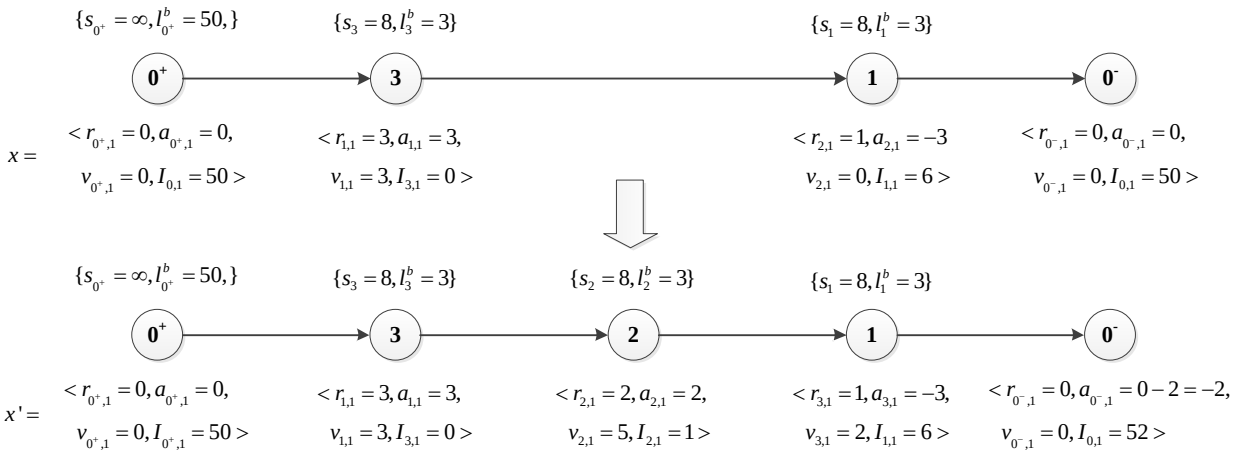

Figure 4.11: Insert neighborhood function, insert pickup station and adjust depot end point

2. Adjust the amount at the previous station - Adjust $a_{p-1,k}$:

If the predecessor station $r_{p-1,k}$ has enough bicycles (i.e. $I_{r_{p-1,k},k-1} - a_{p-1,k} + a_{p,k} \le s_{r_{p-1,k}}$), then station $r_{p-1,k}$ can drop off more / pick up fewer to cover the inserted number (i.e. $a'_{p-1,k} = a_{p-1,k} - a_{p,k}$). Examples 12 and 13 illustrate the case when the predecessor is a pickup and drop off station.

Example 12: Suppose one solution includes one trip visiting 2 stations, $s_1 = s_2 = s_3 = 8$, $l^b_1 = l^b_2 = l^b_3 = 3$. The vehicle c=10. This feasible solution x incudes routing sequence $r =< 0^+, 3, 1, 0^- >$ and assignment plan $a =< 0, +3, -3, 0 >$. The point $< r_{2,1} = 2, a_{2,1} = +2 >$

is inserted into the current solution (i.e. insert station 2 into the second position of the first trip, picking up 2 bicycles). The new generated solution *x*' is shown in Figure 4.12.

Example 12

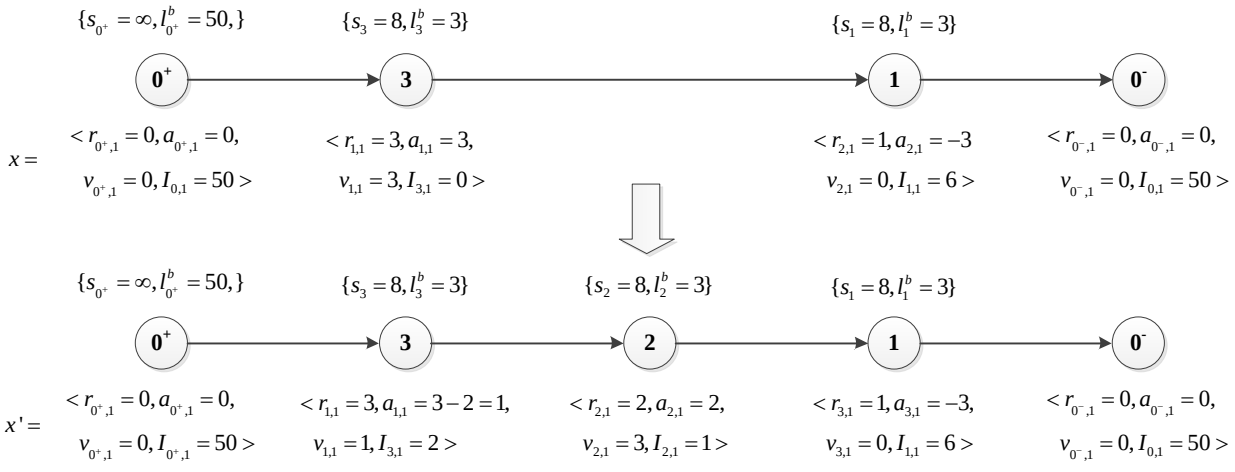

Figure 4.12: Insert neighborhood function, insert pickup station and adjust predecessor pickup station

Example 13: Suppose one solution includes one trip visiting 2 stations, $s_1 = s_2 = s_3 = 8$, $l^b_1 = l^b_2 = l^b_3 = 3$. The vehicle c=10. This feasible solution x includes routing sequence $r =< 0^+, 3, 1, 0^- >$ and assignment plan $a =< 4, -1, -3, 0 >$. The point $< r_{2,1} = 2, a_{2,1} = +2 >$ is inserted into the current solution (i.e. insert station 2 into the second position of the first trip, picking up 2 bicycles). The new generated solution *x*' is shown in Figure 4.13.

Example 13

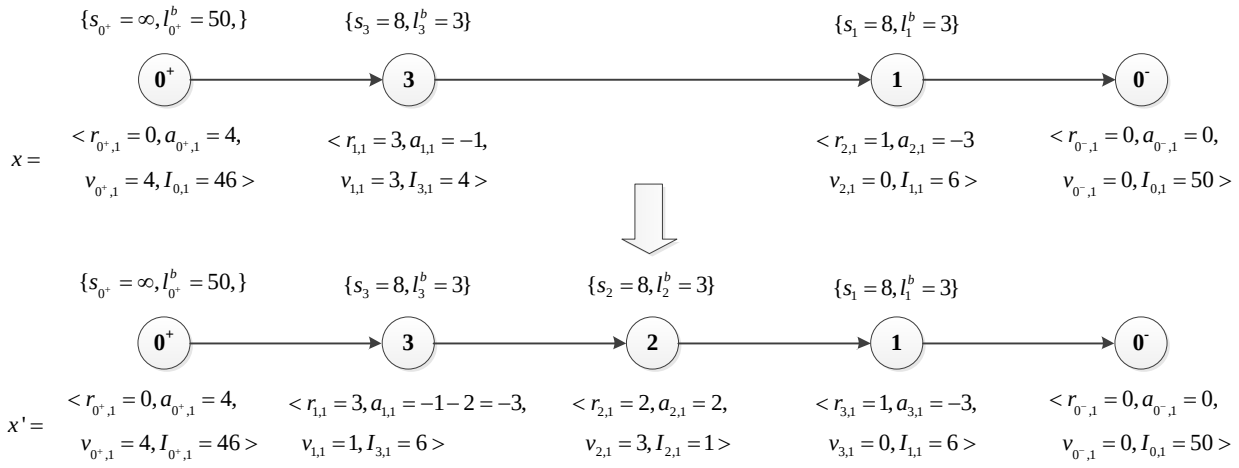

Figure 4.13: Insert neighborhood function, insert pickup station and adjust predecessor drop off station

3. Adjust the amount at the succeeding station - Adjust $a_{p+1,k}$:

If the successor station has enough empty lockers to cover the number of bicycles inserted (i.e. $I_{r_{p+1,k},k-1} - a_{p+1,k} + a_{p,k} \le s_{r_{p+1,k}}$), then station $r_{p+1,k}$ can drop off more to cover the inserted quantity (i.e. $a'_{p+1,k} = a_{p+1,k} - a_{p,k}$). Examples 14 and 15 show the case when the successor is a pickup and drop off station.

Example 14: Suppose one solution includes one trip visiting 2 stations, $s_1 = s_2 = s_3 = 8$, $l^b_1 = l^b_2 = l^b_3 = 3$. The vehicle c=10. This feasible solution x includes routing sequence $r =< 0^+, 3, 1, 0^- >$ and assignment plan $a =< 1, -1, 1, -1 >$. The point $< r_{2,1} = 2, a_{2,1} = +2 >$ is

inserted into the current solution (i.e. insert station 2 into the second position of the first trip, picking up 2 bicycles). The new generated solution x' is shown in Figure 4.14.

Example 14

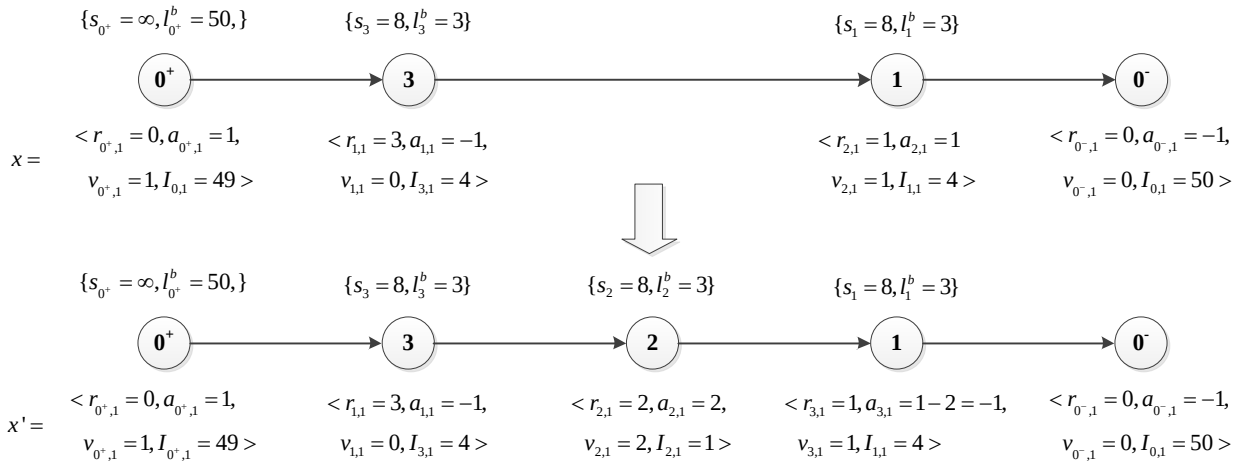

Figure 4.14: Insert neighborhood function, insert pickup station and adjust successor pickup station

Example 15: Suppose one solution includes one trip visiting 2 stations, $s_1 = s_2 = s_3 = 8$, $l^b_1 = l^b_2 = l^b_3 = 3$. The vehicle c=10. This feasible solution x includes routing sequence $r =< 0^+, 3, 1, 0^- >$ and assignment plan $a =< 0, +1, -1, 0 >$. The point $< r_{2,1} = 2, a_{2,1} = +2 >$ is inserted into the current solution (i.e. insert station 2 into the second position of the first trip, picking up 2 bicycles). The new generated solution x' is shown in Figure 4.15.

Example 15

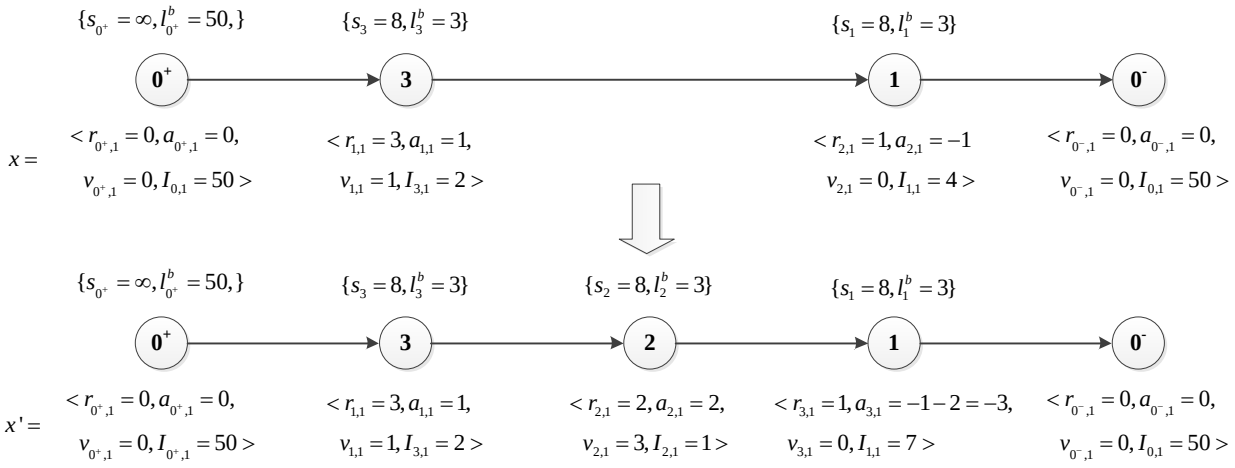

Figure 4.15: Insert neighborhood function, insert pickup station and adjust successor drop off station

Insert point is a drop off station

If the inserted point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ is a drop off station, a maximum of 3 feasible solutions can also be generated.

1. Adjust the amount at the starting depot - Adjust $a_{0^+,k}$:

First, the residual vehicle capacity is calculated $\eta_1 = \min_{t=0^+,1,\ldots,p-1}\{c - v_{t,k}\}$. If $a_{p,k} \le \eta_1$, then the $a_{p,k}$ bicycle can be picked at the depot at the beginning of *k* th *trip*, $a'_{0^+,k} = a_{0^+,k} - a_{p,k}$. Example 16 shows this situation.

Example 16: Suppose one solution includes one trip visiting 2 stations, $s_1 = s_2 = s_3 = 8$, $l_1^b = l_2^b = l_3^b = 3$. The vehicle c=10. This feasible solution x includes routing sequence $r =< 0^+, 3, 1, 0^- >$ and assignment plan $a =< 0, +1, -1, 0 >$. The point $< r_{2,1} = 2, a_{2,1} = -2 >$ is inserted into the current solution (i.e. insert station 2 into the second position of the first trip, picking up 2 bicycles). The new generated solution *x*' is shown in Figure 4.16.

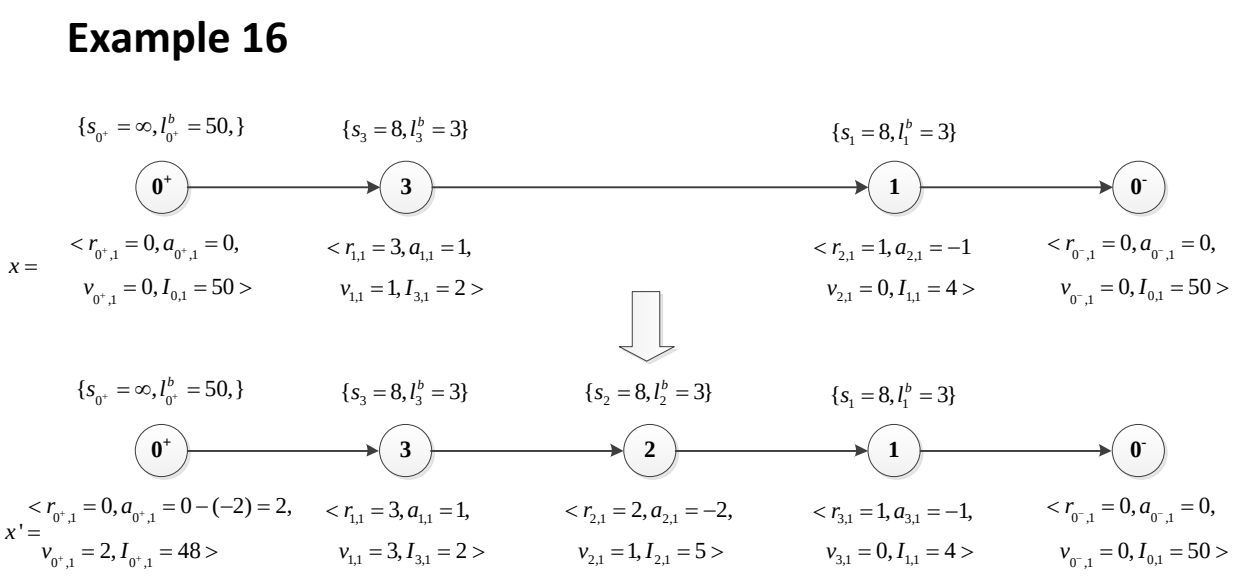

Figure 4.16: Insert neighborhood function, insert drop off station and adjust depot start point

2. Adjust the amount at the previous station - Adjust $a_{p-1,k}$:

If the predecessor station $r_{p-1,k}$ has enough bicycles $r_{p-1,k}$ (i.e. $a_{p-1,k} - a_{p,k} \le I_{r_{p-1,k},k-1}$), then station $r_{p-1,k}$ can pick up enough to cover the inserted quantity (i.e. $a'_{p-1,k} = a_{p-1,k} - a_{p,k}$). Examples 17 and 18 show the case when the predecessor is a pickup and drop off station.

Example 17: Suppose one solution includes one trip visiting 2 stations, $s_1 = s_2 = s_3 = 8$, $l_1^b = l_2^b = l_3^b = 3$. The vehicle *c*=10. This feasible solution x includes routing sequence $r =< 0^+, 3, 1, 0^- >$ and assignment plan $a =< 0, +1, -1, 0 >$. The point $< r_{2,1} = 2, a_{2,1} = -2 >$ is inserted into the current solution (i.e. insert station 2 into the second position of the first trip, picking up 2 bicycles). The new generated solution *x*' is shown in Figure 4.17.

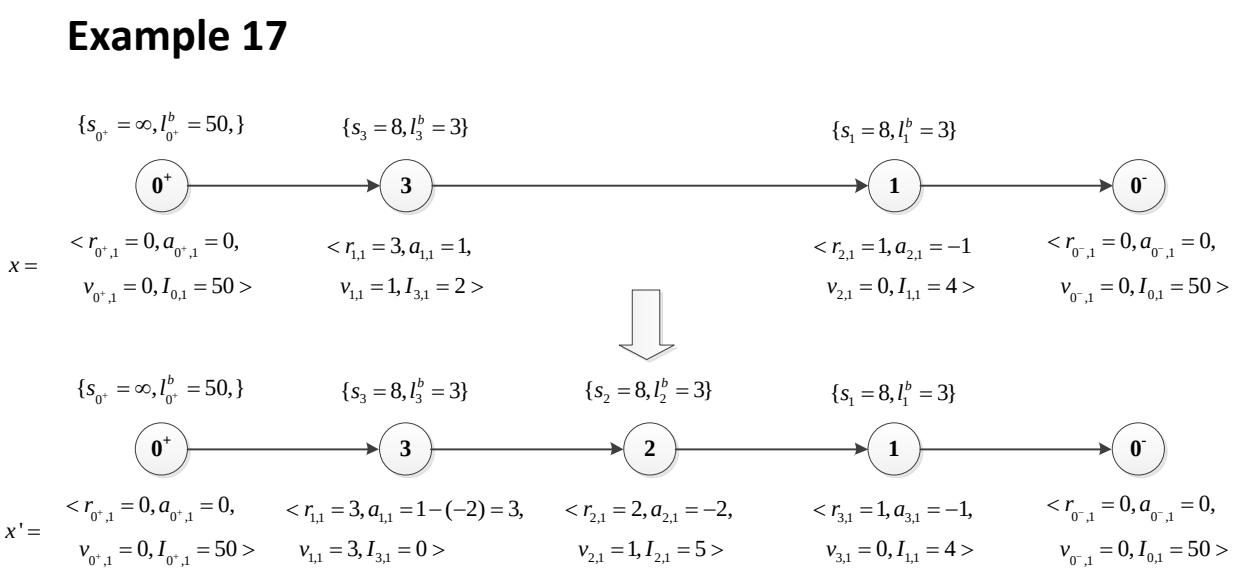

Figure 4.17: Insert neighborhood function, insert drop off station and adjust predecessor pickup station

Example 18: Suppose one solution includes one trip visiting 2 stations, $s_1 = s_2 = s_3 = 8$, $l_1^b = l_2^b = l_3^b = 3$. The vehicle *c*=10. This feasible solution *x* includes routing sequence $r =< 0^+, 3, 1, 0^- >$ and assignment plan $a =< 2, -1, -1, 0 >$. The point $< r_{2,1} = 2, a_{2,1} = -2 >$

is inserted into the current solution (i.e. insert station 2 into the second position of the 1st trip, picking up 2 bicycles). The new generated solution *x*' is shown in Figure 4.18.

Example 18

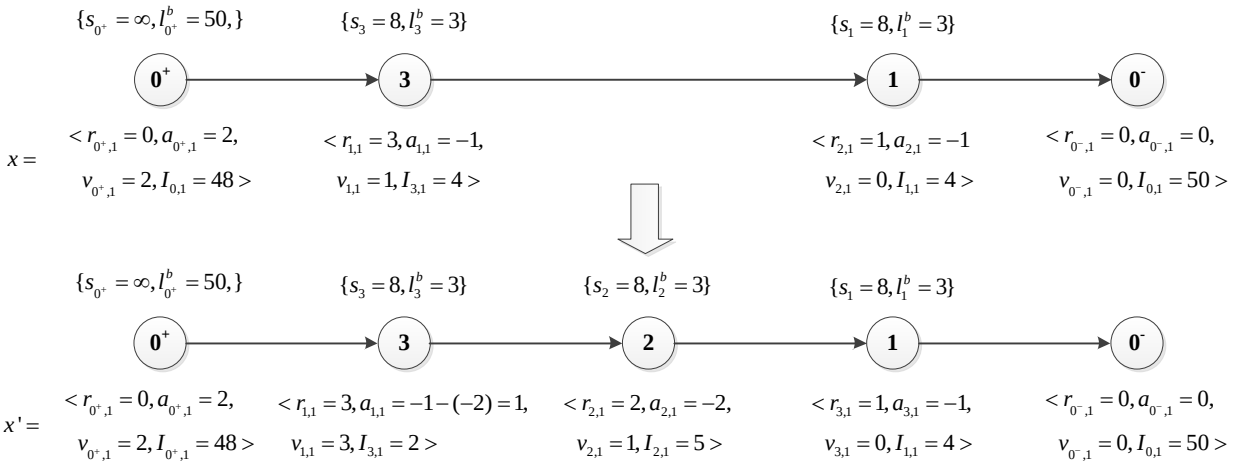

Figure 4.18: Insert neighborhood function, insert drop off station and adjust predecessor drop off station

3. Adjust the amount at the successor station - Adjust $a_{p+1,k}$:

If the successor station has enough inventory capacity to cover the number removed (i.e. $a_{p+1,k} - a_{p,k} \le I_{r_{p+1,k},k-1}$), then station $r_{p+1,k}$ can pick up more bicycles to cover the inserted number (i.e. $a'_{p+1,k} = a_{p+1,k} - a_{p,k}$). Examples 19 and 20 show the case when the successor is a pickup and drop off station.

Example 19: Suppose one solution includes one trip visiting 2 stations, $s_1 = s_2 = s_3 = 8$, $l^b_1 = l^b_2 = l^b_3 = 3$. The vehicle *c*=10. This feasible solution *x* includes the routing sequence

$r =< 0^+, 3, 1, 0^- >$ and the assignment plan $a =< +3, -1, +1, -3 >$. The point $< r_{2,1} = 2, a_{2,1} = -2 >$ is inserted into the current solution (i.e. insert station 2 into the second position of the first trip, picking up 2 bicycles). The new generated solution *x*' is shown in Figure 4.19.

Example 19

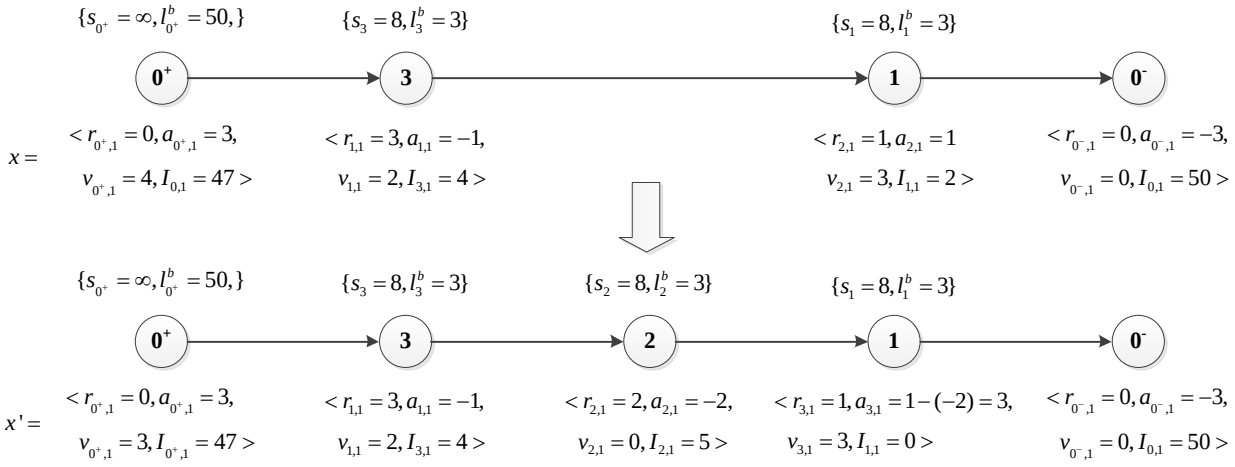

Figure 4.19: Insert neighborhood function, insert drop off station and adjust successor pickup station

Example 20: Suppose one solution includes one trip visiting 2 stations, $s_1 = s_2 = s_3 = 8$, $l^b_1 = l^b_2 = l^b_3 = 3$. The vehicle *c*=10. This feasible solution *x* includes routing sequence $r =< 0^+, 3, 1, 0^- >$ and assignment plan $a =< +3, -1, -1, -1 >$. The point $< r_{2,1} = 2, a_{2,1} = -2 >$ is inserted into the current solution (i.e. insert station 2 into the

second position of the first trip, picking up 2 bicycles). The new generated solution *x'* is shown in Figure 4.20.

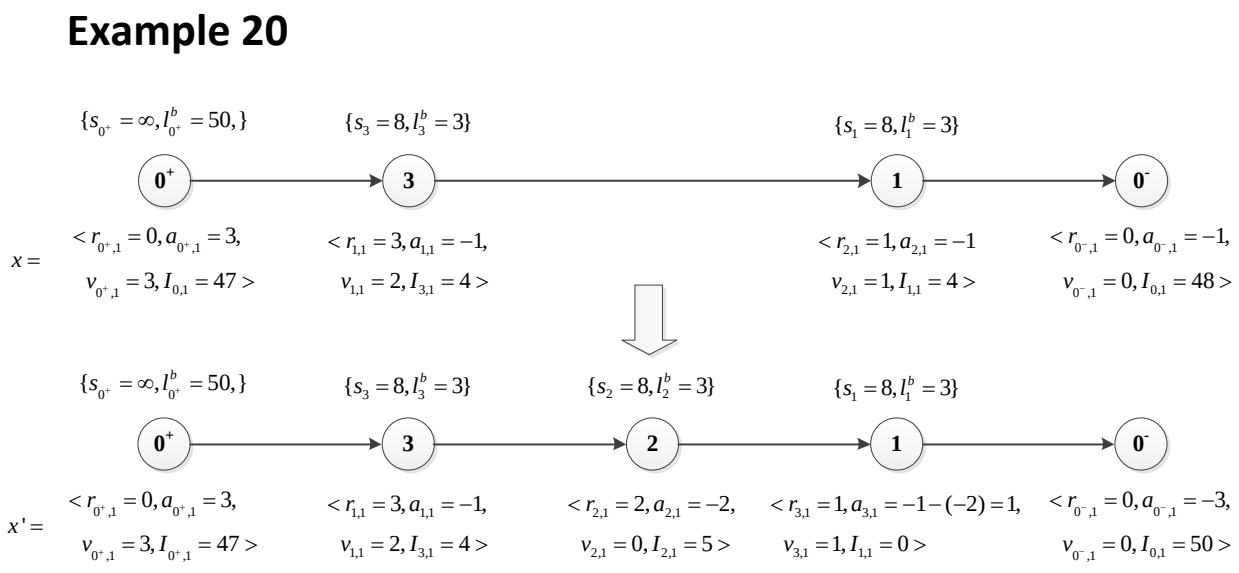

Figure 4.20: Insert neighborhood function, insert drop off station and adjust successor drop off station

Insert point is a depot

As mentioned earlier, visiting the depot in the vehicle routing sequence is a delimiter specifying only one trip, meaning inserting the depot into the current vehicle routing sequence divides one trip into two separate trips. Since it is known that the depot has enough bicycle and locker capacity, this insert action always generates a feasible solution if the new solution does not violate the time horizon constraint. Furthermore, because it is assumed that all bicycles are left at the depot end point and the required bicycles are picked up at the depot start point, it is not necessary to adjust the assignment

plan of other visit points to generate a new feasible solution. As a result, only 1 possible solution is generated when a depot is inserted in the vehicle routing sequence.

The neighborhood functions for complex station

Similar to the neighborhood function for a simple station, the neighborhood function for complex stations includes 2 basic functions: (1) Remove complex station *CR*(x), and (2) Insert complex stations *CI*(x). As opposed to the simple station, the complex station is visited multiple times in the solution. If the remove / insert visit point is not the last visit to the complex station in the vehicle routing, this action will affect the feasibility of its following *trips*, consequently also affecting the solution's feasibility.

Proposition 4.1:

Suppose station *i* has been visited a total of *n* times in solution *x*. These *n* visiting points in the solution are $< r_{t_l,k_l}, a_{t_l,k_l}, v_{t_l,k_l}, I_{i,k_l} >$ where $r_{t_l,k_l} = i$ and $l = 1,2,...,n$. To obtain the sequence of these points, we assume $k_j < k_{j+1}$ where $j = 1,2,...,n-1$ so that $< r_{t_l,k_l}, a_{t_l,k_l}, v_{t_l,k_l}, I_{i,k_l} >$ is the l th time to visit station *i*. If only I_{i,k_l} is changed, the inventory at station *i* for the k_l th *trip* becomes $I'_{i,k_l} = I_{i,k_l} + \tau_{i,k_l}$ while the rest of the routing sequence and assignment plans remain unchanged for solution *x*, where τ_{i,k_l} is the additional increased / decreased inventory for station *i*. Only the inventory level at station *i* for current and later trips is affected, nothing else. For all *trips* greater than or

equal to the k_l th *trip*, the initial inventory level at station *i* increases τ_{i,k_l} units. Based on the station inventory constraint, the inventory level should between 0 and s_i, for each trip, $0 \le I'_{i,q} = I_{i,q} + \tau_{i,k_l} \le s_i$ with $\tau_{i,k_l} \in [\alpha_1, \alpha_2]$ where $\alpha_1 = \max_{q=k_l,\ldots,k_n}\{-I_{i,q}\}$, $\alpha_2 = \min_{q=k_l,\ldots,k_n}\{s_i - I_{i,q}\}$,

Remove a multiple visited station

The previous section provided 3 possible ways to generate feasible solutions when removing a station from the vehicle routing sequence: (1) adjust the depot assignment plan, (2) adjust the predecessor assignment plan, and (3) adjust the successor assignment plan. The combination of the removed station and predecessor / successor station with pickup / drop off actions resulted in 10 possible ways to manage the assignment plan in the trip affected to obtain a feasible solution.

As opposed to the simple station remove function, removing a complex station not only affects the current *trip* but also later ones scheduled for the removed complex station. The following discussion separates the complex station remove function into two cases: (1) only the removed point is a complex station, and (2) the removed station and adjusted adjacent station are both complex stations.

(1) Only the removed point is complex station

Suppose the removed visit point is $< r_{t_l,k_l}, a_{t_l,k_l}, v_{t_l,k_l}, I_{i,k_l} >$, where $r_{t_l,k_l} = i$ which has been defined earlier in this chapter. For the k_l th *trip*, the simple station remove function is used to obtain the feasible trip route. Originally, the initial inventory at station *i* for k_{l+1} th *trip* was I_{i,k_l}. However, $< r_{t_l,k_l}, a_{t_l,k_l}, v_{t_l,k_l}, I_{i,k_l} >$ has been removed, meaning the initial inventory at station *i* for k_{l+1} th *trip* is now $I_{i,k_{l-1}} = I_{i,k_l} + a_{t_l,k_l}$. Based on Proposition 4.1, the feasible solution must satisfy the following constraints: $a_{t_l,k_l} \in [\alpha_1, \alpha_2]$ where $\alpha_1 = \max_{q=k_l,\ldots,k_n}\{-I_{i,q}\}$ and $\alpha_2 = \min_{q=k_l,\ldots,k_n}\{s_i - I_{i,q}\}$ to ensure the remaining trips are feasible. Satisfying both constraints results in a new feasibility constraint for the new solution generated.

(2) Removed point and adjusted adjacent point (predecessor / successor) both are complex station

Suppose the removed visit point is $< r_{t_l,k_l}, a_{t_l,k_l}, v_{t_l,k_l}, I_{i,k_l} >$ (station *i*), and the predecessor point is $< r_{t_l-1,k_l}, a_{t_l-1,k_l}, v_{t_l-1,k_l}, I_{j,k_l} >$ (station *j*), both being complex stations. For the k_l th *trip*, the simple station remove function is used to obtain the feasible trip route for the k_l th *trip*. The assignment plan for station *i* has been merged to station *j*, meaning that for the k_{l+1} th *trip*, the initial inventory for station *i* is increased a_{t_l,k_l} units and the initial inventory for station *j* is increased $-a_{t_l,k_l}$ units. Based on Proposition 4.1, for the remaining trip to be feasible, the following constraints need to be satisfied:

$a_{t_l,k_l} \in [\alpha_1, \alpha_2]$ where $\alpha_1 = \max_{q=k_{l+1},\ldots,k_n}\{-I_{i,q}\}$, $\alpha_2 = \min_{q=k_{l+1},\ldots,k_n}\{s_i - I_{i,q}\}$

$-a_{t_l,k_l} \in [\beta_1, \beta_2]$ where $\beta_1 = \max_{q=k_{l+1},\ldots,k_n}\{-I_{j,q}\}$, $\beta_2 = \min_{q=k_{l+1},\ldots,k_n}\{s_j - I_{j,q}\}$

Satisfying all constraints results in a new feasibility constraint for the new solution generated.

The improvement method

Once the VNS method has been applied, it is necessary to use improvement function *P*(*x*) to determine whether the current assignment plan can be improved.

Improvement neighborhood function

1. Improvement function by changing one station's assignment plan

This improvement function attempts to improve the penalty cost by changing only one station's assignment plan without changing the vehicle routing schedule. As constrained by the flow balance on the vehicle, the total number of bicycle loaded into the vehicle should be equal to the total number of bicycle unloaded from the vehicle for each *trip* and the entire repositioning process. When the assignment plan for one point is changed, at least one other point's assignment plan is also changed. Since this method changes only one station's assignment plan, the other changed assignment plan points are the depot. Suppose station *i* can be improved ($I_i^a \neq t_i$), and point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$

visits station *i* (i.e. $r_{p,k} = i$). This point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ can be split into 2 points: $< r_{p,k}, a_{p,k} + \Delta, v_{p,k} + \Delta, I_{r_{p,k},k} - \Delta >$ and $< r_{p,k}, -\Delta, v_{p,k}, I_{r_{p,k},k} >$, where $\Delta = 0, ..., I_i^a - t_i$. By using the remove function to remove point $< r_{p,k}, -\Delta, v_{p,k}, I_{r_{p,k},k} >$, solution *x*' which changes only the assignment plan for station *i* is obtained. Then the best penalty cost can be determined using these feasible solutions to obtain an improved solution for the current solution *x*.

2. Improvement function by changing the assignment plan of two stations

This improvement function attempts to improve the current solution by changing the assignment plan for two stations in the same *trip* to reduce the penalty cost while not changing the vehicle routing schedule. Because of the different sizes of, locations of and customer demands for each station, the penalty cost function for each is different as is the target inventory level. For each station, any change in its target inventory level causes a penalty cost.

Suppose improvement in the *k* th trip is considered using the two visit points $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ and $< r_{q,k}, a_{q,k}, v_{q,k}, I_{r_{q,k},k} >$, where $p < q$. Assume $r_{p,k} = i$ and $r_{q,k} = j$; both station *i* and *j* can be either a single station or complex stations. There are two ways to make the adjustment: (1) a forward adjustment loading $\Delta \geq 0$ more bicycle units at station *i*, and loading Δ fewer bicycle units at station *j*. (2) a backward adjustment loading $\Delta \geq 0$ fewer bicycle units at the station *i*, and loading Δ more bicycle units at station *j*.

(1) The forward adjustment:

In this case, the new solution will pick up more (drop off fewer) bicycles at the first visit point and pick up fewer (drop off more) bicycles at the second visit point. These 2 new adjusted visit points are $< i, a_{p,k} + \Delta, v_{p,k} + \Delta, I_{i,k} - \Delta >$ and $< j, a_{q,k} - \Delta, v_{q,k}, I_{j,k} + \Delta >$. All visit points between these two are modified to $< r_{s,k}, a_{s,k}, v_{s,k} + \Delta, I_{r_{s,k},k} >$ where $p < s < q$. To have this adjustment lead to a feasible solution requires the following constraints: $0 \le I_{i,k} - \Delta \le s_i$, $0 \le I_{j,k} + \Delta \le s_j$, $0 \le v_{p,k} + \Delta \le c$, and $0 \le v_{s,k} + \Delta \le c$ where $p < s < q$. Merging these constraints results in $\Delta \in \gamma_1$ where $\gamma_1 = [0, \min\{c - v_{p,k}, ..., c - v_{q-1,k}, s_j - I_{j,k}, I_{i,k}\}]$. If station *i* or station *j* is the complex station, the solution includes a total of ρ trips. For each station *i* and *j*, the change in the inventory levels in the *k* th *trip* will affect all of the initial inventory levels for later trips, meaning that for all visit points after the *k* th *trip*, the initial inventory level for station *i* will be decreased Δ. To maintain feasibility, all constraints $0 \le I_{i,t} - \Delta \le s_i$, where $t \in [k, \rho]$ need to be satisfied. Thus, $\Delta \in \gamma_2$, where $\gamma_2 = [0, \min\{I_{i,k}, ..., I_{i,\rho}\}]$. Applying the same method to station *j* leads to a similar group of constraints for maintaining the feasibility, resulting in $\Delta \in \gamma_3$, where $\gamma_3 = [0, \min\{s_j - I_{j,k}, ..., s_j - I_{i,\rho}\}$. Merging the limited ranges results in $\Delta \in \gamma$, where $\gamma = [0, \min\{c - v_{p,k}, ..., c - v_{q-1,k}, s_j - I_{j,k}, ..., s_j - I_{j,\rho}, I_{i,k}, ..., I_{i,\rho}\}]$. Finding the new solution improves the current solution by picking up $\Delta^* = \arg\min_{\Delta \in \gamma} f_i(I_i^a - \Delta) + f_j(I_j^a + \Delta)$ more bicycles at station *i* and dropping off Δ^* more units at station *j*.

(2) The backward adjustment:

In this case, the new solution will pick up fewer (drop off more) bicycles at the first visit point and pick up more (drop off fewer) bicycles at the second visit point. These 2 new adjusted visit points are $< i, a_{p,k} - \Delta, v_{p,k} - \Delta, I_{i,k} + \Delta >$ and $< j, a_{q,k} + \Delta, v_{q,k}, I_{j,k} - \Delta >$. All visit points between these two are modified to $< r_{s,k}, a_{s,k}, v_{s,k} - \Delta, I_{r_{s,k},k} >$ where $p < s < q$. To make this adjustment lead to a feasible solution requires satisfying the following constraints: $0 \le I_{i,k} + \Delta \le s_i$, $0 \le I_{j,k} - \Delta \le s_j$, $0 \le v_{p,k} - \Delta \le c$, and $0 \le v_{s,k} - \Delta \le c$ where $p < s < q$. Merging these constraints results in $\Delta \in \varphi_1$ where $\varphi_1 = [0, \min\{v_{p,k}, ..., v_{q-1,k}, I_{j,k}, s_i - I_{i,k}\}]$. If station i or station j are a complex station, the solution includes a total of ρ trips. For station *i* and *j*, the change in the inventory level in the *k* th *trip* will affect the initial inventory levels for all later trips. For all visit points after the *k* th *trip*, the initial inventory level for station *i* will be decreased Δ. To maintain the feasibility, all constraints $0 \le I_{i,t} + \Delta \le s_i$, where $t \in [k, \rho]$ need to be met, meaning $\Delta \in \varphi_2$, where $\varphi_2 = [0, \min\{s_i - I_{i,k}, ..., s_i - I_{i,\rho}\}]$. Applying the same method for station *j* results in a similar group of constraints for maintaining the feasibility, $\Delta \in \gamma_3$, where $\varphi_3 = [0, \min\{I_{j,k}, ..., I_{i,\rho}\}$. Merging the limited range results in $\Delta \in \varphi$, where $\varphi = [0, \min\{v_{p,k}, ..., v_{q-1,k}, I_{j,k}, ..., I_{j,\rho}, s_i - I_{i,k}, ..., s_i - I_{i,\rho}\}]$. Finding the new solution improves the objective by dropping off $\Delta^* = \arg\min_{\Delta \in \varphi} f_i(I_i^a - \Delta) + f_j(I_j^a + \Delta)$ more bicycles at station *i* and picking up Δ^* more units at station *j*.

Numeric Experiment

This section use the results of numeric experiments to test the performance of the heuristic algorithm (H2) proposed in this research to solve the SBRP problem with multiple trips. Unlike the heuristic algorithm (H1) proposed in Chapter 3 for solving the SBRP problem with a single trip, the H2 algorithm uses a 1-step strategy to solve the problem. While the H1 algorithm uses a routing first, loading plan second strategy, the H2 algorithm creates / modifies the routing schedule and loading plan at the same time. Making two decisions (vehicle routing schedule and loading plan) simultaneously makes the H2 algorithm more complex than the H1. This research runs the test on the data instances found in Ho and Szeto's (2014) research, the same datasets used to test the H1 algorithm. This section then compares the solving time and the objective results of the two algorithms.

All tests were conducted on a Dell notebook with an Intel Core i5-2520M CPU @ 2.5 GHz with 2GB RAM. The heuristic algorithm was coded using C++ in Microsoft Visual Studio 2013.

Testing Scenarios

All the testing scenarios used in this analysis come from Ho and Szeto's (2014) research. In total 13 instances involving different numbers of stations were tested. The

first instance includes 100 stations and the second instances include 125 stations. Each instance includes 25 more stations than the previous one, and the last instance includes 400 stations. In addition, 2 levels of time horizon, 9000 and 18000 time units were used, and the vehicle capacity was also tested at 2 levels, 10 and 20. Table 4.1 lists the parameters for all testing scenarios. The combination of the instance, time horizon and vehicle capacity resulted in a total of 13*2*2=52 testing scenarios classified in 4 group sets.

Table 4.1: Parameter Table for instances in Ho and Szeto (2014)'s research

Parameters	Values
Station Number	U(100,400)
Time Horizon	{9000,18000}
Vehicle capacity	{10,20}

Testing Results

Since both H1 and H2 algorithm includes the randomness, it is possible to get the different results with the same testing scenario and input parameters. To reduce the randomness in the comparison, we run m=30 iterations for each algorithm with each testing scenario. We summarized two measure criteria from these m=30 iterations for each test scenario: Objective Value and Solving Time. These two values are used to represent the performance of the algorithm in the testing scenario.

In order to consistent with previous analysis, we still just use the penalty cost to be the objective. All the resting results are shown in the following table 4.2~4.5.

Table 4.2: Test Cases with Time horizon = 9000, and vehicle capacity = 10

$\|N\|$	H1 algorithm					H2 algorithm					
	Min	Avg	Max	Std	Time (s)	Min	Avg	Max	Std	Time (s)	GAP
100	749.6	752.9	754.6	1.32	153	650.3	656.8	657.2	1.96	2.08	12.8%
125	1004.2	1004.4	1007.3	0.93	142	886.5	889.5	889.6	0.85	2.14	11.4%
150	1222.9	1223.4	1229.7	1.94	169	1079.8	1084.7	1092.2	3.50	2.86	11.3%
175	1372.6	1375.0	1378.9	1.74	184	1177.3	1192.4	1193.4	4.34	3.47	13.3%
200	1615.1	1616.7	1617.6	0.71	150	1217.9	1229.8	1242.4	7.72	3.70	23.9%
225	1874.1	1885.0	1890.2	4.97	192	1631.4	1635.2	1635.6	1.23	6.86	13.2%
250	2102.8	2112.8	2119.1	4.15	160	1896.7	1897.0	1898.8	0.66	5.18	10.2%
275	2236.8	2239.7	2243.4	1.89	195	1940.2	1946.4	1949.4	2.85	5.28	13.1%
300	2497.6	2507.7	2508.5	2.93	169	2149.2	2155.4	2163.3	4.09	6.77	14.0%
325	2740.3	2741.6	2742.8	0.95	202	2354.8	2361.3	2364.0	2.41	5.91	13.9%
350	2972.8	2973.9	2976.7	1.05	264	2673.7	2678.9	2683.1	2.79	6.46	9.9%
375	3118.9	3121.3	3126.7	1.89	258	2750.0	2752.6	2762.7	3.78	6.48	11.8%
400	3385.2	3385.8	3397.9	3.97	289	2823.5	2836.2	2849.8	6.94	7.41	16.2%

* The objective value does not include the transportation cost.

Table 4.3: Test Cases with Time horizon = 18000 and vehicle capacity = 10

$\|N\|$	H1 algorithm					H2 algorithm					
	Min	Avg	Max	Std	Time (s)	Min	Avg	Max	Std	Time (s)	GAP
100	680.9	666.6	684.1	0.63	331	587.4	589.6	602.4	4.85	2.85	11.6%
125	920.5	891.1	923.6	1.72	304	790.5	790.6	791.6	0.35	3.1	11.3%
150	1136.8	1116.4	1149.2	0.58	207	977.3	981.7	981.8	1.21	3.3	12.1%
175	1283.5	1245.7	1287.4	5.05	317	1117.2	1123.8	1124.5	1.83	2.77	9.8%
200	1511.9	1502.5	1524.3	1.33	285	1202.8	1203.0	1207.4	1.17	3.81	19.9%
225	1790.2	1732.4	1790.7	2.12	293	1535.6	1540.6	1541.9	2.12	4.07	11.1%
250	2014.3	1962.5	2020.6	2.48	306	1638.9	1641.5	1647.9	2.59	6.9	16.4%
275	2118.7	2151.0	2127.8	1.98	376	1848.5	1849.0	1859.6	3.09	7.53	14.0%
300	2403.4	2376.2	2408.7	4.72	325	1995.3	2010.3	2017.0	5.9	9.8	15.4%
325	2628.7	2616.6	2643.3	6.45	362	2234.1	2240.6	2246.3	3.8	10.42	14.4%
350	2864.1	2853.1	2868.3	0.2	343	2418.3	2434.2	2434.5	4.47	12.1	14.7%
375	2993.4	3028.5	3006.8	1.91	382	2576.2	2577.0	2583.0	1.72	12.73	14.9%
400	3262.3	3265.2	3270.8	4.02	421	2778.2	2789.3	2791.2	3.85	14.56	14.6%

* The objective value does not include the transportation cost.

Table 4.4: Test Cases with Time horizon = 9000 and vehicle capacity = 20

$\|N\|$	H1 algorithm					H2 algorithm					
	Min	Avg	Max	Std	Time (s)	Min	Avg	Max	Std	Time (s)	GAP
100	680.9	682.5	684.1	0.83	684	563.4	565.4	567.3	2.96	2.37	17.2%
125	920.5	923.5	923.6	0.95	613	735.2	744.4	753.6	2.25	2.58	19.4%
150	1136.8	1144.9	1149.2	3.35	723	884.9	885.1	886.3	2.2	3.47	22.7%
175	1283.5	1285.5	1287.4	1	789	1048.6	1050.7	1051.5	2.79	3.97	18.3%
200	1511.9	1515.1	1524.3	2.95	898	1135.0	1144.8	1146.2	2.32	5.09	24.4%
225	1790.2	1790.2	1790.7	0.11	921	1468.8	1474.2	1487.8	1.78	5.12	17.7%
250	2014.3	2019.1	2020.6	1.58	969	1622.3	1634.4	1636.0	3.11	5.28	19.0%
275	2118.7	2127.1	2127.8	2.8	1123	1767.9	1775.0	1775.0	1.53	6.92	16.6%
300	2403.4	2406.8	2408.7	1.39	1266	1927.8	1928.1	1930.8	1.24	7.16	19.9%
325	2628.7	2632.9	2643.3	5.1	1607	2157.4	2164.0	2169.1	1.31	9.04	17.8%
350	2864.1	2866.4	2868.3	1.22	1772	2334.5	2353.9	2361.9	2.11	9.37	17.9%
375	2993.4	3005.7	3006.8	4.41	1764	2478.4	2480.3	2480.4	4.04	9.6	17.5%
400	3262.3	3280.2	3270.8	2.5	2215	2605.4	2617.7	2622.8	4.48	10.96	20.2%

* The objective value does not include the transportation cost.

Table 4.5: Test Cases with Time horizon = 18000 and vehicle capacity = 20

$\|N\|$	H1 algorithm					H2 algorithm					
	Min	Avg	Max	Std	Time (s)	Min	Avg	Max	Std	Time (s)	GAP
100	578.7	580.9	584.6	1.64	2029	500.3	501.6	512.2	3.7	3.10	13.6%
125	790.0	790.3	795.3	1.57	1903	640.3	642.1	652.8	3.63	5.83	18.8%
150	1000.8	1001.7	1003.1	0.73	2362	822.8	826.1	841.7	5.74	3.93	17.5%
175	1115.4	1118.4	1119.3	1.17	2851	969.6	977.6	988.5	5.82	4.87	12.6%
200	1361.4	1368.2	1373.4	3.08	2938	1139.0	1139.7	1139.8	0.27	6.60	16.7%
225	1584.8	1585.3	1595.8	3.37	2186	1283.4	1284.1	1287.8	1.27	7.80	19.0%
250	1799.9	1801.9	1808.9	2.69	2293	1510.8	1517.4	1520.1	2.75	8.71	15.8%
275	1990.6	1991.3	1992.3	0.55	2153	1641.5	1641.6	1642.7	0.31	10.19	17.6%
300	2192.2	2207.1	2209.1	4.47	2694	1780.1	1788.4	1792.0	3.64	10.32	19.0%
325	2427.5	2431.7	2431.7	1.27	3270	1837.8	1861.6	1866.1	8.23	11.46	23.4%
350	2677.7	2678.8	2681.6	1.17	3565	2132.7	2134.4	2135.9	1.04	15.08	20.3%
375	2849.8	2855.0	2856.4	1.79	3503	2138.5	2152.2	2154.8	4.61	15.13	24.6%
400	3071.3	3074.3	3085.6	3.91	3009	2315.8	2323.5	2330.9	4.32	16.14	24.4%

* The objective value does not include the transportation cost.

The “|*N*|”, “Min”, “Avg”, “Max”, “Std” and “Time” columns for H1 algorithm are all copied from chapter 3 research which represent “the total station number”, “Minimal objective value of H1 algorithm with *m* iterations”, “Average objective value of H1 algorithm with *m* iterations”, “Maximal objective value of H1 algorithm with *m* iterations”, “Standard Deviation of m iterations objective value” and “Average solving time of H1 with *m* iterations”. Since the H2 algorithm also include the randomness, we use the same testing method to measure H2 as we did to H1 in chapter 3, i.e. run 30 duplications for each testing scenario to get the summary result. The column “Min”, “Avg”, “Max”, “Std”, “Time” column for H2 algorithm represent the minimal objective value, average objective value, maximal objective value, standard deviation of objective value, and average running time for 30 duplication running results, respectively. The “GAP” is the improvement gap between the “H1 algorithm” and “H2 algorithm”, determined using the formula GAP = (H1 – H2) / H1.

As shown in the tables above, the H2 algorithm provides a smaller objective value for the same testing scenario, with the improvement gap ranging from 9.8% to 24.6%, with an average value of 16.3%. All these observation shows that the H2 algorithm can provide better solution than H1 algorithm. At first, the Anderson-Darling test was used to check the normality of the raw data. The results shown in Figures B.1~B.4 indicate that the average objective values for H2 do not follow a normal distribution, meaning nonparametric statistical tests were required.

This analysis used the Freidman test, a nonparametric statistical tool similar to a two-way ANOVA, to explore these observations. The three algorithms served as the

treatment and the testing scenarios as the blocks. In general, the Freidman test ranks the average objective values from each algorithm for each testing scenario, with the algorithm with the lowest value being assigned rank 1, the second best rank 2, and so on until all are ranked. In the case of a tie, average ranks are assigned. For example, if 2 algorithms are tied for rank 1, they are both ranked 1.5 and next rank is 3. The hypotheses for Freidman test are listed below:

H_0: the median of the average objective values is equal for two algorithms.

H_1: medians of the average objective values for two algorithms are equal.

Appendix Tables B.1, B.2, B.3 and B.4 present the statistical results from the Freidman tests obtained using MINITAB. All tests results provide very small *p*-values (<0.001), meaning that there is sufficient evidence to reject the null hypothesis and conclude that not all medians of the average values of all algorithms are equal.

Combined with previous observation, we can say that the H2 algorithm can provide higher quality solution for the SBRP problem than the H1 algorithm. The reason for this improvement is the relaxation of the visit limitation for the SBRP problem. The H1 algorithm includes a full visit limitation, with each station being visited only once in the solution. The H2 algorithm, on the other hand, partially relaxed this visit limitation, allowing a station to be visited multiple times in different trips. This relaxation allowed the station with a large demand / inventory to be fulfilled if a large deviation from the target value causes a larger penalty cost. At the same time, multiple trips are used in this solution, meaning that the vehicle can visit a station multiple times. Because the depot is assumed to have infinite locker and bicycle inventory capacity, the penalty cost can be

reduced by visiting the depot to unload / pick up extra bicycles if the station is close to the depot. Thus, the change in the performance between the H1 algorithm and the H2 algorithm is not only caused by the improvement in the algorithm but also by the relaxing of the constraint in the research problem.

Furthermore, the H2 algorithm uses much less solving time than the H1 algorithm to obtain the results for the same instance. The H2 algorithm solving time is fairly stable, not changing when the time horizon and vehicle capacity is increased. In general, the H2 algorithm performed much better than the H1 algorithm in relation to solving time, providing feedback in a very short time. In relation to the objective value, the H2 algorithm also performed better, providing a better solution than the H1 algorithm. This improvement is not only caused by the new design of the heuristic algorithm but also by partially removed the station visit limitation. Allowing the same station to be visited multiple times within different trips increases the lower bound of the problem.

Conclusion and Future Work

This research proposed a VNS heuristic to solve the static bicycle repositioning problem using a single vehicle and multiple trips to complete the event. In addition, the visit limitation was partially relaxed by allowing the same station to be visited multiple times in different trips. The multiple trip assumption and visit limitation relaxation

improved the basic model, making it reflect the real-world more closely. Furthermore, rather than using the two-step routing first, loading assignment second strategy, a one-step approached was proposed, meaning the routing schedule and the loading assignment were constructed at the same time.

The experimental results using the instances from Ho and Szeto's (2014) research indicate the new heuristic algorithm H2 provides a good quality solution within a short solving time. In addition, it provides a better solution than the H1 algorithm, with an average improvement of 0.23%. This improvement is caused by the one-step structure of the H2 and the relaxation of the visit limitation constraint.

Future work will extend this research in several aspects: (1) Developing the heuristic to fully remove the visit limitation constraint and allowing the vehicle to use any schedule to complete the repositioning event. (2) Extending the current SBRP problem to include multiple vehicles scenarios.

CHAPTER FIVE

STATIC BICYCLE REPOSITIONING PROBLEM WITH A SINGLE VEHICLE, MULTIPLE TRIPS AND MULTIPLE VISITS

Previous chapters developed mathematical models for solving the static bicycle repositioning problem with a single vehicle. The solution for the model in Chapter 3 was limited to only one trip, and no station could be visited more than once. The model in Chapter 4 relaxed this constraint to allow the vehicle to use multiple trips in the solution and to allow the same station to be visited multiple times in different trips, while maintaining the constraint that each station could be visited at most once in one trip. In this chapter the visit limit constraint is completely relaxed to make the model realistic. In this research, the vehicle can use multiple trips in the solution and each station can be visited as many times as needed within the working time horizon. In other words, the vehicle can use any route without considering the visiting limitation, making this model equivalent to a realistic situation. A VNS-based algorithm was developed to solve the problem.

Introduction

Similar to the previous chapters, the issue addressed in this research remains focused on solving the static bicycle rebalance problem with a single vehicle. As explained earlier, one of the critical issues for maintaining a bicycle sharing system is to

ensure customers can rent / return bicycles to the station at their convenience. To address this objective, the bicycle sharing system needs to maintain the bicycle inventory level at each station at the target value on a daily basis. The static bicycle rebalance problem describes the model used to redistribute the bicycle inventory levels among these various stations with a single vehicle. In this research, we provide the method for addressing the static bicycle rebalance problem with a single vehicle using multiple trips and multiple visit.

Problem Descriptions and Terms

As in the other two studies, this one also is concerned with repositioning the bicycle inventory level among different stations to its target value with a single vehicle within the working time horizon. More specifically, the objective is to create a solution minimizing the total system cost. The operation cost for a repositioning event is linearly dependent on the total operation time. The penalty cost at each station is generated by the convex penalty function when the station's inventory level deviates from its target inventory level. During the repositioning, the vehicle can use multiple trips to fulfill the reposition event, and more importantly, for this study, there is no visit limitation for each station in the trip, meaning that the vehicle can choose any route it wants to fulfill the repositioning. Thus, this scenario is a realistic one.

To clarify this scenario, this study defines the following terms and delimitations using the same definitions as in previous chapters. The *vehicle* is defined as the

transportation equipment used to carry the bicycles among the different stations with the capacity to carry at most *c* units, while *station* is the place where customers can rent or return bicycles; the total number of lockers in the station is defined as the *station capacity*. The *depot* is defined as the distribution center and warehouse for the *vehicle* and the bicycles; it has the same function as the station, the only difference being that the *station* has a finite bicycle inventory and *station capacity* while it is assumed the *depot* has an infinite bicycle inventory and *station capacity*. *Trip* is defined as the vehicle routing sequence that starts at the *depot* and goes to several *stations* before returning to the *depot*. The *visit point* represents the visited *station* or *depot* in the vehicle routing sequence, while the *target* value is the inventory level designated for each *station* that is repositioned to. A *visit point* is a *pickup station* when its current inventory is greater than its target value, while a *visit point* is a *drop off station* when its current inventory is less than its *target* value. If a *station* has been visited multiple times in the solution, it is referred to as a *complex station* for the solution; otherwise, it is defined as a *simple station* for the solution.

The most important contribution of this research is that the same station could be visited multiple times in the same *trip.* This assumption relaxes the visit limit constraint, making the model more closely resemble the real world. However, with this relaxation, using only the station type visited (*simple station* or *complex station*) based on the *visit point* cannot identify its status. To address this issue, a *visit point* is categorized as 3 types: (1) *pure simple visit point*: the *visit point* which visits a *simple station*, (2) s*imple complex visit point*: the *visit point* which visits a *complex station* only once in the current

trip, (3) *multi-complex visit point*: the *visit point* which visits a *complex station* more than once in the current *trip*.

Heuristic Algorithm

This research uses a VNS based algorithm to find the heuristic solution for the SBRP problem. Similar to the algorithm used in Chapter 4, the one here also uses the one step neighborhood function (changing both the number loaded / unloaded and the vehicle routing sequence at the same time) to generate new solutions. The primary difference between the current and previous algorithms is that the current one includes the modification for the scenario when the same station is visited multiple times during the same trip.

The rest of this section is structured as follows: first, the structure of the solution for the SBRP problem is analyzed and symbols representing the solutions defined; then the pseudo code for the VNS algorithm introducing the structure of the heuristic algorithm is given, and finally the algorithm and its related terms and details, such as the initial solution and neighborhood functions, are explained.

Analysis of the SBRP problem solution

In general, the solution for the SBRP problem involves a vehicle routing sequence and loading / unloading plan for each station visited. More specifically, for this research,

the solution is composed of two parts: (1) a qualified vehicle routing sequence whose total traveling time does not violate the time horizon limitation and (2) the associated loading / unloading plan for each station for the routing sequence generated in part (1). For consistency, the same symbol $x =< r, a >$ used in the previous chapters is used here to define a solution. The r represents the routing schedule and a the loading / unloading plan at each station for the routing sequence r . For this routing sequence, $r = (r_{0^+,1}, r_{11}, ..., r_{\rho_1 1}, r_{0^-,1}, ..., r_{0^+,k}, r_{1k}, ..., r_{\rho_k k}, r_{0^-,k})$, where $r_{p,k}$ is the station ID for the p th stop in the *k* th *trip*, and ρ_k is the total number of stations visited in *k* th *trip*. The r_{0^+} and r_{0^-} represent the depot start and end point, respectively. Since no limitation for the maximum number of trips used in one solution is set, the variable *k*, i.e. the total number of trips used in the solution, is an uncertain number; as described in chapter 4, we can get the upper bound for number of trips in the solution, $k^{\max} = \lceil T/(2\mu) \rceil$, where the μ is the travel time from the depot to the closest station and *T* is the limit time horizon.

The associated loading / unloading plan for routing sequence *r*, referred to as the assignment plan, is $a = (a_{0^+,1}, a_{11}, ..., a_{\rho_1 1}, a_{0^-,1}, ..., a_{0^+,k}, a_{1k}, ..., a_{\rho_k k}, a_{0^-,k})$, where $a_{p,k}$ is the number of bicycles loaded / unloaded at station $r_{p,k}$ at the *p* th stop in the *k* th *trip*. Using the routing sequence *r* and its related assignment plan *a*, the inventory level on the vehicle / at the station when the vehicle departures from each station can be calculated: $v_{p,k}$ is defined as the inventory level on the vehicle when it leaves station $r_{p,k}$ on *p* th stop in the *k* th trip and $I_{r_{p,k},k}$ as the corresponding inventory level. All values can be

calculated using the formulas $v_{p,k} = v_{p-1,k} + a_{p,k}$ and $I_{r_{p,k},k} = I_{r_{p,k},k-1} - a_{p,k}$. For special cases, $v_{0^+,1} = a_{0^+,1}$, $I_{i,0} = l_i^b$, i=0,1,2,…,n is used.

The routing sequence r and the assignment plan a must have elements in the solution that are highly connected. Although the inventory level on the vehicle and at the station can be calculated from the solution $x =< r, a >$, it is easier to check the feasibility of solution using these 2 variables. Thus, the inventory level on the vehicle and the station are included in solution in a new formula, called the full solution:

$$x =< r, a, v, I >= (< r_{0^+,1}, a_{0^+,1}, v_{0^+,1}, I_{r_{0^+,1},1} >, ..., < r_{0^-,k}, a_{0^-,k}, v_{0^-,k}, I_{r_{0^-,k},k} >)$$

The tuple $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ represents the p th stop in the k th *trip* for station $r_{p,k}$ and the number of bicycles loaded / unloaded $a_{p,k}$, with the inventory level on vehicle after this repositioning being $v_{p,k}$ and the inventory level left at the station $I_{r_{p,k},k}$.

Variable Neighborhood Search Algorithm

The variable neighborhood search algorithm, a meta-heuristic algorithm created by Mladenovic and Hansen in 1997, is based on the local search algorithm combined with the distant neighborhood search to solve the global optimization. Listed below are the pseudo code steps of the general VNS algorithm:

Repeat following sequence until the stopping condition is met:
    Set $k \leftarrow 1$;
    Repeat the following steps until $k = k_{max}$

Shaking. Generate a solution x' at random from the k^{th} Neighborhood function $x' = N_k(x)$

Local search

(b1) Set $l \leftarrow 1$;

(b2) Repeat following steps until $l = l_{max}$

- Exploration of neighborhood. Find the best neighbor $x'' = N_l(x')$
- Move or not. If $f(x'') < f(x')$, set $x' \leftarrow x''$ and $l \leftarrow 1$; otherwise set $l \leftarrow l+1$

Move or not. If this local optimum is better than the incumbent, move there ($x \leftarrow x''$), and continue the search with $N_1 (k \leftarrow 1)$; otherwise, set $k \leftarrow k + 1$

In this research, the combination of the insert point function I(x) and the delete function D(x) are used to generate the sharking neighborhood function, while the improvement function P(x) is used to create the local search neighborhood function. These three functions will be discussed in detail in later sections.

The Initial Solution

The initial solution is the start search point for the algorithm, with a good initial solution reducing the total search time. This section details several methods based on various rules for providing the initial solutions. In general, these initial solutions are based on two principles: (1) Randomness principles. The randomness property makes the initial solution randomly scattered in the solution space, potentially increasing the robustness of the algorithm. (2) Quality principle. In general, good solutions should share some property that results in a better objective value. The initial solution with a better objective should be close to the global optimum. Similar to the concept in the previous chapter, a pickup station is one which can reduce the penalty cost because it has bicycles

available for pickup (i.e. the current inventory level is greater than its target inventory level). On the other hand, a drop off station is one which can reduce the penalty cost because it can accept additional bicycles (i.e. the current inventory level is less than its target inventory level).

In this research, we use the following method to generate the initial solutions, randomly selecting in each run which method is used:

1. Randomly Single Alternative Selection:

Based on the current inventory level and target value, all stations can be categorized into two large sets: a pickup set and a drop off set. This method alternately selects stations from the pickup set and drop off set, inserting the station selected into the vehicle routing sequence. The number of bicycles loaded / unloaded quantity for this station is the number which provides the minimal penalty cost without violating the vehicle capacity constraint and station capacity constraint. Once a station selected is repositioned to its target value, it is removed from the pickup / drop off set; otherwise, it remains in the pickup / drop off set. The selection continues until the total traveling time violates the working time horizon constraint.

2. Randomly Full Load Alternative Selection:

Similar to the Randomly Single Alternative Selection method, this method selects pickup stations and drop off stations full vehicle load alternately, the only difference being when to change the selection between the two sets. If the pickup station selected (drop off station) does not fulfill (empty) the vehicle capacity, then the next station

selected is again from the pickup set (drop off set) until the vehicle had been fulfilled (emptied); then, a station from the drop off set (pickup set) is selected next.

3. Penalty Cost Priority Selection:

This method selects the station which gives the best penalty cost reduction, adding it to the routing sequence. In this method, the selection principle is based solely on the reduction of the penalty cost without considering whether the station is a pickup or drop off station.

4. Travel Cost Priority Selection:

This method selects the station which gives the smallest travelling cost increase, adding it to the routing sequence. It uses the same steps as the Penalty Cost Priority Selection, with the only difference being that the selection is changed from the largest penalty cost reduction to smallest travelling cost increase.

5. Cost Ratio Priority Selection:

This method selects the station which gives the best ratio of penalty cost reduction over travelling cost; thus, it not only considers the reduction in the penalty cost but also the total traveling time. Since this research includes a working time horizon constraint, this method has the potential to use the time more efficiently.

The Neighborhood Function

The neighborhood $N(x)$ is the new solution generated from the current solution x with a small modification. This research uses two basic neighborhood functions: (1) the insert neighborhood function $I(x)$, and (2) the removal neighborhood function $R(x)$. All the other neighborhood functions used here are created by combining these two. As explained earlier, the solution for the SBRP problem includes not only the vehicle routing sequence but also the assignment plan associated with each station visited on the route. The insert and removal neighborhoods change both the vehicle routing sequence and its associated assignment plan to generate a new solution.

Both the insert and removal neighborhood functions modify the solution based on a visit point in the solution, defined here as three types: (1) pure simple visit point, (2) simple complex visit point, and (3) multi-complex visit point. The pure simple visit point involves a simple station, which is a unique visit point in the solution. The remaining two types are complex visit points involving visits to complex stations. More specifically, the simple complex visit point only visits a complex station once in a current trip, while the multi-complex visit point visits a complex station more than once in a current trip. For example, if 3 visit points visit station i, and visit point 1 and visit point 2 are both in the first trip and visit point 3 is in the second, by definition both visit point 1 and 2 are multi-complex visit points and visit point 3 is a simple complex visit point. As we can see, because of the at most once visit limitation constraint, all visit points in Chapter 3 are pure single visit points, with the visit points in Chapter 4 including both pure single visit points and single complex visit points. The research in this chapter includes all 3 types of visit points.

To be consistent, it is assumed that the modified (inserted or removed) *visit point* is $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$, which visits station i in the p th position in the k th *trip* ($r_{p,k} = i$). Simplifying the notation results in visit point $< i, a_{p,k}, v_{p,k}, I_{i,k} >$. If the visit point $< i, a_{p,k}, v_{p,k}, I_{i,k} >$ is a single complex visit point, and there are ρ visit points visiting station i after this visit point, then the remaining visit points are $< i, a_{p_l,k_l}, v_{p_l,k_l}, I_{i,k_l} >$, where $k_l > k$, $l = 1,..., \rho$. If the visit point $< i, a_{p,k}, v_{p,k}, I_{i,k} >$ is a multi-complex visit point and there are ρ_l visit points visiting station i after this visit point in the l th trip, then the remaining visit points in the current trip are $< i, a_{q_j^k,k}, v_{q_j^k,k}, I_{i,k} >$, where $q_j^k > p$, $j = 1, 2,..., \rho_i$ and all in the later *trips* are $< i, a_{q_j^l,l}, v_{q_j^l,l}, I_{i,l} >$, where $l > k$.

Proposition 5.1:

Suppose a complex visit point (i.e. either a single complex visit point or a multi-complex visit point) visits complex station i and there are ρ visit points visiting station i after this visit point. Assume all of the remaining visit points are $< i, a_{t_l,k_l}, v_{t_l,k_l}, I_{i,k_l} >$, where $k \le k_l$, $l = 1,..., \rho$. To obtain the sequence of these points, we assume $k_j \le k_{j+1}$. For visit point $< i, a_{t_l,k_l}, v_{t_l,k_l}, I_{i,k_l} >$, if only its inventory level is changed, $I'_{i,k_l} = I_{i,k_l} + \Delta$ and the assignment plan remains unchanged, the changed inventory must be within the range $\Delta \in [\alpha_1, \alpha_2]$,where $\alpha_1 = \max_{q=k_l,...,k_n}\{-I_{i,q}\}$ and $\alpha_2 = \min_{q=k_l,...,k_n}\{s_i - I_{i,q}\}$ to ensure the new solution is feasible.

Proof:

If visit point $< i, a_{t_l,k_l}, v_{t_l,k_l}, I_{i,k_l} >$ increases its inventory level with Δ units, then $I'_{i,k_l} = I_{i,k_l} + \Delta$. If all assignment plans are not changed, the inventory level at station *i* increases Δ units for all visit points after $< i, a_{t_l,k_l}, v_{t_l,k_l}, I_{i,k_l} >$, meaning all visit points after $< i, a_{t_q,k_q}, v_{t_q,k_q}, I_{i,k_q} >$ have $0 \le I_{i,k_q} + \Delta \le s_i$. Solving this inequality results in $\Delta \in [\alpha_1, \alpha_2]$ where $\alpha_1 = \max_{q=k_l,\ldots,k_n}\{-I_{i,q}\}$, $\alpha_2 = \min_{q=k_l,\ldots,k_n}\{s_i - I_{i,q}\}$

Proposition 5.2:

Assume two visit points: $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ and $< r_{q,k}, a_{q,k}, v_{q,k}, I_{r_{q,k},k} >$ in the *k* th *trip* where $p < q$ and the change $a'_{p,k} = a_{p,k} + \Delta$, $a'_{q,k} = a_{q,k} - \Delta$ to obtain a new trip. The new trip is feasible only when the modified bicycle units satisfy $\Delta \in [\beta_1, \beta_2]$, where $\beta_1 = \max_{t\in[p,q)}\{-v_{t,k}, I_{r_{p,k},k} - s_{r_{p,k}}, -I_{r_{q,k},k}\}$, $\beta_2 = \min_{t\in[p,q)}\{c - v_{t,k}, s_{r_{q,k}} - I_{r_{q,k},k}, I_{r_{p,k},k}\}$.

Proof:

At visit point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$, Δ more bicycle units are loaded into the vehicle. This modification changes the assignment plan $a'_{p,k} = a_{p,k} + \Delta$, the on-vehicle inventory level $v'_{p,k} = v_{p,k} + \Delta$, and the inventory level at station $I'_{r_{p,k},k} = I_{r_{p,k},k} - \Delta$. All the visit points between these 2 visit points $< r_{t,k}, a_{t,k}, v_{t,k}, I_{r_{t,k},k} >$, where $p < t < q$ only realize a change in the on-vehicle inventory level $v'_{t,k} = v_{t,k} + \Delta$, where $p < t < q$. At visit point $< r_{q,k}, a_{q,k}, v_{q,k}, I_{r_{q,k},k} >$, Δ less bicycle units are loaded into the vehicle. So, only the

assignment plan $a'_{q,k} = a_{q,k} - \Delta$ and inventory level at station $I'_{r_{q,k},k} = I_{r_{q,k},k} + \Delta$ are changed. To maintain the feasibility of this trip, these updated parameters need to satisfy the constraints, resulting in $\Delta \in [\beta_1, \beta_2]$, where $\beta_1 = \max_{t\in[p,q)}\{-v_{t,k}, I_{r_{p,k},k} - s_{r_{p,k}}, -I_{r_{q,k},k}\}$ and $\beta_2 = \min_{t\in[p,q)}\{c - v_{t,k}, s_{r_{q,k}} - I_{r_{q,k},k}, I_{r_{p,k},k}\}$.

The Removal Neighborhood Function

The removal neighborhood function *R*(*x*), a basic neighborhood function, generates a new solution *x'*=*R*(*x*) by removing one visit point from the current solution. This visit point can either be visiting a station or visiting the *depot*, and then the assignment plan is adjusted to try to make the new solution feasible by making as small a change as possible. This research focuses on 3 types of assignment plan modifications for the removal neighborhood function: (1) changing the predecessor *visit point*'s assignment plan, (2) changing the successor visit point's assignment plan, and (3) changing the *depot*'s assignment plan. If the visit point removed is a station, a maximum of four feasible solutions are generated with the same routing sequence but with different loading / unloading plans. If the visit point removed is a *depot*, a maximum of three feasible solutions are generated. The method is detailed in the following section. To be consistent, it is assumed the point removed is $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ in solution *x*.

1. Remove the pure simple visit point

The pure simple visit point visits a simple station, which is visited only once in the entire solution, meaning removing it affects only the current trip. For the new solution to be feasible, it is necessary only to ensure that the current trip is feasible after the removal event. In total, there are 3 ways to change the assignment plan when the pure simple visit point is removed.

a. Adjust a depot - Adjust the $a_{0^+,k}$ or $a_{0^-,k}$

In this method, the assignment plan at the *depot* is adjusted by changing the number of bicycles loaded / unloaded to cover the units needed to be loaded / unloaded at the visit point removed. Denote η_1 as the residual vehicle capacity for *k* th trip before visit point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$. Denote η_2 as the residual vehicle capacity for *k* th trip after visit station $r_{p,k}$. Thus, $\eta_1 = \min_{t=0^+,1,\ldots,p-1}\{c - v_{t,k}\}$ and $\eta_2 = \min_{t=p+1,\ldots,\rho_k}\{c - v_{t,k}\}$. If $0 < a_{p,k} \le \eta_1$, then $a_{p,k}$ bicycle units can be picked at the *depot* at the beginning of *k* th trip without violating the vehicle capacity constraint. If $-\eta_2 \le a_{p,k} < 0$, then the $|a_{p,k}|$ extra bicycles units can be dropped off at the *depot* at the end of *k* th trip without violating the vehicle capacity constraint. Combining these 2 conditions results in the conclusion that if $a_{p,k} \in [-\eta_2, \eta_1]$, then the *visit point* at the *depot* can be adjusted, making $a'_{0^+,k} = a_{0^+,k} + a_{p,k}$ or $a'_{0^-,k} = a_{0^-,k} + a_{p,k}$ to obtain a new feasible solution.

b. Adjust the previous visit point - Adjust $a_{p-1,k}$

In this method, the assignment plan at the predecessor visit point is adjusted to cover the extra bicycles loaded / unloaded when visit point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ is removed. Moving the loaded / unloaded bicycle units to the predecessor visit point results in $a'_{p-1,k} = a_{p-1,k} + a_{p,k}$. The new inventory level at this visit point after the repositioning is $I'_{p-1,k} = I_{p-1,k-1} - a'_{p-1,k} = I_{p-1,k-1} - a_{p,k}$.

Suppose the predecessor visit point $< j, a_{p-1,k}, v_{p-1,k}, I_{j,k} >$ visits station *j*; the station capacity constraint needs to be checked to ensure the new trip generated is feasible. If the predecessor visit point $< j, a_{p-1,k}, v_{p-1,k}, I_{j,k} >$ is a pure simple visit point, then $0 \le I_{j,k-1} - a_{p,k} \le s_j$ (i.e. $a_{p,k} \in [I_{j,k-1} - s_j, I_{j,k-1}]$), and the new solution is feasible when $a_{p,k} \in [I_{j,k-1} - s_j, I_{j,k-1}]$. If the predecessor visit point is a complex visit point and later visit points which visit station *j* will be affected, based on proposition 5.1, the new generated solution is feasible when $a_{p,k} \in [\alpha_1, \alpha_2]$ where $\alpha_1 = \max_{q=k_l,\ldots,k_n}\{-I_{j,q}\}$ and $\alpha_2 = \min_{q=k_l,\ldots,k_n}\{s_j - I_{j,q}\}$. Combining these 2 ranges, the new generated solution is feasible when $a_{p,k} \in [\max\{\alpha_1, I_{j,k-1} - s_j\}, \min\{\alpha_2, I_{j,k-1}\}]$.

c. Adjust the successor visit point - Adjust $a_{p+1,k}$

In this method, the assignment plan at the successor visit point is adjusted to cover the extra bicycles loaded / unloaded when visit point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ is removed. Moving the loaded / unloaded bicycle units to the successor visit point results

in $a'_{p+1,k} = a_{p+1,k} + a_{p,k}$. The new inventory level at the predecessor after the repositioning is $I'_{p+1,k} = I_{p+1,k-1} - a'_{p+1,k} = I_{p+1,k-1} - a_{p,k}$.

Suppose the successor visit point $< j, a_{p+1,k}, v_{p+1,k}, I_{j,k} >$ visits station *j*; the station capacity constraint needs to be checked to ensure the new trip generated is feasible. If the successor visit point $< j, a_{p+1,k}, v_{p+1,k}, I_{j,k} >$ is a pure simple visit point, then $0 \le I_{j,k-1} - a_{p,k} \le s_j$ (i.e. $a_{p,k} \in [I_{j,k-1} - s_j, I_{j,k-1}]$), and the new solution is feasible when $a_{p,k} \in [I_{j,k-1} - s_j, I_{j,k-1}]$. If the successor visit point is a complex visit point and later visit points which visit station *j* will be affected, based on proposition 5.1, the new solution generated is feasible when $a_{p,k} \in [\alpha_1, \alpha_2]$ where $\alpha_1 = \max_{q=k_l,\ldots,k_n}\{-I_{j,q}\}$ and $\alpha_2 = \min_{q=k_l,\ldots,k_n}\{s_j - I_{j,q}\}$. Combining these 2 ranges, the new solution generated is feasible when $a_{p,k} \in [\max\{\alpha_1, I_{j,k-1} - s_j\}, \min\{\alpha_2, I_{j,k-1}\}]$.

2. Remove the *simple complex visit point*

As stipulated in the model, a vehicle can make multiple trips to complete the repositioning event. For a station, for example station *i*, the inventory level at station *i* in the *k* th trip, $I_{i,k}$, after the repositioning is the initial inventory level at station *i* for the *k*+1 th trip. If visit point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ is a simple complex visit point, it is the only visit point visiting station *i* in the *k* th *trip*. If visit point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ is removed, the inventory level at station *i* after repositioning is changed

$I'_{i,k} = I_{i,k-1} = I_{i,k} + a_{p,k}$. Removing a simple complex visit point might affect the feasibility of the current and all successive trips.

Based on proposition 5.1, the feasibility of all trips after the *k* th trip is guaranteed by satisfying constraint $a_{p,k} \in [\alpha_1, \alpha_2]$ where $\alpha_1 = \max_{q=k_l,\ldots,k_n}\{-I_{i,q}\}$ and $\alpha_2 = \min_{q=k_l,\ldots,k_n}\{s_i - I_{i,q}\}$. Using the method for removing a pure simple visit point makes the current trip feasible. Thus, to remove a simple complex visit point, then the remove a pure simple visit point method could be used while at the same time checking the constraint $a_{p,k} \in [\alpha_1, \alpha_2]$ where $\alpha_1 = \max_{q=k_l,\ldots,k_n}\{-I_{i,q}\}$, $\alpha_2 = \min_{q=k_l,\ldots,k_n}\{s_i - I_{i,q}\}$. If the $a_{p,k}$ is within this range, the new solution generated is feasible.

3. Remove the multi-complex visit point

If visit point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ is a multi-complex visit point and it visits station *i*, by definition, this solution will include multiple visit points visiting station *i* in the *k* th trip, meaning the inventory level at station *i* can change multiple times in the *k* th trip. The 3 methods for removing a multi-complex visit point proposed in this research can generate a maximum of 5 possible new solutions.

a. Using the remove a simple complex visit point method

This method follows the same procedure as the remove a simple complex visit point method. First, the remove a pure simple visit point method is used to generate the new solution *x*', while at the same time creating a feasibility constraint for $a_{p,k}$ called

constraint 1. Then proposition 5.1 is used to create a second feasibility constraint for $a_{p,k}$, called constraint 2. Constraint 1 guarantees the feasibility of the *k* th trip, while constraint 2 guarantees the feasibility of the successive trips after the *k* th *trip*. If the value of $a_{p,k}$ can satisfy both constraint 1 and 2, then the new solution *x*' is feasible. Using this method can result in a maximum of 3 possible new solutions.

b. Adjust prior multi-complex visit point

If the visit point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ removed is a multi-complex visit point and it visits station *i* ($r_{p,k} = i$), by definition, there is at least has one more visit point visiting station *i* in the *k* th trip. If the solution includes a visit point which visits station *i* prior to the removed visit point, for example $< r_{q,k}, a_{q,k}, v_{q,k}, I_{r_{q,k},k} >$ where $q < p$, then the assignment plan at visit point $< r_{q,k}, a_{q,k}, v_{q,k}, I_{r_{q,k},k} >$ can be adjusted and the predecessor multi-complex visit point takes the extra units $a'_{q,k} = a_{q,k} + a_{p,k}$. In this way, the inventory level for station *i* do not change for the current trip after the adjustment. The feasibility of the current trip after the adjustment guarantees the feasibility of the new solution. Based on proposition 5.2, the trip after the adjustment is feasible when $a_{p,k} \in [\beta_1, \beta_2]$, where $\beta_1 = \max_{t \in [p,q)} \{v_{t,k} - c, I_{r_{q,k},k} - s_{r_{q,k}}, -I_{r_{p,k},k}\}$ and $\beta_2 = \min_{t \in [p,q)} \{v_{t,k}, s_{r_{p,k}} - I_{r_{p,k},k}, I_{r_{q,k},k}\}$. When these constraints are satisfied, the adjusted solution is feasible.

c. Adjust successor multi-complex visit point

If the removed visit point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ is a multi-complex visit point and it visits station *i* ($r_{p,k} = i$), by definition, there is at least one more visit point visiting station *i* in the *k* th *trip*. If the solution includes a visit point which visits station *i* prior to the removed visit point, for example $< r_{q,k}, a_{q,k}, v_{q,k}, I_{r_{q,k},k} >$ where $p < q$, then the assignment plan at visit point $< r_{q,k}, a_{q,k}, v_{q,k}, I_{r_{q,k},k} >$ is adjusted; the predecessor multi-complex visit point can take the extra units $a'_{q,k} = a_{q,k} + a_{p,k}$. In this way, the inventory level for station *i* do not change for the current *trip* after the adjustment. The feasibility of the current trip after the adjustment can guarantee the feasibility of the new solution. Based on proposition 5.2, the trip after the adjustment is feasible when $a_{p,k} \in [\beta_1, \beta_2]$, where $\beta_1 = \max_{t\in[p,q)}\{-v_{t,k}, I_{r_{p,k},k} - s_{r_{p,k}}, -I_{r_{q,k},k}\}$ and $\beta_2 = \min_{t\in[p,q)}\{c - v_{t,k}, s_{r_{q,k}} - I_{r_{q,k},k}, I_{r_{p,k},k}\}$. Once these constraints are satisfied, the adjusted solution is feasible.

4. Visit point removed is the *depot*

A visit point that is a *depot* can also be removed from the solution. Since there is no visiting limitation for stations in this research, any routing schedule is allowed as long as the total travelling time is within the time horizon. In this research, the *depot* visit point comes in pairs, i.e. always loading at 0^+ *depot* and unloading at 0^- *depot*, except for the beginning and ending visit points. The end of the *k* th trip is the beginning of the *k*+1 th trip, i.e. $r_{0^-,k}, r_{0^+,k+1}$, meaning paired *depot* visit points must be removed at the same

time, with these paired *depot* visit points being considered as one visit point. Except for this change, the method for removing a *depot* visit point is the same as for removing a pure simple visit point.

The Insert Neighborhood Function

Like the removal neighborhood function, the insert neighborhood function $I(x)$ is a basic neighborhood function which generates a new solution $x'=I(x)$, except by inserting into rather than removing one visit point from the current solution. The visit point to be inserted can be either visiting a station or visiting the *depot*. The assignment plan is then adjusted to try to make the new solution feasible by making as a small change as possible. Similar to the method used in the removal neighborhood function, this research focuses on 3 types of assignment plan change: (1) changing the assignment plan of the predecessor *visit point*, (2) changing the assignment plan of the successor *visit point*, and (3) changing the *depot*'s assignment plan. To be consistent, it is assumed that the point removed is $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ in solution x.

1. Insert a pure simple visit point

Since a pure simple visit point is visited only once in the solution, the insert adjustment affects the feasibility of only the current trip. This research proposes 3 ways to adjust the assignment plan and possibly maintain the feasibility of the solution.

a. Adjust a depot assignment - Adjust the $a_{0^+,k}$ or $a_{0^-,k}$

In this method, the assignment plan at the *depot* is adjusted to load / unload more bicycle units to cover the units needed to be loaded / unloaded at the visit point inserted. The definition of residual vehicle capacity before and after the visit point is the same as the one used previously, $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ in the k th *trip*, where $\eta_1 = \min_{t=0^+,1,\ldots,p-1}\{c - v_{t,k}\}$, $\eta_2 = \min_{t=p+1,\ldots,\rho_k}\{c - v_{t,k}\}$. If $0 \le a_{p,k} \le \eta_2$, then the $a_{p,k}$ extra bicycles can be dropped at the *depot* in the end of the k th trip, $a'_{0^-,k} = a_{0^-,k} - a_{\rho_k k}$. However, if $-\eta_1 \le a_{p,k} \le 0$, then the $a_{p,k}$ bicycle can be picked at the *depot* at the beginning of k th *trip*, $a'_{0^+,k} = a_{0^+,k} - a_{p,k}$. Combining these 2 conditions results in a new feasible solution when $a_{p,k} \in [-\eta_1, \eta_2]$, where $\eta_1 = \min_{t=0^+,1,\ldots,p-1}\{c - v_{t,k}\}$ and $\eta_2 = \min_{t=p+1,\ldots,\rho_k}\{c - v_{t,k}\}$.

b. Adjust the previous visit point - Adjust $a_{p-1,k}$

In this method, the assignment plan at the predecessor visit point is adjusted to cover the extra loaded / unloaded bicycle units when visit point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ is inserted. The predecessor visit point is adjusted $a'_{p-1,k} = a_{p-1,k} - a_{p,k}$. The new inventory level at the predecessor after the repositioning is $I'_{p-1,k} = I_{p-1,k-1} + a_{p,k}$, and the inventory level on vehicle is $v'_{p-1,k} = v_{p-1,k} - a_{p,k}$. For the inserted visit point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$,

the on-vehicle inventory level and station inventory level are

$v_{p,k} = v_{p-1,k} - a_{p,k} + a_{p,k} = v_{p-1,k}$ and $I_{r_{p,k},k} = I_{r_{p,k},k-1} - a_{p,k}$, respectively.

Suppose the predecessor visit point visits station *j*, and the inserted visit point visits station *i*, the predecessor visit point can be rewritten as $< j, a_{p-1,k}, v_{p-1,k}, I_{j,k} >$ and the inserted visit point as $< i, a_{p-1,k}, v_{p-1,k}, I_{i,k} >$. If the predecessor visit point is also a pure simple visit point, the feasibility of the predecessor and inserted visit points needs to be guaranteed. Satisfying the vehicle capacity and station capacity constraint results in $0 \le I_{j,k-1} + a_{p,k} \le s_j$, $0 \le v_{p-1,k} - a_{p,k} \le c$ and $0 \le I_{i,k-1} - a_{p,k} \le s_i$. The new solution is feasible when $a_{p,k} \in [\chi_1, \chi_2]$ where $\chi_1 = \max\{I_{i,k-1} - s_i, -I_{j,k-1}, c - v_{p-1,k}\}$ and $\chi_2 = \min\{I_{i,k-1}, s_j - I_{j,k-1}, v_{p-1,k}\}$.

If the predecessor visit point is a complex visit point, the feasibility not only of the current trip but also of all trips after the current trip needs to be guaranteed. The previous analysis provides the feasible range of $a_{p,k}$ for the current trip. Based on proposition 5.1, all the rest of the trips are feasible when $a_{p,k} \in [\alpha_1, \alpha_2]$ where $\alpha_1 = \max_{q=k_l,\ldots,k_n}\{-I_{j,q}\}$ and $\alpha_2 = \min_{q=k_l,\ldots,k_n}\{s_j - I_{j,q}\}$. By combining these two conditions, the new solution is feasible when $a_{p,k} \in [\phi_1, \phi_2]$ where $\phi_1 = \max\{\alpha_1, \chi_1\}$, $\phi_2 = \min\{\alpha_2, \chi_2\}$.

c. Adjust the successor visit point - Adjust $a_{p+1,k}$

Similar to the adjust predecessor visit point method, this method adjusts the successor visit point to cover the extra loaded / unloaded bicycle units when the visit point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ is inserted. The inserted visit point is represented as $v_{p,k} = v_{p-1,k} + a_{p,k}$ and $I_{r_{p,k},k} = I_{r_{p,k},k-1} - a_{p,k}$, and the successor visit point is adjusted to $a'_{p-1,k} = a_{p-1,k} - a_{p,k}$. The new inventory level at the predecessor after the repositioning is $I'_{p-1,k} = I_{p-1,k-1} + a_{p,k}$, and the inventory level on the vehicle is $v'_{p-1,k} = v_{p-1,k} + a_{p,k} - a_{p,k} = v_{p-1,k}$.

Suppose the successor *visit point* visits station *j*, and the inserted visit point visits station *i*, the successor visit point is rewritten as $< j, a_{p-1,k}, v_{p-1,k}, I_{j,k} >$ and the inserted visit point as $< i, a_{p-1,k}, v_{p-1,k}, I_{i,k} >$. If the predecessor visit point is a pure simple visit point, the feasibility of these inserted and successor visit points needs to be guaranteed. *S*atisfying the vehicle capacity and station capacity constraint results in $0 \le I_{i,k-1} - a_{p,k} \le s_i$, $0 \le v_{p-1,k} + a_{p,k} \le c$ and $0 \le I_{j,k-1} + a_{p,k} \le s_j$. The new solution generated is feasible when $a_{p,k} \in [\chi_1, \chi_2]$ where $\chi_1 = \max\{I_{i,k-1} - s_i, -I_{j,k-1}, -v_{p-1,k}\}$ and $\chi_2 = \min\{I_{i,k-1}, s_j - I_{j,k-1}, c - v_{p-1,k}\}$. If the successor visit point is a complex visit point, the feasibility not only of the current trip but also of all successive trips must be guaranteed. The previous analysis provides the feasible range of $a_{p,k}$ for the current trip. Based on proposition 5.1, the remaining trips are feasible when $a_{p,k} \in [\alpha_1, \alpha_2]$ where

$\alpha_1 = \max_{q=k_l,\ldots,k_n}\{-I_{j,q}\}$ and $\alpha_2 = \min_{q=k_l,\ldots,k_n}\{s_j - I_{j,q}\}$. By combining these 2 conditions, the new solution is feasible when $a_{p,k} \in [\phi_1, \phi_2]$ where $\phi_1 = \max\{\alpha_1, \chi_1\}$, $\phi_2 = \min\{\alpha_2, \chi_2\}$.

2. Insert a simple complex visit point

As stipulated in the model, the vehicle can use multiple trips to complete the repositioning event. For a station, for example station *i*, the inventory level after repositioning at station *i* in the *k* th trip, $I_{i,k}$, is the initial inventory level at station *i* for the *k*+1 th trip. If the visit point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ is a simple complex visit point in current trip, it is the only visit point that visits station *i* in the *k* th trip. By inserting visit point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$, the inventory level at station *i* after repositioning becomes $I'_{i,k} = I_{i,k-1} = I_{i,k} - a_{p,k}$. Inserting this simple complex visit point may affect the feasibility of the current and all successive trips.

Based on proposition 5.1, the feasibility of all trips after *k* th trip is guaranteed by satisfying constraint $a_{p,k} \in [\alpha_1, \alpha_2]$ where $\alpha_1 = \max_{q=k_l,\ldots,k_n}\{-I_{i,q}\}$ and $\alpha_2 = \min_{q=k_l,\ldots,k_n}\{s_i - I_{i,q}\}$. Using the method for inserting a pure simple visit point makes the current trip feasible. Thus, to insert a simple complex visit point, the "remove a pure simple visit point" method are applied at the same time to check the constraint $a_{p,k} \in [\alpha_1, \alpha_2]$ where $\alpha_1 = \max_{q=k_l,\ldots,k_n}\{-I_{i,q}\}$ and $\alpha_2 = \min_{q=k_l,\ldots,k_n}\{s_i - I_{i,q}\}$. If $a_{p,k}$ is within this range, the new generated solution is feasible.

3. Insert the multi-complex visit point

If visit point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ is a multi-complex visit point and it visits the station *i*, by definition, this solution includes multiple visit points visiting station *i* in the *k* th trip, meaning that the inventory level at station *i* is changed multiple times during this trip. This research proposes three methods for inserting a multi-complex visit point, generating a maximum of five possible new solutions.

a. The same method as insert simple complex visit point

This method uses the same steps as the inserting simple complex visit point method. First, the insert simple complex visit point method is used to generate a new solution *x*', while at the same time creating a feasibility constraint for $a_{p,k}$, called constraint 1. Then, proposition 5.1 is used to create a second feasibility constraint for $a_{p,k}$, called constraint 2. Constraint 1 guarantees the feasibility of the *k* th *trip*, while constraint 2 guarantees the feasibility of the trips after the *k* th trip. If the value of $a_{p,k}$ satisfies both constraint 1 and 2, then the new solution *x*' is feasible. Using this method results in a maximum of three possible new solutions.

b. Adjust prior multi-complex visit point

If the removed visit point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ is a multi-complex visit point and it visits station *i* ($r_{p,k} = i$), by definition, there is at least one more visit point visiting station *i* in the *k* th *trip*. If the solution includes a visit point which visits station *i* prior to

the removed visit point, for example $< r_{q,k}, a_{q,k}, v_{q,k}, I_{r_{q,k},k} >$ where $q < p$, then the assignment plan is adjusted at visit point $< r_{q,k}, a_{q,k}, v_{q,k}, I_{r_{q,k},k} >$ and the predecessor multi-complex visit point takes the extra units $a'_{q,k} = a_{q,k} - a_{p,k}$. Thus, the inventory level at station *i* do not change for the current trip after the adjustment. The feasibility of the current trip after the adjustment can guarantee the feasibility of the new solution. Based on proposition 5.2, the trip after the adjustment is feasible when $a_{p,k} \in [\beta_1, \beta_2]$, where $\beta_1 = \max_{t\in[p,q)}\{-v_{t,k}, I_{r_{p,k},k} - s_{r_{p,k}}, -I_{r_{q,k},k}\}$ and $\beta_2 = \min_{t\in[p,q)}\{c - v_{t,k}, s_{r_{q,k}} - I_{r_{q,k},k}, I_{r_{p,k},k}\}$. Once these constraints are satisfied, the adjusted solution is feasible.

c. Adjust successor multi-complex visit point

If the removed *visit point* $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ is a multi-complex visit point and it visits station *i* ($r_{p,k} = i$), by definition, there is at least one more visit point visiting station *i* in the *k* th trip. If the solution includes a visit point which visits station *i* after the removed visit point, for example $< r_{q,k}, a_{q,k}, v_{q,k}, I_{r_{q,k},k} >$ where $p < q$, then the assignment plan is adjusted at visit point $< r_{q,k}, a_{q,k}, v_{q,k}, I_{r_{q,k},k} >$ and the predecessor multi-complex visit point takes the extra units $a'_{q,k} = a_{q,k} - a_{p,k}$. The inventory level for station *i*, then, do not change for current trip after the adjustment. The feasibility of the current trip after the adjustment guarantees the feasibility of the new solution. Based on Proposition 5.2, the trip after the adjustment is feasible when $a_{p,k} \in [\beta_1, \beta_2]$, where

$\beta_1 = \max_{t\in[p,q)}\{v_{t,k} - c, I_{r_{q,k},k} - s_{r_{q,k}}, -I_{r_{p,k},k}\}$ and $\beta_2 = \min_{t\in[p,q)}\{v_{t,k}, s_{r_{p,k}} - I_{r_{p,k},k}, I_{r_{q,k},k}\}$. Once these constraints are satisfied, the adjusted solution is feasible.

4. Insert the *depot*

If the inserted visit point is a *depot*, it can also be inserted into the solution. Since there is infinite capacity and inventory at a *depot*, any inserted event for a depot is feasible.

The improvement method

In addition to the insert and removal neighborhood function, other improvement functions *P*(x) to modify the current solution to obtain a better objective value can be applied. These improvement methods change only the assignment plan, not the vehicle routing schedule, meaning they improve the solution by reducing the total penalty cost.

Improvement neighborhood function

1. Change one visit point's assignment plan

In this improvement function, we try to reduce the penalty cost by only changing one station's assignment plan. As the total number of bicycles loaded into and unloaded from the vehicle should be balanced for each trip, when the assignment plan in one visit point is changed, at least one other visit point's assignment plan must be changed. In this

method, both the visit point's assignment plan and the *depot*'s assignment plan are changed at the same time.

Suppose visit point $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ visits station i (i.e. $r_{p,k} = i$) whose penalty cost can be reduced (i.e. $I_i^a \neq t_i$). This visit point can be separated into 2 visit points: $< r_{p,k}, a_{p,k} + \Delta, v_{p,k} + \Delta, I_{r_{p,k},k} - \Delta >$ and $< r_{p,k}, -\Delta, v_{p,k}, I_{r_{p,k},k} >$, where $\Delta = 0, ..., I_i^a - t_i$. Using the remove function to remove visit point $< r_{p,k}, -\Delta, v_{p,k}, I_{r_{p,k},k} >$ results in a maximum of three possible new solutions *x*' which change only the assignment plan for current solution. Choosing the solution which reduces the highest amount of the penalty cost results in improving the current solution *x*.

2. Change two visit point's assignment plan

This improvement function is used to try to improve the current solution by changing the assignment plan of two visit points in the same trip to reduce the penalty cost. Since each station has a unique penalty cost function, correctly assigning the loaded / unloaded units between visit points will reduce the total penalty cost.

Suppose two visit points $< r_{p,k}, a_{p,k}, v_{p,k}, I_{r_{p,k},k} >$ and $< r_{q,k}, a_{q,k}, v_{q,k}, I_{r_{q,k},k} >$ in the the *k* th trip, where $p < q$ visit station *i* and station *j*, i.e. $r_{p,k} = i$ and $r_{q,k} = j$. Though the total number loaded / unloaded for these two visit points is fixed $a_{p,k} + a_{q,k}$, adjusting the balance between these two can reduce the total penalty cost.

Assume the adjustment amount is Δ, and these two new adjusted visit points are $< i, a_{p,k} + \Delta, v_{p,k} + \Delta, I_{i,k} - \Delta >$ and $< j, a_{q,k} - \Delta, v_{q,k}, I_{j,k} + \Delta >$. Because the on-vehicle inventory level changes at station *i*, all the on-vehicle inventory levels for the remaining visit points between these two automatically change to $< r_{s,k}, a_{s,k}, v_{s,k} + \Delta, I_{r_{s,k},k} >$ where $p < s < q$. To ensure the feasibility of this new solution, constraints $0 \le I_{i,k} - \Delta \le s_i$, $0 \le I_{j,k} + \Delta \le s_j$, $0 \le v_{p,k} + \Delta \le c$, and $0 \le v_{s,k} + \Delta \le c$ where $p < s < q$ need to be satisfied. Combining the constraints results in $\Delta \in [\gamma_1, \gamma_2]$, where

$\gamma_1 = \max_{t \in [p,q)} \{-v_{t,k}, -I_{j,k}, I_{i,k} - s_i\}]$ and $\gamma_2 = \min_{t \in [p,q)} \{c - v_{t,k}, s_j - I_{j,k}, I_{i,k}\}]$. By searching for Δ within this available range, the Δ^* that will maximize the total penalty cost reduction is found. When station *i* or station *j* are the complex stations, Proposition 5.1 is applied to determine the suitable range for the Δ to ensure the feasibility of the trips after the current trip.

Numeric Experiment

This research proposes a new heuristic (H3) to solve the SBRP problem with single vehicle, multiple trips and no visit limitation. To check the performance of the new heuristic (H3) proposed here, it is compared to the other two algorithms introduced in this research: (1) the H1 algorithm proposed in Chapter 3 for solving the SBRP problem with a single vehicle and a single trip with station visit limitation and (2) the H2 algorithm

proposed in Chapter 4 solving the SBRP problem with a single vehicle, multiple trips and partial station visit limitation.

This analysis uses the 52 testing scenarios from Ho and Szeto's (2014) research. To be consistent, the penalty cost is again used as the objective value. In addition, the same testing scenarios used for the H1 and H2 numeric experiments are again used here. Running the same testing scenario but with different algorithms allows for a comparison of their performances (i.e. solving time and quality of the solution). The following analysis compares the solving times and the objective values of the H1, H2 and H3 algorithms across the 52 testing scenarios.

All testing was conducted on a Dell notebook with an Intel Core i5-2520M CPU @ 2.5 GHz with 2GB RAM. The heuristic algorithms were coded in C++ with Microsoft Visual Studio 2013.

Testing Scenarios

The dataset from Ho and Szeto's (2014) research includes 52 testing scenarios encompassing a total of 13 different instances, which include a different number of stations, station penalty costs and station capacities. The number of stations begins at 100, increasing by 25 for each instances until reaching 400. There are two levels of time horizon, 9000 and 18000, and two levels of vehicle capacity, 10 and 20. The Table 5.1 lists the basic parameters for all of the testing scenarios. Based on the number of stations,

the time horizon and the vehicle capacity, these 52 testing scenarios were classified into 4 sets.

Table 5.1: Basic Parameters for the Instances in Ho and Szeto's (2014) Research

Parameters	Values
Station Number	U(100,400)
Time Horizon	{9000,18000}
Vehicle capacity	{10,20}

Testing Results

Because of the randomness embedded in H1, H2 and H3, the results for an algorithm may differ among the iterations even if the input parameters are the same. To reduce the randomness in the comparison, m=30 iterations were run for each algorithm in each testing scenario. After running these iterations, five measurement criteria were averaged from these m=30 iterations for each test scenario, the Minimal Objective Value (Min), the Maximal Objective Value (Max), the Average Objective Value (Avg), the Standard Deviation of the Objective Value for 30 iterations (Std) and the Average Solving Time (Time). These values were used to represent the performance of the algorithm in the testing scenarios. All testing results are shown in Tables 5.2~5.5. The $|N|$ column represents the number of stations used in the instance, for each testing scenario, the five measurement criteria (i.e. Min, Max, Avg, Std and Time) were collected for H1, H2 and H3 algorithms and the results are saved in the tables. The GAP2 and GAP3

columns show the improvement / decrease based on the results from the H1 algorithm, using the formula GAP2 = (H1 – H2) / H1, GAP3 = (H1 – H3) / H1.

Table 5.2: Test Cases with Time Horizon = 9000, and Vehicle Capacity = 10

$\|N\|$	H1					H2						H3					
	Max	Avg	Min	Std	Time (s)	Max	Avg	Min	Std	Time (s)	GAP2	Max	Avg	Min	Std	Time (s)	GAP3
100	749.6	752.9	754.6	1.32	153	650.3	656.8	657.2	1.96	2.08	12.8%	565.4	564.0	557.6	2.18	2.27	25.1%
125	1004.2	1004.4	1007.3	0.93	142	886.5	889.5	889.6	0.85	2.14	11.4%	786.0	781.7	770.3	4.73	3.22	22.2%
150	1222.9	1223.4	1229.7	1.94	169	1079.8	1084.7	1092.2	3.50	2.86	11.3%	965.8	964.0	951.9	3.54	3.97	21.2%
175	1372.6	1375.0	1378.9	1.74	184	1177.3	1192.4	1193.4	4.34	3.47	13.3%	1096.3	1094.9	1089.9	1.82	2.93	20.4%
200	1615.1	1616.7	1617.6	0.71	150	1217.9	1229.8	1242.4	7.72	3.70	23.9%	1306.5	1301.4	1298.6	2.29	3.64	19.5%
225	1874.1	1885.0	1890.2	4.97	192	1631.4	1635.2	1635.6	1.23	6.86	13.2%	1535.5	1532.8	1527.0	2.73	4.14	18.7%
250	2102.8	2112.8	2119.1	4.15	160	1896.7	1897.0	1898.8	0.66	5.18	10.2%	1693.9	1680.4	1675.9	4.66	5.85	20.5%
275	2236.8	2239.7	2243.4	1.89	195	1940.2	1946.4	1949.4	2.85	5.28	13.1%	1829.5	1826.4	1818.1	3.70	6.56	18.5%
300	2497.6	2507.7	2508.5	2.93	169	2149.2	2155.4	2163.3	4.09	6.77	14.0%	2036.6	2033.7	2026.4	2.61	6.02	18.9%
325	2740.3	2741.6	2742.8	0.95	202	2354.8	2361.3	2364.0	2.41	5.91	13.9%	2270.1	2253.9	2252.0	6.09	6.38	17.8%
350	2972.8	2973.9	2976.7	1.05	264	2673.7	2678.9	2683.1	2.79	6.46	9.9%	2576.0	2574.8	2570.8	1.70	6.83	13.4%
375	3118.9	3121.3	3126.7	1.89	258	2750.0	2752.6	2762.7	3.78	6.48	11.8%	2738.6	2727.7	2723.6	5.18	7.37	12.6%
400	3385.2	3385.8	3397.9	3.97	289	2823.5	2836.2	2849.8	6.94	7.41	16.2%	2737.1	2734.8	2725.5	3.32	7.55	19.2%

* The objective value does not include the transportation cost

Table 5.3: Test Cases with Time Horizon = 18000 and Vehicle Capacity = 10

$\|N\|$	H1					H2						H3					
	Max	Avg	Min	Std	Time (s)	Max	Avg	Min	Std	Time (s)	GAP2	Max	Avg	Min	Std	Time (s)	GAP3
100	680.9	666.6	684.1	0.63	331	587.4	589.6	602.4	4.85	2.85	11.6%	454.2	450.8	450.7	1.04	2.77	32.4%
125	920.5	891.1	923.6	1.72	304	790.5	790.6	791.6	0.35	3.10	11.3%	628.8	628.5	624.3	1.07	4.51	29.5%
150	1136.8	1116.4	1149.2	0.58	207	977.3	981.7	981.8	1.21	3.30	12.1%	767.1	759.1	759.0	2.16	4.86	32.0%
175	1283.5	1245.7	1287.4	5.05	317	1117.2	1123.8	1124.5	1.83	2.77	9.8%	925.3	922.2	909.2	3.99	5.59	26.0%
200	1511.9	1502.5	1524.3	1.33	285	1202.8	1203.0	1207.4	1.17	3.81	19.9%	1056.7	1055.7	1044.5	3.74	8.26	29.7%
225	1790.2	1732.4	1790.7	2.12	293	1535.6	1540.6	1541.9	2.12	4.07	11.1%	1278.5	1269.0	1266.9	2.45	9.39	26.7%
250	2014.3	1962.5	2020.6	2.48	306	1638.9	1641.5	1647.9	2.59	6.90	16.4%	1425.9	1419.3	1417.9	2.17	9.85	27.7%
275	2118.7	2151.0	2127.8	1.98	376	1848.5	1849.0	1859.6	3.09	7.53	14.0%	1541.7	1539.1	1537.8	0.92	9.88	28.4%
300	2403.4	2376.2	2408.7	4.72	325	1995.3	2010.3	2017.0	5.9	9.80	15.4%	1661.0	1652.9	1650.5	3.16	10.11	30.4%
325	2628.7	2616.6	2643.3	6.45	362	2234.1	2240.6	2246.3	3.8	10.42	14.4%	1835.3	1834.5	1820.5	4.45	12.45	29.9%
350	2864.1	2853.1	2868.3	0.20	343	2418.3	2434.2	2434.5	4.47	12.10	14.7%	2046.6	2034.7	2029.3	4.73	11.61	28.7%
375	2993.4	3028.5	3006.8	1.91	382	2576.2	2577.0	2583.0	1.72	12.73	14.9%	2151.4	2147.7	2147.2	1.27	12.69	29.1%
400	3262.3	3265.2	3270.8	4.02	421	2778.2	2789.3	2791.2	3.85	14.56	14.6%	2311.7	2305.7	2296.3	4.37	14.35	29.4%

* The objective value does not include the transportation cost

Table 5.4: Test Cases with Time Horizon = 9000, and Vehicle Capacity = 20

$\|N\|$	H1					H2						H3					
	Max	Avg	Min	Std	Time (s)	Max	Avg	Min	Std	Time (s)	GAP2	Max	Avg	Min	Std	Time (s)	GAP3
100	680.9	682.5	684.1	0.83	684	563.4	565.4	567.3	2.96	2.37	17.2%	536.9	534.5	534.4	0.67	2.34	21.7%
125	920.5	923.5	923.6	0.95	613	735.2	744.4	753.6	2.25	2.58	19.4%	719.0	715.8	713.6	1.66	3.62	22.5%
150	1136.8	1144.9	1149.2	3.35	723	884.9	885.1	886.3	2.2	3.47	22.7%	872.4	860.7	857.8	3.89	2.06	24.8%
175	1283.5	1285.5	1287.4	1	789	1048.6	1050.7	1051.5	2.79	3.97	18.3%	1051.5	1047.6	1041.7	2.95	4.67	18.5%
200	1511.9	1515.1	1524.3	2.95	898	1135.0	1144.8	1146.2	2.32	5.09	24.4%	1108.9	1106.8	1105.7	0.91	5.42	26.9%
225	1790.2	1790.2	1790.7	0.11	921	1468.8	1474.2	1487.8	1.78	5.12	17.7%	1422.1	1421.0	1406.1	4.25	5.78	20.6%
250	2014.3	2019.1	2020.6	1.58	969	1622.3	1634.4	1636.0	3.11	5.28	19.0%	1561.2	1560.6	1551.6	2.91	5.99	22.7%
275	2118.7	2127.1	2127.8	2.8	1123	1767.9	1775.0	1775.0	1.53	6.92	16.6%	1751.5	1743.5	1743.4	2.33	6.56	18.0%
300	2403.4	2406.8	2408.7	1.39	1266	1927.8	1928.1	1930.8	1.24	7.16	19.9%	1905.9	1902.7	1891.0	3.97	7.06	20.9%
325	2628.7	2632.9	2643.3	5.1	1607	2157.4	2164.0	2169.1	1.31	9.04	17.8%	2136.2	2135.7	2134.3	0.53	10.18	18.9%
350	2864.1	2866.4	2868.3	1.22	1772	2334.5	2353.9	2361.9	2.11	9.37	17.9%	2298.5	2298.3	2296.1	0.62	12.36	19.8%
375	2993.4	3005.7	3006.8	4.41	1764	2478.4	2480.3	2480.4	4.04	9.6	17.5%	2434.6	2430.2	2428.4	1.45	10.90	19.1%
400	3262.3	3280.2	3270.8	2.5	2215	2605.4	2617.7	2622.8	4.48	10.96	20.2%	2614.4	2614.2	2614.0	0.12	13.72	20.3%

* The objective value does not include the transportation cost

Table 5.5: Test Cases with Time Horizon = 18000, and Vehicle Capacity = 20

$\|N\|$	H1					H2						H3					
	Max	Avg	Min	Std	Time (s)	Max	Avg	Min	Std	Time (s)	GAP2	Max	Avg	Min	Std	Time (s)	GAP3
100	578.7	580.9	584.6	1.64	2029	500.3	501.6	512.2	3.7	3.10	13.6%	427.4	420.4	409.2	4.53	2.96	27.6%
125	790.0	790.3	795.3	1.57	1903	640.3	642.1	652.8	3.63	5.83	18.8%	569.1	565.1	562.4	1.86	3.08	28.5%
150	1000.8	1001.7	1003.1	0.73	2362	822.8	826.1	841.7	5.74	3.93	17.5%	695.1	690.5	682.8	3.22	5.76	31.1%
175	1115.4	1118.4	1119.3	1.17	2851	969.6	977.6	988.5	5.82	4.87	12.6%	832.1	829.8	828.4	0.85	5.98	25.8%
200	1361.4	1368.2	1373.4	3.08	2938	1139.0	1139.7	1139.8	0.27	6.60	16.7%	989.1	981.5	978.9	2.72	4.06	28.3%
225	1584.8	1585.3	1595.8	3.37	2186	1283.4	1284.1	1287.8	1.27	7.80	19.0%	1180.3	1173.2	1164.5	4.44	8.37	26.0%
250	1799.9	1801.9	1808.9	2.69	2293	1510.8	1517.4	1520.1	2.75	8.71	15.8%	1322.0	1310.9	1310.6	3.11	9.12	27.2%
275	1990.6	1991.3	1992.3	0.55	2153	1641.5	1641.6	1642.7	0.31	10.19	17.6%	1446.9	1439.6	1438.4	2.22	12.27	27.7%
300	2192.2	2207.1	2209.1	4.47	2694	1780.1	1788.4	1792.0	3.64	10.32	19.0%	1579.3	1572.6	1567.2	3.33	13.22	28.7%
325	2427.5	2431.7	2431.7	1.27	3270	1837.8	1861.6	1866.1	8.23	11.46	23.4%	1753.8	1753.5	1742.8	3.11	16.01	27.9%
350	2677.7	2678.8	2681.6	1.17	3565	2132.7	2134.4	2135.9	1.04	15.08	20.3%	1879.1	1878.1	1866.0	3.30	16.31	29.9%
375	2849.8	2855.0	2856.4	1.79	3503	2138.5	2152.2	2154.8	4.61	15.13	24.6%	2027.7	2016.7	2010.6	4.94	16.34	29.4%
400	3071.3	3074.3	3085.6	3.91	3009	2315.8	2323.5	2330.9	4.32	16.14	24.4%	2195.4	2192.9	2190.6	1.46	19.72	28.7%

* The objective value does not include the transportation cost

A quick reading of the results in the tables indicates that the H3 algorithm usually had the smallest objective value and the H1 the largest objective value. The objective value for the same testing scenario usually followed the pattern of H1 > H2 > H3, meaning that the H3 algorithm provides the best quality solution of the three. Since it is difficult to check the differences among the three algorithms by manually reading the values in the tables, a statistical tool was used to help find them.

Similar to previous testing, first the Anderson-Darling test was used to check the normality of the raw data. The test results are shown in Appendix A, B and C. The Figures A.1~A.4, B.1~B.4, C.1~C.4 indicate that the average objective values for H1, H2 and H3 do not follow a normal distribution, meaning nonparametric statistical tests were required.

This analysis used the Freidman test, a nonparametric statistical tool similar to a two-way ANOVA, to explore these observations. The three algorithms served as the treatment and the testing scenarios as the blocks. In general, the Freidman test ranks the average objective values from each algorithm for each testing scenario, with the algorithm with the lowest (best) value being assigned rank 1, the second best rank 2, and so on until all are ranked. In the case of a tie, average ranks are assigned. For example, if 2 algorithms are tied for rank 1, they are both ranked 1.5 and next rank is 3. The hypotheses for Freidman test are listed below:

H_0: the median of the average objective values is equal for all algorithms.

H_1: not all medians of the average objective values for all algorithms are equal.

Appendix Tables C.1, C.2, C.3 and C.4 present the statistical results from the Freidman tests obtained using MINITAB. All tests results provide very small *p*-values (<0.001), meaning that there is sufficient evidence to reject the null hypothesis and conclude that not all medians of the average values of all algorithms are equal. Figure 5.1 shows the sums of the ranks for all three algorithms for all 52 testing scenarios. The H3 algorithm has a sum rank of 53 out of 52 testing scenarios, meaning that the H3 algorithm was ranked second once and ranked first 51 times for all 52 testing scenarios, results suggesting that in general it performed than the H2 and H1 algorithms across all of these testing scenarios. The H1 algorithm has a sum rank of 156, indicating that it ranked third for all 52 testing scenarios. Based on the results from the Freidman test and the post-hoc analysis, it can be concluded that the performance of three algorithms follows the pattern H3 > H2 > H1.

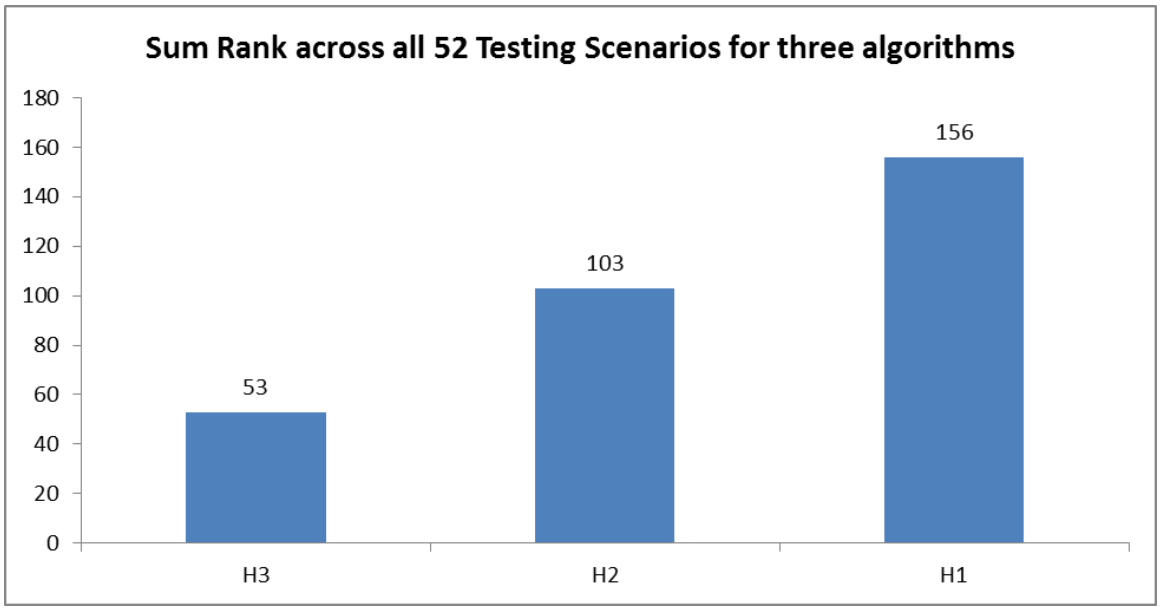

Figure 5.1: Three algorithms Sum Rank for all 52 Testing Scenario in Freidman Test

Similar to the analysis of the H2 algorithm in Chapter 4, the H3 algorithm performed better because of the relaxation of the visit limitation in the SBRP problem. The H1 algorithm is fully restricted by the visit limitation as each station can be visited at most once in each solution. The H2 algorithm partially relaxed this visit limitation as the same station could be visited multiple times in different trips. With this relaxation, the station with a large demand / inventory can be visited multiple times to have its requests fulfilled. However, this algorithm uses multiple trips to fulfill the repositioning, meaning this routing schedule allowed the vehicle to visit the depot multiple times. Because the depot maintains infinite locker capacity and bicycle inventory capacity, it creates the opportunity to reduce the penalty cost by visiting it to unload / pick up extra bicycles when the station is close to the depot. The H3 algorithm removed the visit limitation completely, allowing any vehicle schedule for the repositioning event. Thus, it further helps to reduce the transportation time by provide more selection options for determining the repositioning routing schedule. This improvement from H1 to H2 and finally to the H3 algorithm was not only caused by the improvement in the algorithm but also by the relaxing of constraint of the research problem. The solution for the H3 algorithm is more closely related to a real-world scenario, giving the highest quality solution among the 3 algorithms. Furthermore, comparing the solving time for the H2 and H3 algorithms indicates that both provide solutions within a short time period, meaning in general, the H3 algorithm provides the best quality solution of the three algorithms in a short solving time.

Conclusion and Future Work

We proposed a VNS heuristic to solve the static bicycle repositioning problem with a single vehicle, multiple trips and no station visit limitation. In this research, we use multiple trips to fulfill the repositioning event and fully relaxed the station visit limitation, meaning that the vehicle can use any schedule to fulfill the repositioning event. These assumptions make this research is similar to real-world situations.

The experimental results using the instances from Ho and Szeto's (2014) research indicate the new heuristic algorithm H3 provides the best quality solution within a short solving time compared to the other 2 algorithms (H1 and H2). The H3 algorithm provides a better solution than the H1 algorithm with an average improvement of 0.45%, and a 0.22% improvement over the H2 algorithm. This improvement is caused by its structure of the heuristic algorithm and the relaxation of the visit limitation constraint.

In the future, this research will extend the current SBRP problem with single vehicle into multiple vehicles scenarios. In addition, currently we consider only a single type of the bicycle at the station. However, in the real-world, multiple types of bicycles such as 2-man bicycles or 3-man bicycles could be located at one station. The SBRP problem could be extended to include these various types of bicycles.

CHAPTER SIX

CONCLUSION AND FUTURE WORK

In this research, we studied the static bicycle repositioning problem with a single vehicle. We first focused on the very basic SBRP problem with a single vehicle; the first study fully implemented the station visit limitation which only allowed the station to be visited once in the solution, and we proposed a 2-step algorithm to solve the problem. A new auxiliary method was developed to solve the 2nd step optimally by a given routing schedule. Then we partially relaxed the station visit limitation and use multiple trips to fulfill the reposition event. A new heuristic is constructed by using the 1-step algorithm to modify the routing schedule and assignment plan at the same time. The third study fully relaxed the station visit limitation and allowed the vehicle to use any schedule to complete the repositioning event. Also, a new heuristic was proposed to solve the problem with this scenario.

As we can see our studies try to relax the SBRP model constraints and make it more similar with the real world scenario. The numeric experiments indicate our algorithm can provide a good solution for the SBRP problem. The solving time for the model is also short. We could use the result of this research to provide the SBRP problem with up to 400 station nodes a good quality result within 15 seconds.

In the future, we want to extend our research in several aspects: (1) since we already create a near real world scenario model for the SBRP problem with a single vehicle, we want to extend this model into multiple vehicles scenarios. In this scenario, vehicles with identical or different capacities would be considered. For multiple vehicles,

the partition of different areas will be a good idea. For vehicles with different capacities, how to reduce the waste of transportation capacity could be an important topic.

(2) A study of design station locations in order to reduce rebalancing is also an interesting aspect. With analysis of historical data of bicycle trends, changing the price charged if the customer returns a bicycle to a less preferred location is a kind of intentional guide to let customers balance the bicycle quantities without the company engaging in repositioning. This kind of price leverage will help reduce the overall cost of repositioning.

(3) Currently we only consider a single type of bicycle in the station. But in the real world, there could be multiple different types of bicycles in the same station, such as 2 man bicycle or various qualities of bicycles. We could extend our SBRP problem with multiple types of bicycles. In part due to the reasonable solving time, the results of chapter five can be extended to the dynamic bicycle repositioning problem as well.

(4) Dynamic bicycle sharing system is another popular topic and it reflects the real world scenario. Based on price leverage and historical bicycle trends data, we may develop a simulation model to predict the vacancy rate of stations and encourage customers to return bicycles to the empty stations, which will be helpful to minimize reposition cost and reposition time interval.

(5) This paper used a routing first and assignment second sequence, while, in future research, an assignment first and routing second could be considered to see if it can approach better results. In that case, Fisher and Kaikumar algorithm, the Petal algorithm, the Sweep algorithm and the Taillard algorithm should be compared to find out which one is a better solution for bicycle sharing problem.

APPENDICES

Appendix A
Statistical Results for Chapter 3 Numerical Experiment

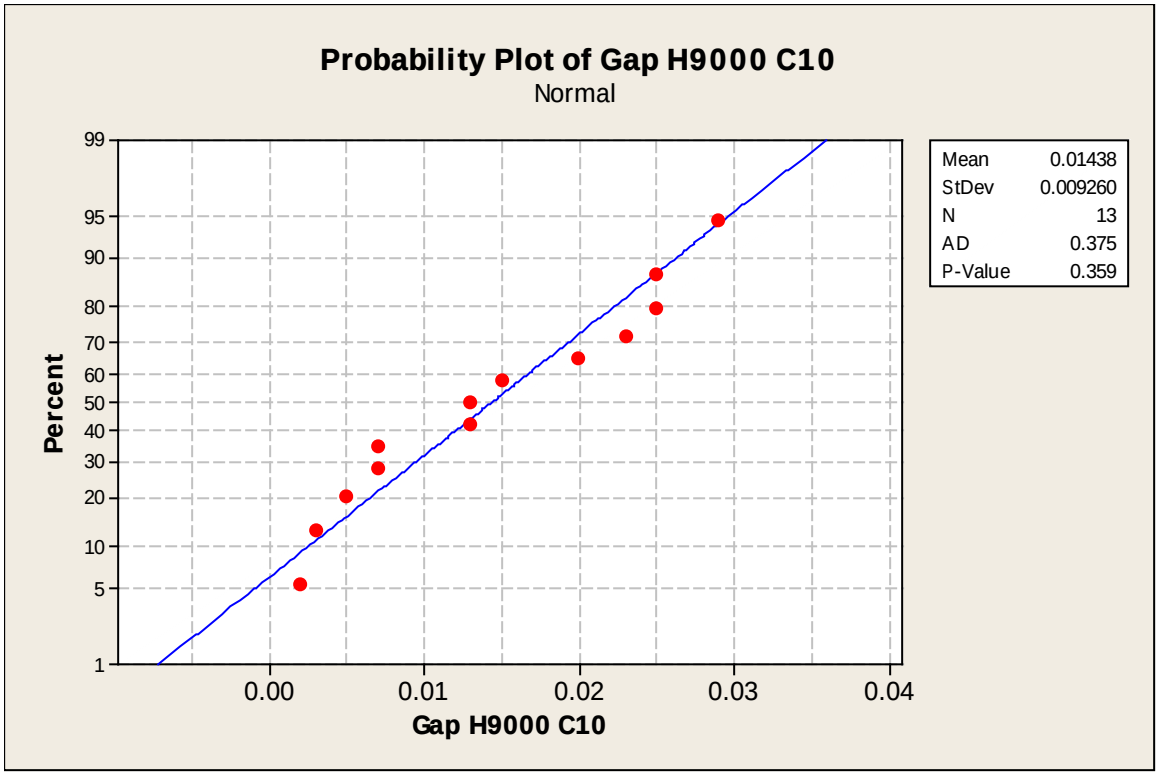

Figure A.1: Normality test for GAP, time horizon = 9000, capacity = 10

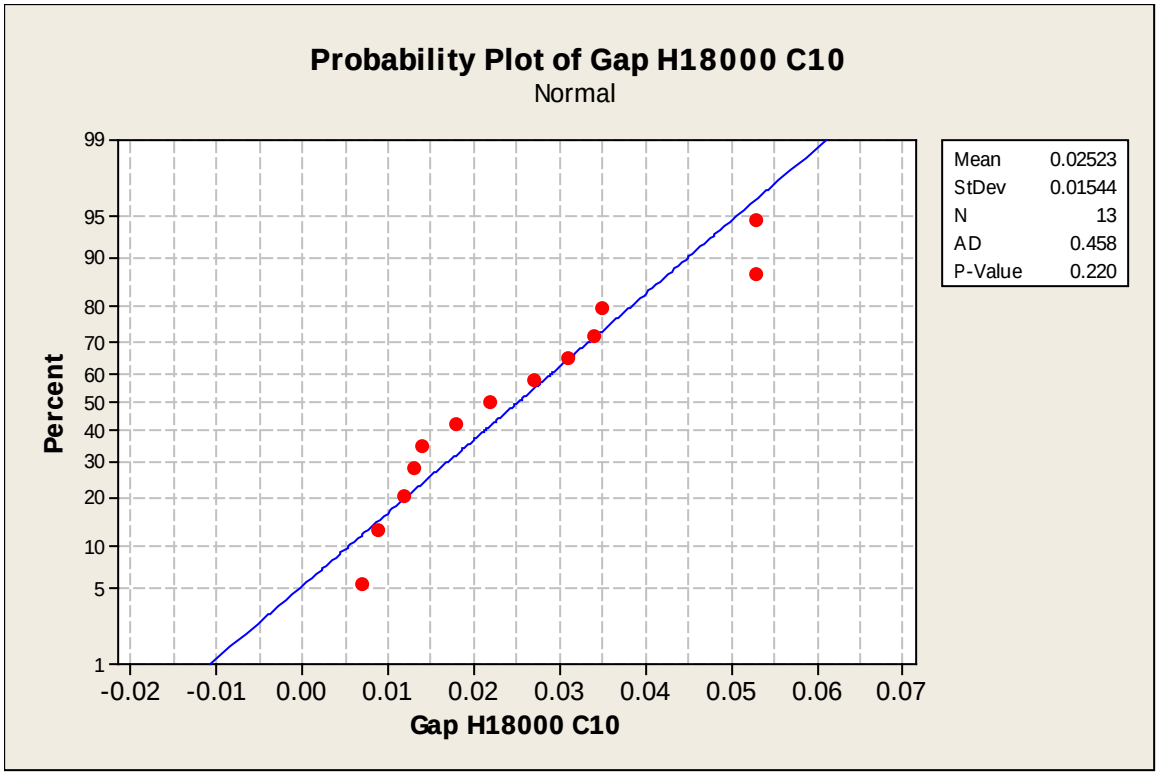

Figure A.2: Normality test for GAP, time horizon = 18000, capacity = 10

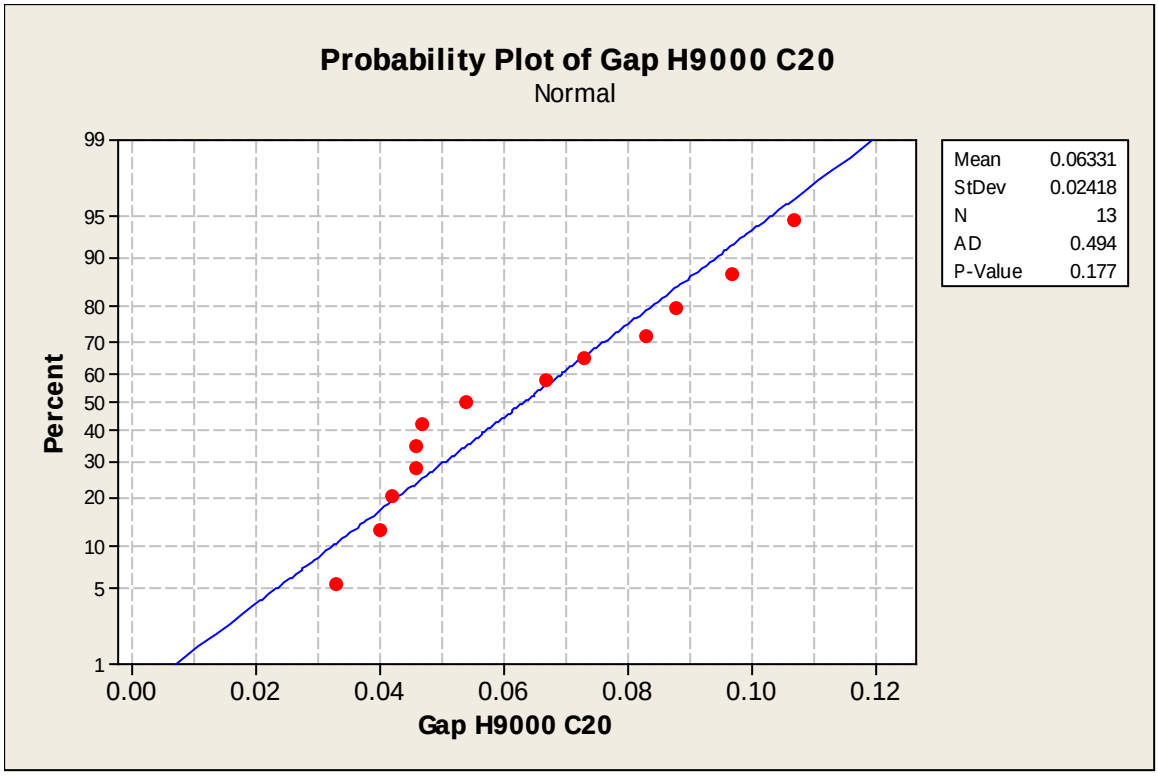

Figure A.3: Normality test for GAP, time horizon = 9000, capacity = 20

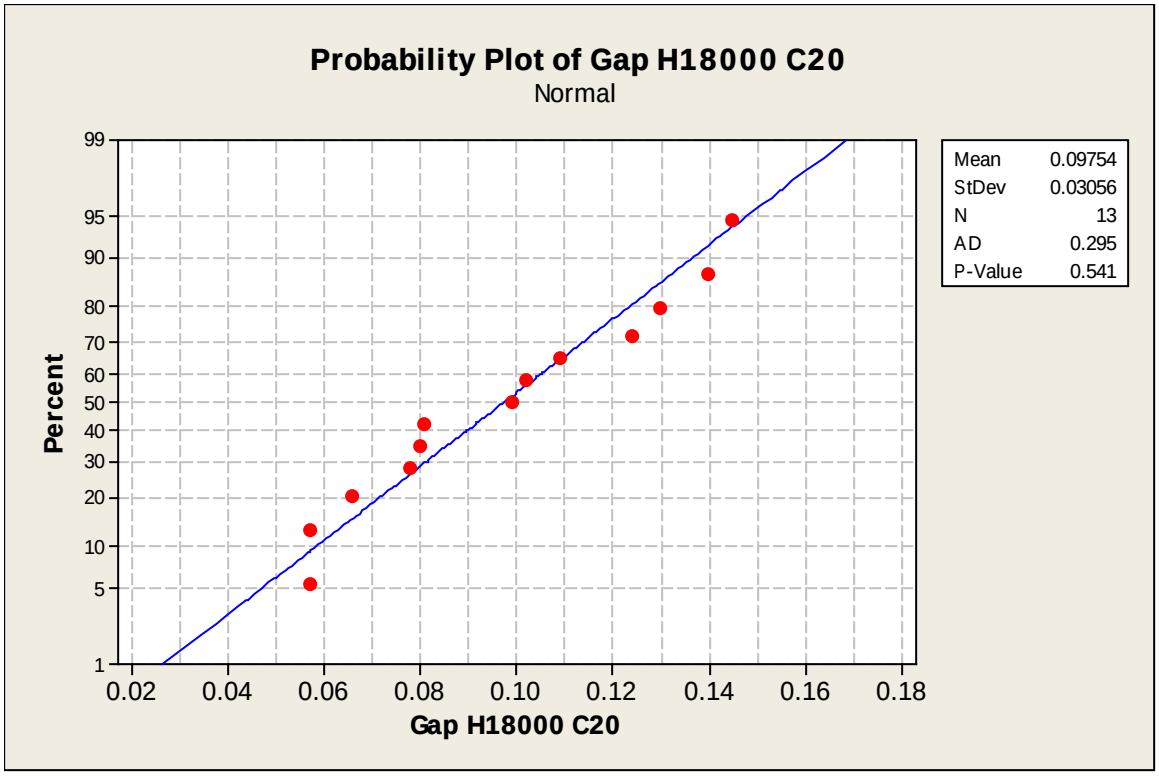

Figure A.4: Normality test for GAP, time horizon = 18000, capacity = 20

Table A.1: Wilcoxon Signed Rank Test for GAP, time horizon=9000, capacity=10

Wilcoxon Signed Rank Test: Gap H=9000 C=10

```
Test of median = 0.000000 versus median > 0.000000

         N for   Wilcoxon         Estimated
      N   Test  Statistic      P     Median
Gap  13     13       91.0  0.001    0.01400
```

Table A.2: Wilcoxon Signed Rank Test for GAP, time horizon=18000, capacity=10

Wilcoxon Signed Rank Test: Gap H=18000 C=10

```
Test of median = 0.000000 versus median > 0.000000

         N for   Wilcoxon         Estimated
      N   Test  Statistic      P     Median
Gap  13     13       91.0  0.001    0.02350
```

Table A.3: Wilcoxon Signed Rank Test for GAP, time horizon=9000, capacity=20

Wilcoxon Signed Rank Test: Gap H=9000 C=20

```
Test of median = 0.000000 versus median > 0.000000

         N for   Wilcoxon         Estimated
      N   Test  Statistic      P     Median
Gap  13     13       91.0  0.001    0.06350
```

Table A.4: Wilcoxon Signed Rank Test for GAP, time horizon=18000, capacity=20

Wilcoxon Signed Rank Test: Gap H=18000 C=20

```
Test of median = 0.000000 versus median > 0.000000

         N for   Wilcoxon         Estimated
      N   Test  Statistic      P     Median
Gap  13     13       91.0  0.001    0.09850
```

Appendix B
Statistical Results for Chapter 4 Numerical Experiment

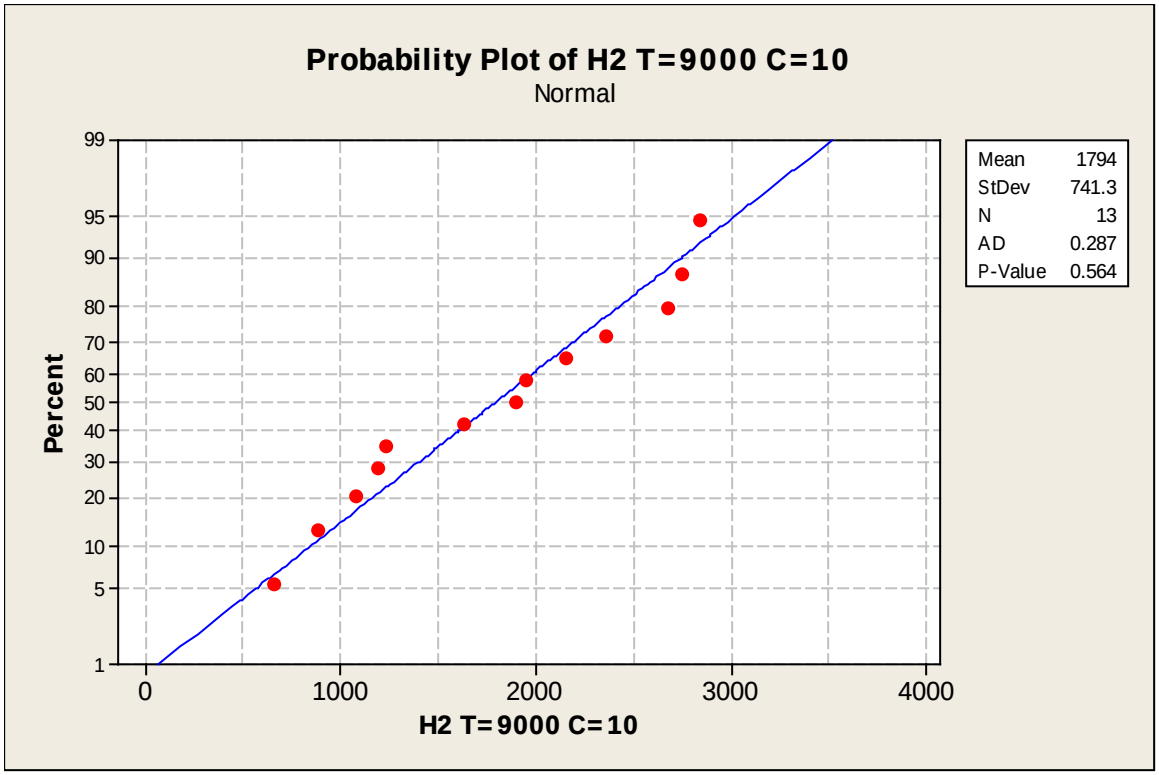

Figure B.1: Normality test for H2 AvgObj, time horizon = 9000, capacity = 10

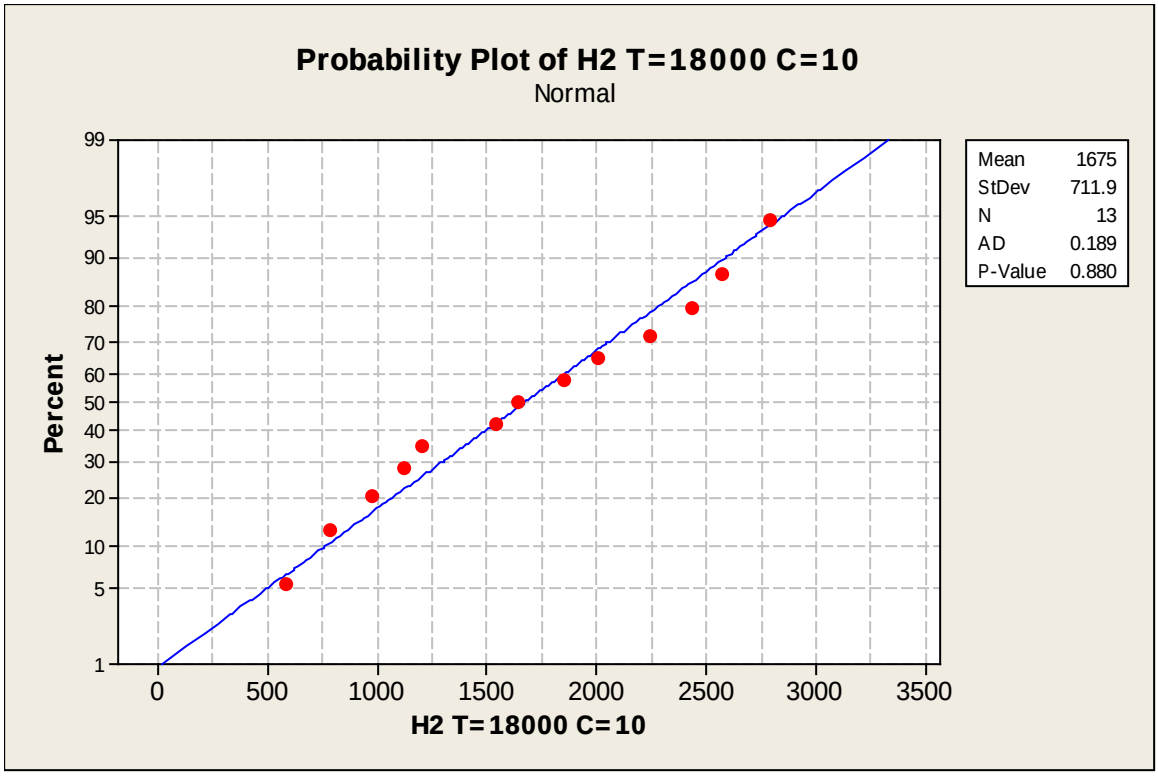

Figure B.2: Normality test for H2 AvgObj, time horizon = 18000, capacity = 10

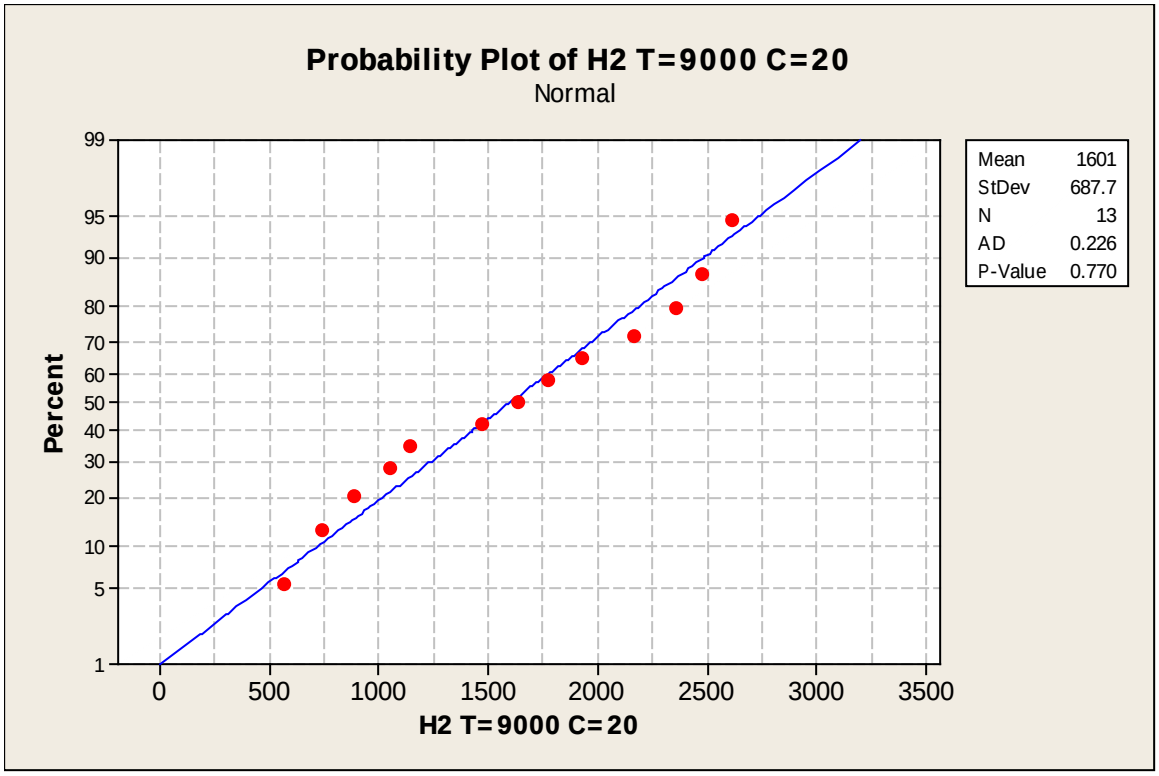

Figure B.3: Normality test for H2 AvgObj, time horizon = 9000, capacity = 20

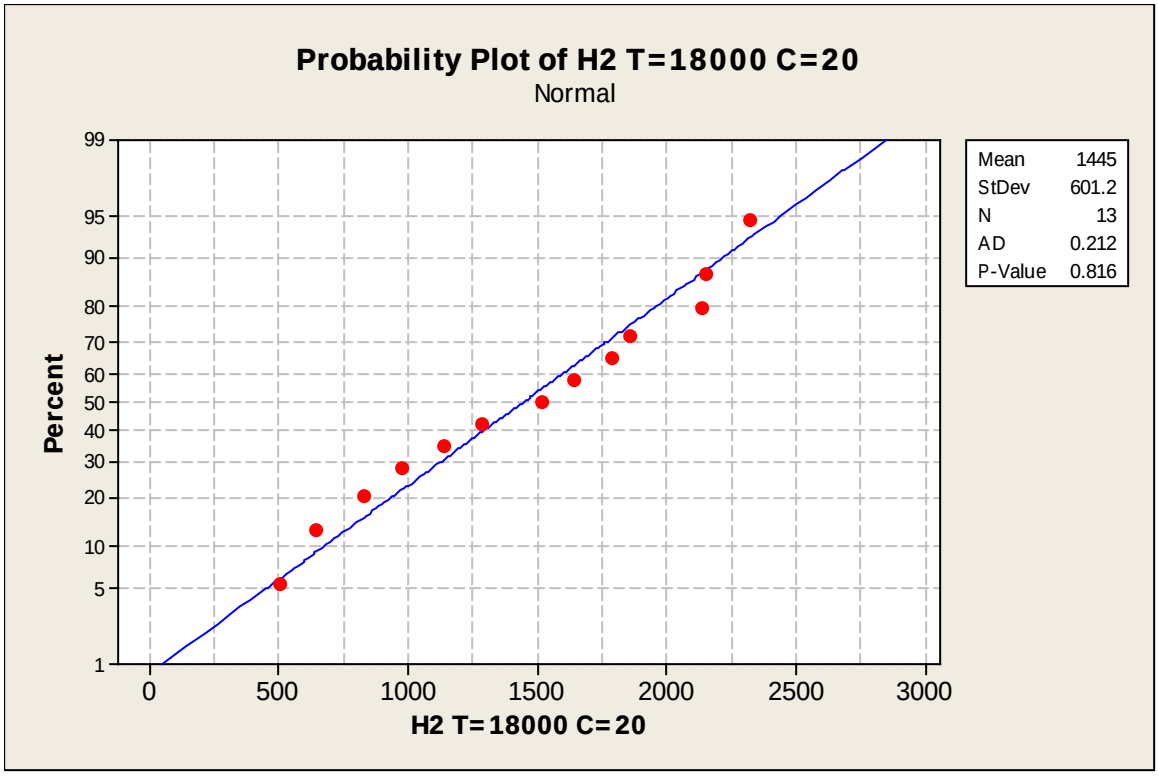

Figure B.4: Normality test for H2 AvgObj, time horizon = 18000, capacity = 20

Table B.1: Friedman Test result, objective vs Algorithm blocked by Testing scenarios, time horizon = 9000, capacity = 10

Friedman Test: Objective versus Algorithm blocked by Testing Scenario

```
S = 13.00  DF = 1  P = 0.000

                            Sum of
Algorithm   N  Est Median   Ranks
H1         13      2151.6    26.0
H2         13      1858.3    13.0
```

Table B.2: Friedman Test result, objective vs Algorithm blocked by Testing scenarios, time horizon = 18000, capacity = 10

Friedman Test: Objective versus Algorithm blocked by Testing Scenario

```
S = 13.00  DF = 1  P = 0.000

                            Sum of
Algorithm   N  Est Median   Ranks
H1         13      1953.0    26.0
H2         13      1651.0    13.0

Grand median = 1802.0
```

Table B.3: Friedman Test result, objective vs Algorithm blocked by Testing scenarios, time horizon = 9000, capacity = 20

Friedman Test: Objective versus Algorithm blocked by Testing Scenario

```
S = 13.00  DF = 1  P = 0.000

                            Sum of
Algorithm   N  Est Median   Ranks
H1         13      2011.9    26.0
H2         13      1641.6    13.0

Grand median = 1826.8
```

Table B.4: Friedman Test result, objective vs Algorithm blocked by Testing scenarios, time horizon = 18000, capacity = 20

Friedman Test: Objective versus Algorithm blocked by Testing Scenario

```
S = 13.00  DF = 1  P = 0.000

                            Sum of
Algorithm   N  Est Median   Ranks
H1         13      1810.3    26.0
H2         13      1509.1    13.0

Grand median = 1659.7
```

Appendix C
Statistical Results for Chapter 5 Numerical Experiment

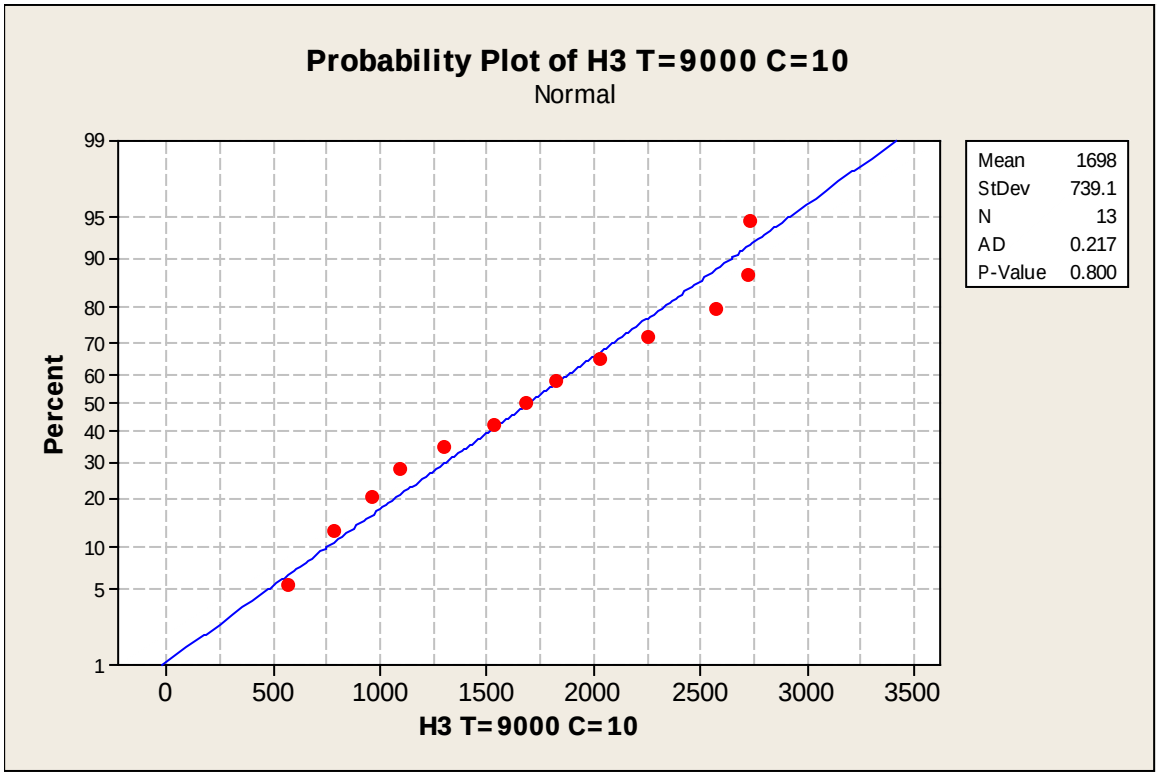

Figure C.1: Normality test for H3 AvgObj, time horizon = 9000, capacity = 10

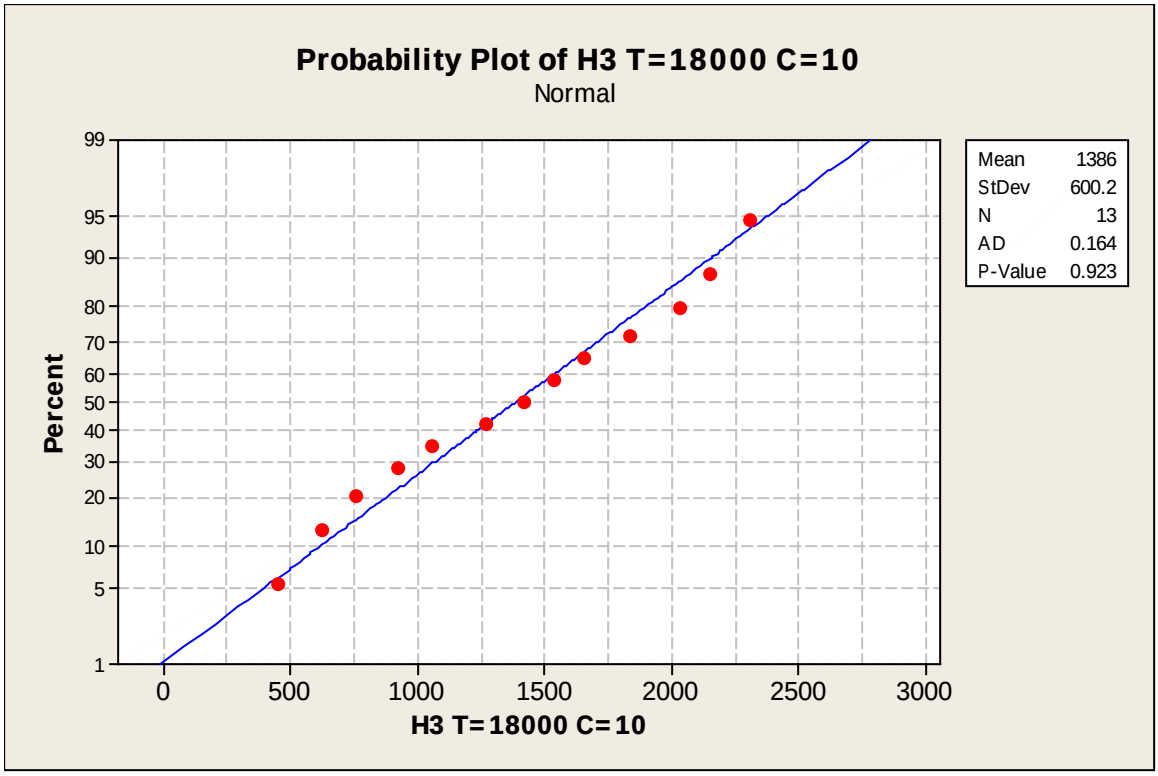

Figure C.2: Normality test for H3 AvgObj, time horizon = 18000, capacity = 10

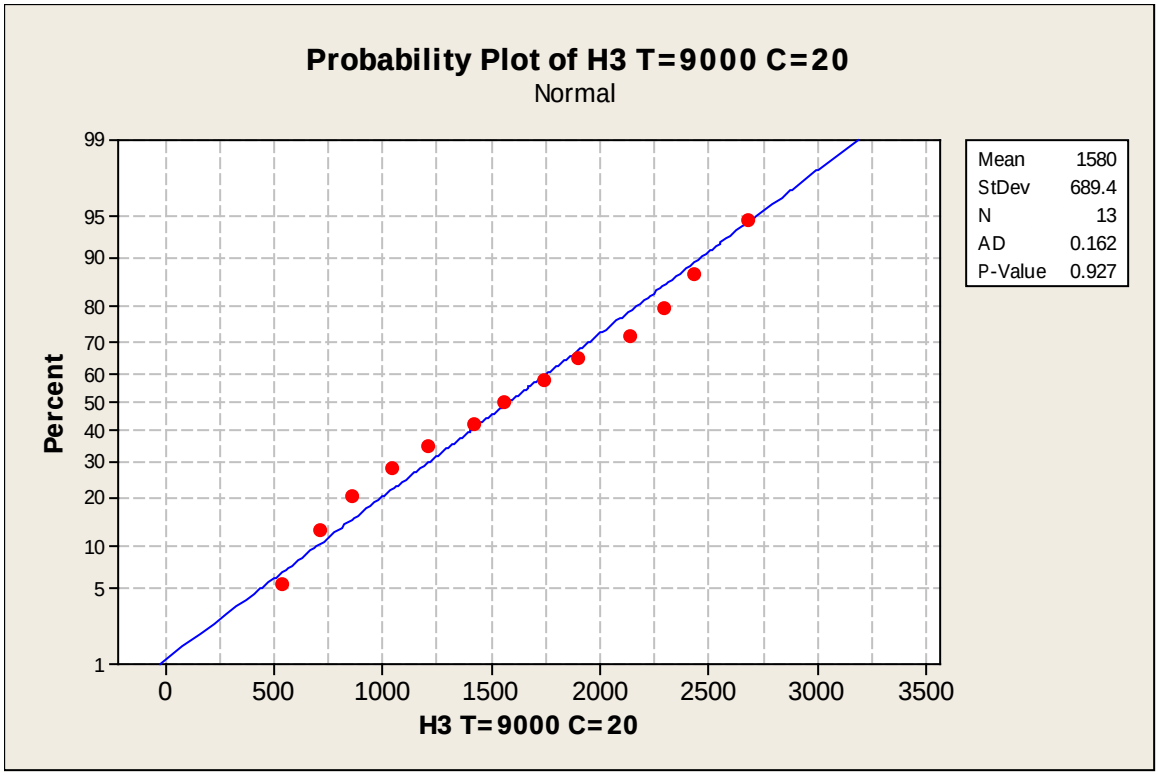

Figure C.3: Normality test for H3 AvgObj, time horizon = 9000, capacity = 20

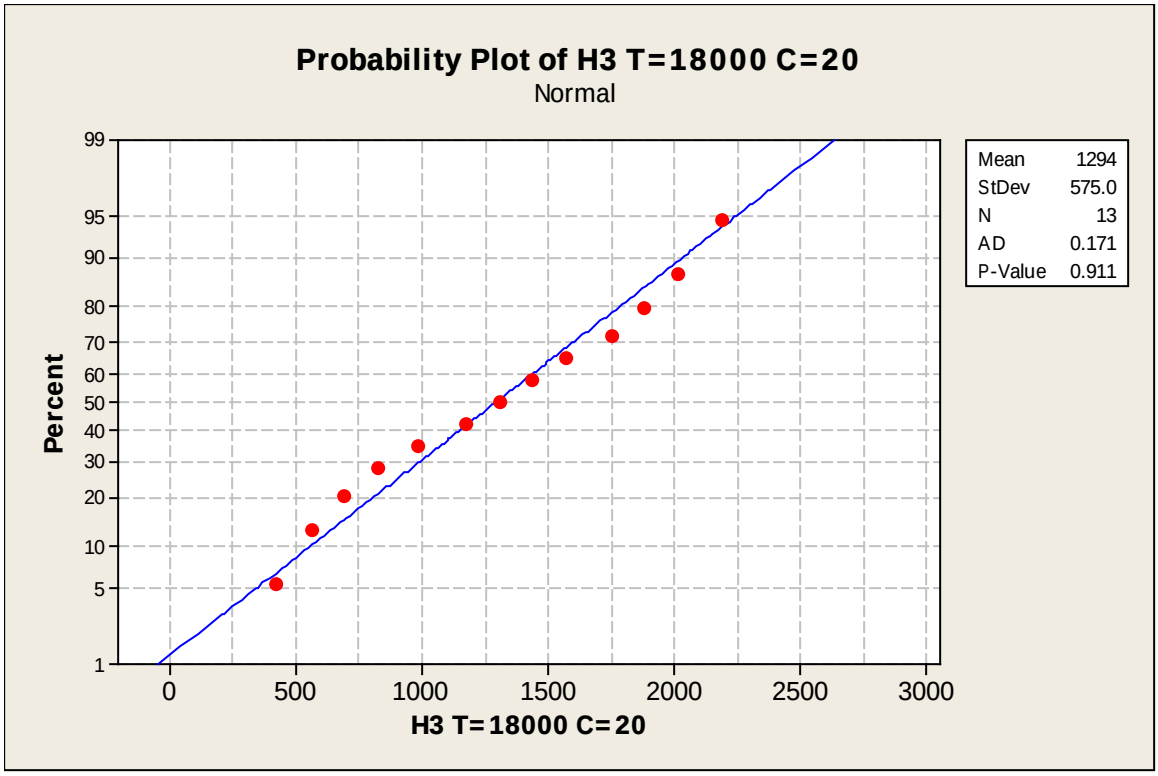

Figure C.4: Normality test for H3 AvgObj, time horizon = 18000, capacity = 10

Table C.1: Friedman Test result, objective vs Algorithm blocked by Testing scenarios, time horizon = 9000, capacity = 10

Friedman Test: Objective versus Algorithm blocked by Testing Scenario

```
S = 24.15  DF = 2  P = 0.000

                            Sum of
Algorithm   N  Est Median   Ranks
H1         13      2112.8    39.0
H2         13      1820.8    25.0
H3         13      1718.0    14.0

Grand median = 1883.8
```

Table C.2: Friedman Test result, objective vs Algorithm blocked by Testing scenarios, time horizon = 18000, capacity = 10

Friedman Test: Objective versus Algorithm blocked by Testing Scenario

```
S = 26.00  DF = 2  P = 0.000

                            Sum of
Algorithm   N  Est Median   Ranks
H1         13      1962.5    39.0
H2         13      1670.7    26.0
H3         13      1409.2    13.0

Grand median = 1680.8
```

Table C.3: Friedman Test result, objective vs Algorithm blocked by Testing scenarios, time horizon = 9000, capacity = 20

Friedman Test: Objective versus Algorithm blocked by Testing Scenario

```
S = 26.00  DF = 2  P = 0.000

                            Sum of
Algorithm   N  Est Median   Ranks
H1         13      2007.1    39.0
H2         13      1634.4    26.0
H3         13      1601.2    13.0

Grand median = 1747.6
```

Table C.4: Friedman Test result, objective vs Algorithm blocked by Testing scenarios, time horizon = 18000, capacity = 20

Friedman Test: Objective versus Algorithm blocked by Testing Scenario

```
S = 26.00  DF = 2  P = 0.000

                            Sum of
Algorithm   N  Est Median   Ranks
H1         13      1801.9    39.0
H2         13      1482.6    26.0
H3         13      1329.0    13.0

Grand median = 1537.8
```

REFERENCE

Alvarez-Valdes, R., Belenguer, J. M., Benavent, E., Bermudez, J. D., Muñoz, F., Vercher, E., & Verdejo, F. (2016). Optimizing the level of service quality of a bike-sharing system. *Omega, 62*, 163-175.

Angeloudis, P., Hu, J., & Bell, M. G. (2014). A strategic repositioning algorithm for bicycle-sharing schemes. *Transportmetrica A: Transport Science, 10*(8), 759-774.

Anily, S., & Hassin, R. (1992). The swapping problem. *Networks, 22*(4), 419-433.

Benchimol, M., Benchimol, P., Chappert, B., De La Taille, A., Laroche, F., Meunier, F., & Robinet, L. (2011). Balancing the stations of a self service "bike hire" system. *RAIRO-Operations Research, 45*(1), 37-61.

Berbeglia, G., Cordeau, J.-F., Gribkovskaia, I., & Laporte, G. (2007). Static pickup and delivery problems: a classification scheme and survey. *Top, 15*(1), 1-31.

Boarnet, M. G., Chester, M., Joh, K., Fulton, W., Guzman, S., Handy, S. L., . . . Siembab, W. (2011). ACCESS Magazine Fall 2011. *ACCESS Magazine, 1*(39).

Brinkmann, J., Ulmer, M. W., & Mattfeld, D. C. (2015). Inventory Routing for Bikes Sharing Systems: Working Paper (2015-01-12).

Caggiani, L., & Ottomanelli, M. (2013). A dynamic simulation based model for optimal fleet repositioning in bike-sharing systems. *Procedia-Social and Behavioral Sciences, 87*, 203-210.

Chajakis, E. D., & Guignard, M. (2003). Scheduling deliveries in vehicles with multiple compartments. *Journal of Global Optimization, 26*(1), 43-78.

Chalasani, P., & Motwani, R. (1999). Approximating capacitated routing and delivery problems. *SIAM Journal on Computing, 28*(6), 2133-2149.

Chemla, D., Meunier, F., & Calvo, R. W. (2013). Bike sharing systems: Solving the static rebalancing problem. *Discrete Optimization, 10*(2), 120-146.

Chemla, D., Meunier, F., Pradeau, T., Calvo, R. W., & Yahiaoui, H. (2013). Self-service bike sharing systems: simulation, repositioning, pricing.

Cherkesly, M., Desaulniers, G., & Laporte, G. (2015). A population-based metaheuristic for the pickup and delivery problem with time windows and LIFO loading. *Computers & Operations Research, 62*, 23-35.

Christiansen, M., Fagerholt, K., Flatberg, T., Haugen, Ø., Kloster, O., & Lund, E. H. (2011). Maritime inventory routing with multiple products: A case study from the cement industry. *European Journal of Operational Research, 208*(1), 86-94.

Contardo, C., Morency, C., & Rousseau, L.-M. (2012). *Balancing a dynamic public bike-sharing system* (Vol. 4): Cirrelt.

Cornillier, F., Boctor, F., & Renaud, J. (2012). Heuristics for the multi-depot petrol station replenishment problem with time windows. *European Journal of Operational Research, 220*(2), 361-369.

Dell'Amico, M., Hadjicostantinou, E., Iori, M., & Novellani, S. (2014). The bike sharing rebalancing problem: Mathematical formulations and benchmark instances. *Omega, 45*, 7-19.

Dumitrescu, I., Ropke, S., Cordeau, J.-F., & Laporte, G. (2010). The traveling salesman problem with pickup and delivery: polyhedral results and a branch-and-cut algorithm. *Mathematical Programming, 121*(2), 269-305.

Erdoğan, G., Cordeau, J.-F., & Laporte, G. (2010). A branch-and-cut algorithm for solving the non-preemptive capacitated swapping problem. *Discrete Applied Mathematics, 158*(15), 1599-1614.

Erdoğan, G., Laporte, G., & Calvo, R. W. (2012). The one-commodity pickup and delivery traveling salesman problem with demand intervals: Working paper.

Erdoğan, G., Laporte, G., & Calvo, R. W. (2014). The static bicycle relocation problem with demand intervals. *European Journal of Operational Research, 238*(2), 451-457.

Forma, I. A., Raviv, T., & Tzur, M. (2015). A 3-step math heuristic for the static repositioning problem in bike-sharing systems. *Transportation research part B: methodological, 71*, 230-247.

Hemmelmayr, V. C., Doerner, K. F., & Hartl, R. F. (2009). A variable neighborhood search heuristic for periodic routing problems. *European Journal of Operational Research, 195*(3), 791-802.

Hernández-Pérez, H., Rodríguez-Martín, I., & Salazar-González, J.-J. (2016). A hybrid heuristic approach for the multi-commodity pickup-and-delivery traveling salesman problem. *European Journal of Operational Research, 251*(1), 44-52.

Hernández-Pérez, H., & Salazar-González, J.-J. (2003). The one-commodity pickup-and-delivery travelling salesman problem *Combinatorial Optimization—Eureka, You Shrink!* (pp. 89-104): Springer

Hernández-Pérez, H., & Salazar-González, J.-J. (2004a). A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics, 145*(1), 126-139.

Hernández-Pérez, H., & Salazar-González, J.-J. (2004b). Heuristics for the one-commodity pickup-and-delivery traveling salesman problem. *Transportation Science, 38*(2), 245-255.

Hernández-Pérez, H., & Salazar-González, J.-J. (2009). The multi-commodity one-to-one pickup-and-delivery traveling salesman problem. *European Journal of Operational Research, 196*(3), 987-995.

Hernández-Pérez, H., & Salazar-González, J. J. (2007). The one-commodity pickup-and-delivery traveling salesman problem: Inequalities and algorithms. *Networks, 50*(4), 258-272.

Ho, S. C., & Szeto, W. (2014). Solving a static repositioning problem in bike-sharing systems using iterated tabu search. *Transportation Research Part E: Logistics and Transportation Review, 69*, 180-198.

Ho, S. C., & Szeto, W. (2016). GRASP with path relinking for the selective pickup and delivery problem. *Expert Systems With Applications, 51*, 14-25.

John, H. (1992). Holland, Adaptation in natural and artificial systems: MIT Press, Cambridge, MA.

Kloimüllner, C., Papazek, P., Hu, B., & Raidl, G. R. (2014). *Balancing bicycle sharing systems: an approach for the dynamic case.* Paper presented at the European Conference on Evolutionary Computation in Combinatorial Optimization.

Kuo, Y., & Wang, C.-C. (2012). A variable neighborhood search for the multi-depot vehicle routing problem with loading cost. *Expert Systems with Applications, 39*(8), 6949-6954.

Lahyani, R., Coelho, L. C., Khemakhem, M., Laporte, G., & Semet, F. (2015). A multi-compartment vehicle routing problem arising in the collection of olive oil in Tunisia. *Omega, 51*, 1-10.

Larsen, J. (2013). Bike-sharing programs hit the streets in over 500 cities worldwide. *Earth Policy Institute, 25*, 1.

Li, Y., Szeto, W., Long, J., & Shui, C. (2016). A multiple type bike repositioning problem. *Transportation Research Part B: Methodological, 90*, 263-278.

Lin, J.-H., & Chou, T.-C. (2012). A geo-aware and VRP-based public bicycle redistribution system. *International Journal of Vehicular Technology, 2012*.

Mahmoudi, M., & Zhou, X. (2016). Finding optimal solutions for vehicle routing problem with pickup and delivery services with time windows: A dynamic programming approach based on state–space–time network representations. *Transportation Research Part B: Methodological, 89*, 19-42.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research, 24*(11), 1097-1100.

Muyldermans, L., & Pang, G. (2010). On the benefits of co-collection: Experiments with a multi-compartment vehicle routing algorithm. *European Journal of Operational Research, 206*(1), 93-103.

Nair, R., Miller-Hooks, E., Hampshire, R. C., & Bušić, A. (2013). Large-scale vehicle sharing systems: analysis of Vélib'. *International Journal of Sustainable Transportation, 7*(1), 85-106.

Papazek, P., Kloimüllner, C., Hu, B., & Raidl, G. R. (2014). *Balancing bicycle sharing systems: an analysis of path relinking and recombination within a GRASP hybrid.* Paper presented at the International Conference on Parallel Problem Solving from Nature.

Papazek, P., Raidl, G. R., Rainer-Harbach, M., & Hu, B. (2013). *A PILOT/VND/GRASP hybrid for the static balancing of public bicycle sharing systems.* Paper presented at the International Conference on Computer Aided Systems Theory.

Parragh, S. N., Doerner, K. F., & Hartl, R. F. (2008). A survey on pickup and delivery problems. *Journal für Betriebswirtschaft, 58*(1), 21-51.

Pfrommer, J., Warrington, J., Schildbach, G., & Morari, M. (2014). Dynamic vehicle redistribution and online price incentives in shared mobility systems. *IEEE Transactions on Intelligent Transportation Systems, 15*(4), 1567-1578.

Pirkwieser, S., & Raidl, G. R. (2008). *A variable neighborhood search for the periodic vehicle routing problem with time windows.* Paper presented at the Proceedings of the 9th EU/meeting on metaheuristics for logistics and vehicle routing, Troyes, France.

Pirkwieser, S., & Raidl, G. R. (2009). *Multiple variable neighborhood search enriched with ILP techniques for the periodic vehicle routing problem with time windows.* Paper presented at the International Workshop on Hybrid Metaheuristics.

Pirkwieser, S., & Raidl, G. R. (2010). *Variable neighborhood search coupled with ILP-based very large neighborhood searches for the (periodic) location-routing problem.* Paper presented at the International Workshop on Hybrid Metaheuristics.

Polacek, M., Benkner, S., Doerner, K. F., & Hartl, R. F. (2008). A cooperative and adaptive variable neighborhood search for the multi depot vehicle routing problem with time windows. *BuR-Business Research, 1*(2), 207-218.

Polacek, M., Hartl, R. F., Doerner, K., & Reimann, M. (2004). A variable neighborhood search for the multi depot vehicle routing problem with time windows. *Journal of heuristics, 10*(6), 613-627.

Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research, 31*(12), 1985-2002.

Psaraftis, H. N. (2011). A multi-commodity, capacitated pickup and delivery problem: The single and two-vehicle cases. *European Journal of Operational Research, 215*(3), 572-580.

Raidl, G. R., Hu, B., Rainer-Harbach, M., & Papazek, P. (2013). *Balancing bicycle sharing systems: Improving a VNS by efficiently determining optimal loading operations.* Paper presented at the International Workshop on Hybrid Metaheuristics.

Rainer-Harbach, M., Papazek, P., Hu, B., & Raidl, G. R. (2013). *Balancing bicycle sharing systems: A variable neighborhood search approach.* Paper presented at the European Conference on Evolutionary Computation in Combinatorial Optimization.

Raviv, T., Tzur, M., & Forma, I. A. (2013). Static repositioning in a bike-sharing system: models and solution approaches. *EURO Journal on Transportation and Logistics, 2*(3), 187-229.

Reed, M., Yiannakou, A., & Evering, R. (2014). An ant colony algorithm for the multi-compartment vehicle routing problem. *Applied Soft Computing, 15*, 169-176.

Relvas, S., Magatão, S. N. B., Barbosa-Póvoa, A. P. F., & Neves, F. (2013). Integrated scheduling and inventory management of an oil products distribution system. *Omega, 41*(6), 955-968.

Rodríguez-Martín, I., & Salazar-González, J. J. (2011). The multi-commodity one-to-one pickup-and-delivery traveling salesman problem: a matheuristic *Network Optimization* (pp. 401-405): Springer

Salazar-González, J.-J., & Santos-Hernández, B. (2015). The split-demand one-commodity pickup-and-delivery travelling salesman problem. *Transportation Research Part B: Methodological, 75*, 58-73.

Schuijbroek, J., Hampshire, R., & van Hoeve, W.-J. (2013). Inventory rebalancing and vehicle routing in bike sharing systems.

Shaheen, S., & Guzman, S. (2011). Worldwide bikesharing. *Access Magazine, 1*(39).

Shaheen, S., Guzman, S., & Zhang, H. (2010). Bikesharing in Europe, the Americas, and Asia: past, present, and future. *Transportation Research Record: Journal of the Transportation Research Board*(2143), 159-167.

Szeto, W., Liu, Y., & Ho, S. C. (2016). Chemical reaction optimization for solving a static bike repositioning problem. *Transportation Research Part D: Transport and Environment, 47*, 104-135.

Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., & Rei, W. (2012). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research, 60*(3), 611-624.

Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2013). A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research, 40*(1), 475-489.

Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2014). A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research, 234*(3), 658-673.