EFFECTIVE ALGORITHMS FOR PICKUP AND DELIVERY PROBLEM WITH LOADING RESTRICTIONS AND HANDLING COSTS

By

DEVARAJA VIGNESH RADHA KRISHNAN

Bachelor of Science in Aeronautical Engineering
Anna University
Chennai, Tamil Nadu, India
2012

Master of Science in Industrial Engineering
Industrial Engineering & Management
Oklahoma State University
Stillwater, Oklahoma, USA
2015

Submitted to the Faculty of the
Graduate College of
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
December, 2020

EFFECTIVE ALGORITHMS FOR PICKUP AND DELIVERY PROBLEM WITH LOADING RESTRICTIONS AND HANDLING COSTS

Dissertation Approved:

Dr. Tieming Liu

Dissertation Advisor

Dr. Austin Buchanan

Dr. Manjunath Kamath

Dr. Yongwei Shan

Dr. Sheryl Tucker

Dean of the Graduate College

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Dr. Tieming Liu for his support, motivation and patience. He took me as an advisee during a desperate period of my academic life and pushed me across the finish line. I hope to take his kindness and pay it forward sometime. I am very grateful to the rest of my committee: Dr. Austin Buchanan, Dr. Manjunath Kamath and Dr. Yongwei Shan for their insightful comments and encouragement. They continued to support me patiently despite my many shortcomings. I am very thankful to the engineering team at Transplace Inc., especially Mr. Prasad Mahajan who helped me a lot to develop professionally during my course period.

My sincere thanks also goes to my colleagues, faculty and staff in the IEM department at Oklahoma State University. They made my course duration a very pleasant experience. My parents and sister endured a lot during my course period, but they always supported me with a smile on their face. I am indebted to them for my life. Finally, I would like to thank my friends, teachers and everybody else who influenced my life and guided me on my journey so far.

Disclaimer: Acknowledgements reflect the views of the author and are not endorsed by committee members or Oklahoma State University.

Name: Devaraja Vignesh Radha Krishnan   Date of Degree: December, 2020

Institution: Oklahoma State University   Location: Stillwater, Oklahoma

Title of Study: EFFECTIVE ALGORITHMS FOR PICKUP AND DELIVERY PROBLEM WITH LOADING RESTRICTIONS AND HANDLING COSTS

Candidate for the Degree of Doctor of Philosophy

Major Field: Industrial Engineering & Management

Abstract: The Pickup-and-Delivery problem is an important category of Vehicle Routing Problem with a lot of practical applications. In practice, the problems in this category often have to be solved with cargo loading/unloading restrictions. For example, shippers may incur cargo handling costs if a driver has to unload and reload pallets into the vehicle at shipment delivery sites. However, this cost can be eliminated by following the Last-In-First-Out (LIFO) order for cargo loading/unloading. Motivated by this application, we explore the Pickup-and-Delivery Problem (PDP) with LIFO loading restrictions in single and multi-vehicle settings. We also study the PDP with handling costs in single and multi-vehicle settings because strictly imposing the LIFO order might force the vehicles to travel long distances. For single-vehicle problems, we present multiple mathematical models and branch-and-cut algorithms. We also introduce new inequalities, warm start, and bound tightening procedures to enhance the scalability of our implementations. The multi-vehicle problems are formulated and solved with many practical considerations including vehicle capacity, customer time windows, and maximum on-road time for drivers. We also propose new heuristic algorithms which were very effective in solving the multi-vehicle problems. This dissertation also introduces new conditional integral separation procedures which could be applicable in large scale mathematical models outside the vehicle routing discipline.

TABLE OF CONTENTS

Chapter Page

Chapter Page

LIST OF TABLES

ABBREVIATIONS

Abbreviation	*Explanation*
ATSP	Asymmetric Traveling Salesman Problem
BB	Branch-and-Bound
BFS	Breadth First Search
DFJ	Dantzig-Fulkerson-Johnson Constraints for sub-tour elimination
FSP1	Fractional Separation Problem 1 (for sub-tour elimination constraints)
ISP1	Integral Separation Problem 1 (for sub-tour elimination constraints)
FSP2	Fractional Separation Problem 2 (for precedence constraints)
ISP2	Integral Separation Problem 2 (for precedence constraints)
FSP3	Fractional Separation Problem 3 (for LIFO constraints)
FSP3H	Fractional Separation Problem 3 (for handling costs)
HC	Handling Cost (Last-In-First-Out violation penalty)
ISP3	Integral Separation Problem 3 (for LIFO constraints)
ISP3H	Integral Separation Problem 3 (for handling costs)
LIFO	Last-In-First-Out
MIP	Mixed Integer Programming
MPDPTL	Multiple Vehicle Pickup-and-Delivery Problem with Time windows and Loading Constraints
MPDPTH	Multiple Vehicle Pickup-and-Delivery Problem with Time windows and Handling Cost
MRP-F	Master Relaxation Problem for branch-and-cut algorithm with fractional separation procedure
MRP-I	Master Relaxation Problem for branch-and-cut algorithm with integral separation procedure
PDP	One-to-one Pickup-and-Delivery Problem
PRE	Precedence constraints
PS-Tuple	Path Sub-tour Tuple
SEC	Sub-tour Elimination Constraints
SPDPH	Single Vehicle Pickup-and-Delivery Problem with Handling costs
SPDPH1	Exact formulation for SPDPH proposed by Veenstra et al. [42]
SPDPH2	Compact formulation for SPDPH presented in this dissertation
SPDPH3	Cut-based formulation for SPDPH presented in this dissertation
SPDPH3-F	Branch-and-Cut algorithms with Fractional separation procedures for SPDPH3
SPDPH3-I	Branch-and-Cut algorithms with Integral separation procedures for SPDPH3
SPDPL	Single Vehicle Pickup-and-Delivery Problem with LIFO Loading Constraints
SPDPL-Cut	Cut-based formulation for SPDPL presented by Cordeau et al. [18]
SPDPL-F	Branch-and-Cut algorithms with Fractional separation procedures for SPDPL-Cut
SPDPL-I	Branch-and-Cut algorithms with Integral separation procedures for SPDPL-Cut
VRP	Vehicle Routing Problem

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

1.1 Trucking industry in U.S.

The trucking industry contributed over $700 billion to U.S. annual revenue in 2017 [4]. This constituted 84% of total annual revenue in the U.S. commercial transportation sector that year. Around 71% of all freight tonnage moved in the U.S. is by trucking [8]. Furthermore, this industry is the source of many direct and indirect employment opportunities in the country. Nearly 6% of all full-time jobs in the U.S. are in the trucking industry [8]. It is a major industry with significant economic implications for the country. However, the U.S. trucking industry is very fragmented. Currently there are over 110,000 carriers and 350,000 independent owner-operators [9]. Among them, around 97% of the carriers own less than 20 trucks, and around 90% own six or lesser trucks [3]. This fragmentation hinders the efficiency of cargo transportation. An estimated 15% to 25% of trucks on road are traveling empty [26]. This reduced efficiency causes a hike in shipping prices, greenhouse gas emissions, and traffic congestion. Further considering the unused space in non-empty trucks, there is more need for better efficiency. Truck sharing is one such way to attain better efficiency in cargo transportation.

Internet and mobile computing technology have made truck sharing more viable.

The number of online marketplaces for freight-matching is on the rise. The concept of freight-matching is like Uber which connects driver and passenger based on request. However, the working principle behind freight-equipment matching is more complicated than Uber, because of the sizes and types of freights and trucks. It is very difficult and time-consuming for carriers to search for shippers' demand information online to identify freight consolidation options. It will be very helpful if the online freight-matching marketplace could provide consolidation solutions to the carriers. Therefore, online market places are in great need of effective freight consolidation algorithms. This dissertation is an effort towards identifying effective consolidation techniques to make truck sharing viable. Significant contributions of this dissertation are algorithms and techniques to solve a specific class of vehicle routing problems.

1.2 Pickup-and-Delivery Problems

Vehicle Routing Problem (VRP) plays the central task in the day-to-day operations of the trucking industry. VRP forms the basis of optimization tools and procedures in most of the trucking enterprises across the U.S. Given a set of customer requests and a fleet of vehicles, VRP seeks to find an optimal set of routes with the minimal operating cost. Due to a wide array of practical aspects in truck operations and routing, VRP is a vast topic with many variants. It has been a topic of great interest to the scientific community owing to its practical applications and difficulty to solve. So, VRP has been intensely studied for over half a decade since its introduction in 1959 by Dantzig and Ramser [23] which is a seminal paper addressing the problem of fuel delivery to gas stations. For general surveys of VRP, we refer the reader to

Vigo and Toth [41], and Cordeau et al. [19].

Pickup-and-delivery problems are an important category of VRP with many practical implementations. The objective of this problem is to find minimal cost vehicle routes when customer requests require vehicles to pick up commodities at certain locations and deliver them elsewhere. Depending on the route structure and type of demand, pickup-and-delivery problems can be classified into three different types:

(1) *Many-to-Many* (M-M) problems seek to find vehicle routes when there are multiple commodities with each commodity having multiple pickup and delivery locations. Furthermore, any location may supply or request multiple commodities. Deliveries from warehouses to retailers and inventory redistribution of finished products among retail stores are some of the practical applications of M-M problems.

(2) *One-to-Many-to-One* (1-M-1) problems arise when there are some commodities to be picked up from customers and delivered to the depot, and other commodities are to be delivered from depot to customers. Milk delivery where dairy products are delivered to customers and empty bottles are collected is a practical application of 1-M-1 problems.

(3) *One-to-One* problems arise when there are multiple customer requests, with each request requiring vehicle(s) to pick up a commodity from an origin and deliver it to a destination. The objective is to identify vehicle routes with the minimal cost from an origin depot to a destination depot while satisfying multiple customer requests. One-to-One problems often arise in the context of

Less-Than-Truckload (LTL) shipping and short-haul transportation.

We address one-to-one Pickup-and-Delivery Problem as *PDP* for the remainder of this document. The focus of this dissertation is on four variants of PDP.

1.3 Pickup-and-Delivery Problem with Loading Constraints

Single Vehicle Pickup-and-Delivery Problem with Loading Constraints (SPDPL) is a variant of PDP in which the loading order of cargo is considered along with vehicle routing. SPDPL enforces *Last-In-First-Out (LIFO)* loading order for pickups and deliveries. This means that an item being picked should be placed at the rear-end of the vehicle and an item can be delivered only if it is at the rear end of the vehicle. Consider two customer requests: $\boldsymbol{S_i}$ (to be picked up at $\boldsymbol{i_+}$ and delivered at $\boldsymbol{i_-}$) and $\boldsymbol{S_j}$ (to be picked up at $\boldsymbol{j_+}$ and delivered at $\boldsymbol{j_-}$). Figure 1.1 is a vehicle route starting from depot $\boldsymbol{d}$, satisfying the aforementioned customer requests while violating LIFO. On the contrary, Figure 1.2 is a vehicle route respecting the LIFO order.

SPDPL arises in the context of vehicles with a single access point at the rear-end. Furthermore, it has the following practical implications:

(1) In local pickups and deliveries where customers are clustered close together, loading/unloading comprises a significant part of trip time. So, the driver can save time by following the LIFO loading/unloading order.

(2) Automated Guided Vehicle (AGV) operations within a warehouse require pickup and delivery in a stack structure which follows the LIFO loading/unloading order.

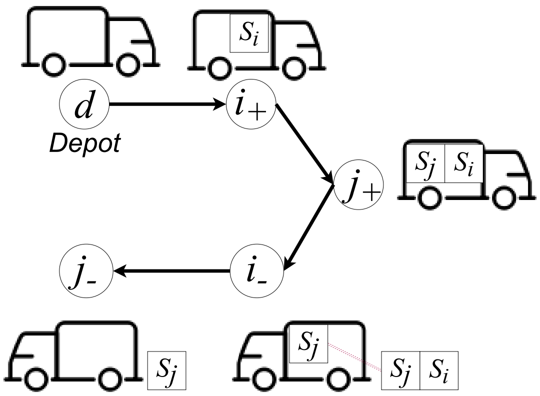


Figure 1.1: A LIFO violating route

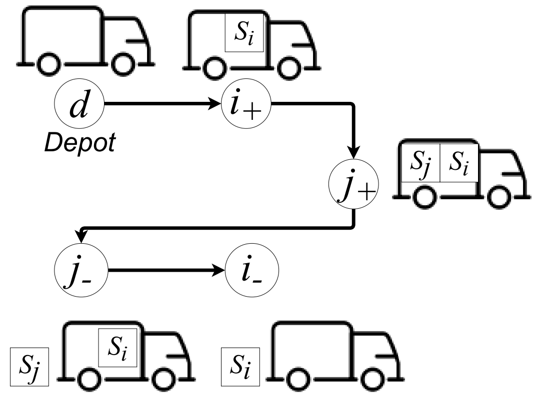


Figure 1.2: A LIFO enforced route

(3) In short-haul LTL trips involving warehouses, handling costs (HC) are a concern due to cargo loading/unloading workers in warehouses known as *Lumpers*. They are paid by the shipping company based on the quantity of cargo handled. Lumper fee is often unavoidable in certain warehouses due to some United States Code provisions [32]. LIFO loading/unloading order reduces the quantity of cargo handled and by extension, the Lumper fee is also reduced.

(4) One more practical motivation behind this problem is the latest developments in autonomous vehicles. Delivery to customers by self-driving trucks is not a distant prospect. In that case, the driver will not be available at customer sites for unloading. So, customers might have to assume the unloading task. LIFO enforced routes will be very convenient for customers in this scenario because they can unload their cargo from the access end of the vehicle without additional handling complications. Furthermore, letting a customer handle

another customers' shipment might lead to liability issues which can be avoided with LIFO enforced routes.

SPDPL is one of the problems that we address in this dissertation. We also focus on *Multiple Vehicle Pickup-and-Delivery Problem with Time windows and Loading constraints (MPDPTL)* which is SPDPL with multiple vehicles and time windows for customer service.

1.4 Pickup-and-Delivery Problem with Handling Costs

Single Vehicle Pickup-and-Delivery Problem with Handling costs (SPDPH) is another variant of PDP. In SPDPL, a vehicle may have to travel a long distance just to satisfy the LIFO loading/unloading order. Therefore, penalizing LIFO violations instead of strict LIFO enforcement can be a preferable alternative. So, SPDPH seeks to find an optimal vehicle route from origin to destination depot while handling multiple customer requests, and enforcing penalty (handling costs) for LIFO violations.

The objective of SPDPH is to find an effective trade-off between transportation cost and handling cost. By penalizing LIFO violations, we can choose to either satisfy LIFO and avoid penalty or to incur handling cost by choosing a shorter path. For example, consider two customer requests: S_i (pickup at i_+ and delivery at i_-) and S_j (pickup at j_+ and delivery at j_-) and a handling cost of \$30 for a LIFO violation. Figure 1.3(a) shows a vehicle route satisfying LIFO loading order with a total transportation cost of \$500. Whereas, Figure 1.3(b) shows a vehicle route violating LIFO once and incurring a penalty with a total cost of \$480 (transportation cost: \$450 and handling cost:\$30).

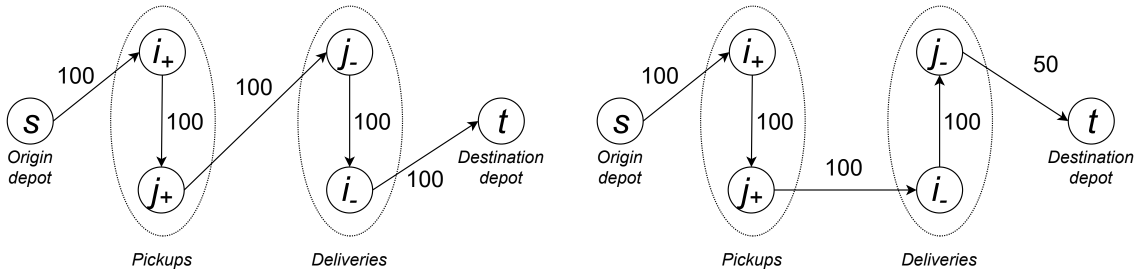


Figure 1.3: (a) Total traveling cost=\$500 (b)Total cost=\$450+\$30=\$480

SPDPH is useful for routing short-haul trips to balance the transportation cost with warehouse Lumper Fee. In short-haul trips, the handling cost is not dominated by the transportation cost. Reducing travel costs may result in handling cost increment. Conversely, reducing handling costs might cause travel cost increment. SPDPH seeks to balance these two cost aspects.

With self-driving trucks, SPDPH is applicable if a customer is willing to do additional cargo handling (perhaps with an incentive subject to liability). As a result, the total traveling distance might be reduced by violating the LIFO loading/unloading order. This may result in minor savings for a single truck, but for a nationwide franchise like UPS the savings could have a huge impact. Transportation during disaster relief is a broader application of SPDPH. During emergency times, short travel distances with minimal cargo handling is a crucial aspect. A trade-off between travel distance and cargo handling is required to minimize response time. Disaster relief is a perfect practical application for SPDPH.

SPDPH is one of the problems that we address in this dissertation. Furthermore, we also focus on *Multiple Vehicle Pickup-and-Delivery Problem with Time windows and Handling cost (MPDPTH)* which is SPDPH in a multi-vehicle setting with time windows for customer service.

1.5 Organization

The remainder of this document is organized as follows: Chapter 2 presents the past works in the literature pertaining to SPDPL, SPDPH, MPDPTL, and MPDPTH problems. Chapter 3 presents the problem statements and specific objectives of this dissertation. Chapter 4 focuses on MIP formulations, algorithms, and methodologies for single-vehicle problems we address in this dissertation. Chapter 5 explains our approaches to solving multi-vehicle problems. Chapter 6 presents computational results from our implementations. In Chapter 7, we conclude this dissertation with a summary of the results and future directions for research.

Some content from Chapter 4 and 6 have been submitted for a journal publication at the time of this dissertation writing (Radha Krishnan and Liu [35]). Also, some contents from Chapters 5 and 6 are currently under preparation for another publication. All the figures in this dissertation were generated using Diagrams.net© and Tableau© 2018.1.

CHAPTER II

LITERATURE REVIEW

This chapter presents the literature review of pickup and delivery problems, loading constraints, and handling costs. For the readers' convenience, a comprehensive summary of literature pertaining to our dissertation is shown in Table 2.1.

One-to-one Pickup-and-Delivery Problem (PDP) has been extensively studied in the literature due to its practical relevance in the transportation sector. We refer the reader to Cordeau et al. [20], and Vigo and Toth [41] for a general review of PDP. There are some notable works in the PDP literature with a focus on exact models and algorithms like Ruland and Rodin [37], and Dumitrescu et al. [24]. However, most of the works in literature seek to solve PDP using heuristics because of the problem difficulty and industrial time restrictions.

PDP with LIFO loading and handling costs has not received much focus. We present some of the relevant literature in this section and classify them under single-vehicle and multi-vehicle categories.

2.1 Single-Vehicle PDP with Loading Constraints

In this section, we present some relevant works from single-vehicle PDP with loading constraints. SPDPL was introduced by Ladany and Meherz [30] in their study of a

Table 2.1: Literature review summary

Category	Problem	Approaches	References
Single vehicle	PDP with LIFO Constraints	Theoritical introduction	Ladany and Meherz [30]
		Branch-and-Bound	Pachecho [34, 33]; Cassani [12]; Carrabs et al. [10]
		Branch-and-Cut with fractional separations	Cordeau et al. [18]
		Branch-and-Cut with integral separations	**This dissertation**
	PDP with LIFO in 2D containers	Tabu search heuristics	Gendreau at al. [29]; Malapert et al. [31]
	PDP with LIFO and Multiple stacks	Branch-and-cut	Côté et al. [21]
		Large neighborhood search	Côté et al. [22]
	PDP with handling costs (single commodity)	Compact formulation and neighborhood search	Veenstra et al. [42]
		Branch-and-Cut	**This dissertation**
	PDP with handling costs (multiple commodities)	Branch-and-cut, and two-phase heuristic	Battarra et al. [6]
		Metaheuristic	Erdogan et al. [27]
Multi vehicle	PDP with time windows	Branch-and-cut-and-price	Ropke et al. [36]
	PDP with time windows and LIFO	Set partitioning model, and branch-and-price-and-cut algorithm	Cherkesly et al. [14]
	PDP with LIFO and max. route duration	Branch-and-cut	Benavent et al. [7]
	PDP with LIFO, time windows and max. route duration	Branch-and-cut, and heuristic	**This dissertation**
	PDP with HC, time windows and max. route duration	Branch-and-cut, and heuristic	**This dissertation**

fuel delivery problem in Israel. They theoretically introduced SPDPL but solution approaches were not discussed. Some works in the literature have focused on solving SPDPL in a Branch-and-Bound (BB) framework. Pachecho [34] [33] published some of the earlier works by solving SPDPL in a BB framework. Cassani [12] presented a BB framework to solve SPDPL using lower bounds calculated using minimum spanning tree. Their implementation optimally solved instances with up to 11 customer requests. Carrabs et al. [10] presented an additive branch-and-bound algorithm that improved the runtime by identifying additive lower bounds and restricting the number of nodes in the enumeration tree.

One of the papers very relevant to this dissertation was proposed by Cordeau et al. [18]. They presented three Mixed Integer Programming (MIP) formulations for SPDPL. A cut-based MIP formulation for SPDPL presented by Cordeau et al [18] is essential for one of the algorithms we present in this dissertation. They also presented a branch-and-cut algorithm with fractional separation procedures to identify violated inequalities for a MIP formulation with an exponential number of constraints. These separation procedures were solved using maximum flow algorithms. We present and discuss the details of this formulation, and branch-and-cut algorithms in Chapter IV. Cordeau et al [18] also introduced a nice property of SPDPL solution which is essential for our MPDPTL problem formulation. More details about this property are discussed in Chapter V.

Côté et al. [21] studied SPDPL with multiple stacks. They extended the algorithms and methodologies proposed by Cordeau et al. [18] for SPDPL with single stack to SPDPL with multiple stacks. The same problem was solved using large neighborhood search heuristic by Côté et al. [22]. Heuristics for SPDPL with movement inside two-dimensional containers was presented by Malapert et al. [31] and Gendreau at al.[29].

SPDPL literature gap: All the past literature work in SPDPL involves solving the problem with heuristics, and branch-and-cut algorithms with fractional separation procedures. However, solving SPDPL in a branch-and-cut framework with integral separation procedures has not been explored in the literature up to our knowledge.

While SPDPL has received sparse attention in the literature, SPDPH has received even lesser attention. Battarra et al. [6] introduced the Traveling Salesman Problem with Pickups, Deliveries, and Handling Costs (TSPPD-H) where handling costs were included in cost objective for One-to-Many-to-One problem. In this problem, all supplies originate from the depot and all deliveries are destined for the depot, and each customer may have a supply or demand. They considered the pick ups and deliveries to be different commodities (consider types $\boldsymbol{a}$ and $\boldsymbol{b}$), therefore at customer sites, some items of type $\boldsymbol{a}$ have to be unloaded with handling cost before accessing a type $\boldsymbol{b}$ item. They presented three MIP formulations and a branch-and-cut algorithm to solve TSPPD-H with instances up to 25 customer orders. Erdogan et al. [27] solved TSPPD-H on larger instances with up to 200 customers using metaheuristics.

SPDPH was introduced recently by Veenstra et al. [42]. To our knowledge, they presented the first work on handling cost variant of one-to-one pickup and delivery problem. They presented an Integer Programming (IP) formulation and a heuristic to solve SPDPH. Due to the relevance to our work, we present their IP formulation in Section 4.3.1.

SPDPH literature gap: There are studies in the literature about SPDPH with focus on compact formulations and heuristics. However, to our knowledge, cut-based formulations and branch-and-cut algorithms have not been studied for SPDPH.

2.2 Multi-Vehicle PDP with Loading Constraints

In this section, we present some relevant works from multi-vehicle PDP with loading constraints.

Multi-vehicle PDP is a well-studied problem in the literature. This problem has been addressed by many variants of branch-and-cut, column generation, and heuristic schemes. The reader is referred to Cordeau et al. [19] for a general review of multiple vehicle PDP. Ropke et al. [36] presented two formulations, and a branch-and-cut-and-price algorithm to solve multiple vehicle PDP with time windows for customers. The MIP formulation presented by them is very relevant for this dissertation. We discuss more about this formulation in Chapter 5.

Loading constraints in a multi-vehicle setting is a fairly recent topic with very sparse literature. LIFO loading constraints in multi vehicles PDP was simultaneously introduced by Cherkesly et al. [14] and Benavent et al. [7]. Cherkesly et al. [14] presented a three-index formulation, set partitioning model, and branch-and-price-and-cut algorithm for multi vehicles PDP with time windows and LIFO loading constraints. They modified and used single-vehicle type constraints presented by Ropke and Cordeau [36] for multi-vehicle PDP with time windows. Cherkesly et al. [14] also identified that the set partitioning model performed better than the three index model and yielded better linear relaxation bounds. Benavent et al. [7]

presented formulations and branch-and-cut algorithms for multi-vehicle PDP with LIFO loading constraints and maximum route duration.

MPDPTL literature gap: Multi-vehicle PDP with time windows and LIFO loading constraints has been explored in the literature. Multi-vehicle PDP with LIFO loading constraints and maximum route duration has also been studied in the literature. However, multi vehicles PDP with time windows, LIFO loading, and maximum route duration has not been studied in the literature up to our knowledge.

MPDPTH literature gap: To our knowledge, Multiple Vehicle Pickup-and-Delivery Problem with Time Windows and Handling Cost has not been studied in the literature.

CHAPTER III

RESEARCH STATEMENT

In this chapter, we state the problems of interest, discuss specific research objectives, and relevant tasks. The scope of this dissertation covers four closely related problems.

3.1 Problem statements

Single vehicle Pickup-and-Delivery Problem with Loading constraints (SPDPL)

The problem is to identify an optimal route for a single vehicle from an origin depot to a destination depot, satisfying multiple pickup-and-delivery requests and respecting the LIFO loading/unloading order for cargo handling. Heuristics and branch-and-cut algorithms with fractional separation procedures have been studied in the literature for SPDPL. However, branch-and-cut algorithms with integral separation procedures have not been explored which is one of the approaches we explore in this dissertation.

Single vehicle Pickup-and-Delivery Problem with Handling costs (SPDPH)

The problem is to identify an optimal vehicle route, satisfying multiple pickup-and-delivery requests, and incurring handling costs for each additional unloading/reloading operation at delivery locations. This problem is motivated by the necessity to find a fair trade-off between handling costs and travel distance. SPDPH was very recently introduced in the literature with a compact formulation and heuristic. Our interests are on cut-based formulations, and branch-and-cut algorithms for SPDPH which have not yet been addressed in the literature.

Multi vehicle Pickup-and-Delivery Problem with Time windows and Loading constraints (MPDPTL)

Given a homogeneous fleet of vehicles with uniform capacity, customer requests with time windows, and maximum on-road time for drivers, MPDPTL seeks to identify vehicle routes satisfying the requests and time constraints. Furthermore, the routes should also respect the LIFO loading/unloading order for cargo handling. MPDPTL is one of the problems we address in this dissertation and it has not been explored in the literature up to our knowledge.

Multi vehicle Pickup-and-Delivery Problem with Time windows and Handling cost (MPDPTH)

This problem is similar to MPDPTL except for the fact that the LIFO order is not strictly enforced for cargo handling. Instead, LIFO violations are penalized by handling costs for additional unloading/reloading operations at delivery locations for each vehicle. MPDPTH has not been explored in the literature up to our knowledge and is a problem of interest in this dissertation.

3.2 Specific tasks

Objective 1- Exploring a new solution methodology for SPDPL.

- **Task 1.1.** Developing a new branch-and-cut algorithm with integral solution separation techniques to solve SPDPL. Branch-and-cut algorithms with fractional and heuristic procedures have been explored in the literature, but integral separation procedures have not been explored yet.

- **Task 1.2.** Empirically assessing the impacts of our integral separation techniques against fractional separation procedures in the literature.

Objective 2- Developing new MIP models and solution methodologies for SPDPH.

- **Task 2.1.** Building new mathematical models for SPDPH. An exact formulation for this problem is already available in the literature. However, a cut-based formulation has not been proposed.

- **Task 2.2.** Developing branch-and-cut algorithms with integral and fractional separation procedures to solve SPDPH.

- **Task 2.3.** Strengthening our implementation by introducing new inequalities.

- **Task 2.4.** Comparing the computational scalability of our algorithms with each other and against the exact formulation provided in the literature.

Objective 3- Developing new mathematical models and solution methodologies for MPDPTL.

- **Task 3.1.** Developing MIP models and heuristics to effectively solve MPDPTL.

- **Task 3.2.** Extending the findings from Objective 1 to multiple vehicles setting to enhance our MPDPTL implementation.

Objective 4- Exploring practically implementable algorithms to solve MPDPTH.

- **Task 4.1.** Applying the results from Objectives 1, 2 and 3 to build an optimization model for MPDPTH

- **Task 4.2.** Assessing the computational scalability of our implementations to ensure that real-world instances are solved within reasonable runtime.

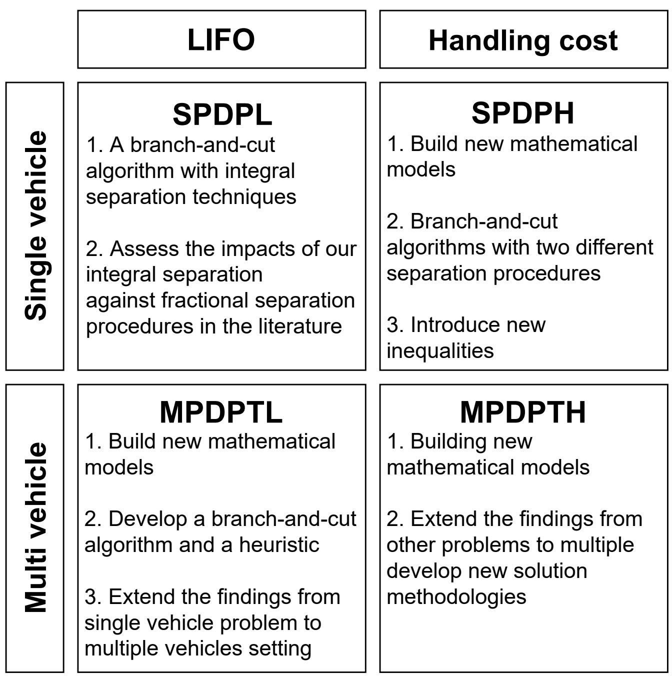


Figure 3.1: Research tasks for the four problems in this dissertation

3.3 Major contributions of this dissertation

In this section, we discuss the major contributions of this dissertation

- We introduced two new problems to the VRP literature in this dissertation, namely: multi-vehicle pickup-and-delivery problem with LIFO and handling costs. These problems have a variety of practical considerations including vehicle capacity, fleet size, customer time windows, driver operating hours, and cargo loading restrictions. We discuss the multi-vehicle problem solution methodologies in Chapter V.

- The branch-and-cut (BC) algorithms in the PDP literature pervasively have a framework that does not permit infeasible integer solutions to be generated at any step in the solution approach. So, none of the model constraints are violated while solving the problem. However, this is not an effective approach because strictly respecting all the constraints in every step could increase the runtime considerably. This increase is due to solving a hard optimization problem while respecting a large number of constraints. So, we developed BC algorithms that permit some constraint violations that are identified and removed later, hence reducing the runtime. In Sections 4.2.4 and 5.2.6, we present the algorithms in single-vehicle and multi-vehicle settings respectively.

- We introduced new conditional structures called integral separation procedures for identifying infeasible integer solutions. This structure is based on a sequence of logical decisions. For example, it is not logical to search for LIFO violations in a path with precedence violations, because that solution will be discarded later. The algorithms based on this logic can be applied to problems with exponential sets of constraints. This could give rise to new algorithms for other problems besides PDP. We discuss this separation procedure details in Section 4.2.4.

- Another major contribution of this dissertation is the multi-vehicle Warm-Start (WS) heuristic algorithms. The WS heuristics identified effective solutions to multi-vehicle problems within very short runtime. Furthermore, we use the results from these algorithms as a starting solution to our exact approaches. The heuristics work by iterating through two logics:

  (1) Clustering and assigning customer requests to vehicles such that the constraints are not violated

  (2) Routing the vehicles by measuring the trade-off between enforcing LIFO and handling costs for customer requests

  In Section 5.2.5, we discuss the WS algorithm structure details. We also discuss the computational scalability of the WS algorithms in Chapter VI

CHAPTER IV

SINGLE VEHICLE PROBLEMS

In this chapter, we present the formulations and methodologies for single-vehicle problems. As mentioned in Chapter III, we address two problems under single vehicle category: Pickup-and-Delivery Problem with Loading constraints (SPDPL) and Pickup-and-Delivery Problem with Handling costs (SPDPH).

4.1 Notations and definitions

In this section, we present some terminologies and notations which will be used in this chapter for single vehicle problems.

4.1.1 Graph structure

Let n denote the number of customer shipment requests. Consider a complete directed graph $G = (N, A)$ with node set $N = \{0, 1, \ldots, 2n, 2n+1\}$ and arc set A. Node subsets $P = \{1, \ldots, n\}$ and $D = \{n+1, \ldots, 2n\}$ are pickup and delivery nodes respectively, whereas 0 and $2n+1$ are origin and destination depots respectively. Each customer request is to transport a load from pickup node $i \in P$ to delivery node $n+i \in D$. We use $i \prec j$ to denote the fact that node i precedes node j in the vehicle route.

For readers' convenience, a centralized table for decision variables, set definitions and other notations has been created (Table 4.1). We refer to this table from the formulations as per necessity.

We also present the following definitions which are required for understanding problem statements.

Definition 1. *A Hamiltonian Path is a directed path from an origin node to a destination node visiting each node in* $\boldsymbol{N}$ *exactly once.*

Definition 2. *A sub-tour is a directed cycle in* $\boldsymbol{G}$ *which does not visit all the nodes in* $\boldsymbol{N}$*.*

Definition 3 (Ahuja et al. [2]). *The* indegree *and* outdegree *of a node is the number of incoming and outgoing arcs of that node respectively.*

Definition 4. *A Path-Subtour tuple (PS-tuple) is a subgraph containing exactly one directed path and at least one subtour, with the path and subtour(s) being pairwise node-disjoint.*

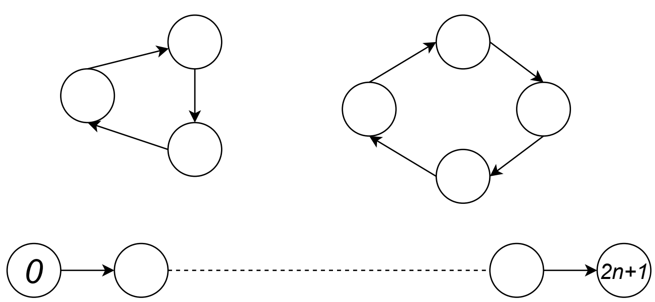


Figure 4.1: A PS-tuple with a path and two node-disjoint subtours

Table 4.1: Notations for MIP models- Single vehicle

Type	**Notation**	**Definition**
Capacity	Q	Capacity of vehicles in a homogeneous fleet
	q_i	size of the load to be transported from pickup node $i \in P$ to delivery node $n+i \in D$.
Cost parameters	c_{ij}	Cost value associated with each directed arc $(i,j) \in A$
	v	Handling cost for one unloading and reloading operation.
Decision variables	f^1_{ijk}	Equal to 1 if arc $(i,j) \in A$ is in the path from node 0 to node k; 0 otherwise
	f^2_{ijk}	Equal to 1 if arc $(i,j) \in A$ is in the path from node k to node $n+k$; 0 otherwise
	f^3_{ijk}	Equal to 1 if arc $(i,j) \in A$ is in the path from node $n+k$ to node $2n+1$; 0 otherwise
	Q_i	Load on the vehicle upon leaving node $i \in N$
	x_{ij}	Equal to 1 if arc $(i,j) \in A$ is in the vehicle route and 0 otherwise
	y_{ij}	Equal to 1 if node $i \in N$ precedes node $j \in N \setminus \{i\}$ in the vehicle route and 0 otherwise
	z_{ij}	Equal to 1 if $j \in P$ is picked up between $i \in P$ and $n+i \in D$, and delivered after $n+i$; 0 otherwise
Set definitions	Γ	Collection of all node subsets $S \subset N$ such that $0 \in S$, $2n+1 \notin S$, and there exists a node $i \in P$ such that $i \notin S$ and $n+i \in S$
	Ω	Collection of node subsets $S \subset P \cup D$ such that there is at least one customer request $j \in P$ such that either $j \in S$ and $n+j \notin S$ or $n+j \in S$ and $j \notin S$
	Υ_j	Collection of all node subsets $S \subset P \cup D$ such that $j \in S$ and $n+j \notin S$
	$\pi(S)$	Set of pickups - $\{i \in P \| n+i \in S\}$ for a node subset $S \subset N$
	$\sigma(S)$	Set of deliveries - $\{n+i \in D \| i \in S\}$ for a node subset $S \subset N$
	$\bar{S}$	$N \setminus S$ for a node subset $S \subset N$
	A'	Subset of arcs having both endpoints in $P \cup D$
Other	$A(S)$	Set of all arcs $(i,j) \in A$ such that $i,j \in S$, where $S \subset N$
	$x(A(S))$	$\sum_{i,j \in S} x_{ij}$
	$x(i,S)$	$\sum_{j \in S} x_{ij}$
	$x(S,i)$	$\sum_{j \in S} x_{ji}$

4.2 Single Vehicle Pickup-and-Delivery Problem with Loading Constraints

In this problem, a single vehicle serves all customer requests. The objective of SPDPL is to find a minimum cost Hamiltonian path from origin depot 0 to destination depot $2n+1$. We associate a load q_i with each node $i \in N$, such that $q_i = -q_{n+i}, \forall i \in P$. We also assume $q_0 = q_{2n+1} = 0$ for the depots. A feasible route must satisfy the following conditions:

(1) For each shipment $i = 1, \ldots, n$, the pickup node $i \in P$ must be visited before the delivery node $n + i \in D$.

(2) Load picked from a location should be placed at the rear end of the vehicle (top of the stack).

(3) A delivery node $n + i \in D$ can be visited only if the load picked from $i \in P$ is at the rear end of the vehicle.

Cordeau et al [18] introduced a cut-based formulation with exponential number of constraints for SPDPL. This formulation was implemented using a branch-and-cut algorithm with fractional separation procedures. In this dissertation, we present a new branch-and-cut algorithm with integral separation techniques to solve SPDPL. One of our objectives is to computationally compare our implementation against the already existing branch-and-cut approach. In the following section, we discuss the formulation and separation techniques presented by Cordeau et al [18] before introducing our branch-and-cut algorithm.

4.2.1 SPDPL formulation from Cordeau et al. [18]

Decision variables

x variables as defined in Table 4.1

Sets

Γ and Ω as defined in Table 4.1

Formulation 4.2.1 (SPDPL-Cut by Cordeau et al. [18]).

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{4.1}$$

subject to:

$$\sum_{j:(i,j)\in A} x_{ij} = 1 \quad \forall i \in P \cup D \cup \{0\} \tag{4.2}$$

$$\sum_{i:(i,j)\in A} x_{ij} = 1 \quad \forall j \in P \cup D \cup \{2n+1\} \tag{4.3}$$

$$Q_j \geq (Q_i + q_j)x_{ij} \quad \forall (i,j) \in A \tag{4.4}$$

$$max\{0, q_i\} \leq Q_i \leq min\{Q, Q + q_i\} \quad \forall i \in N \tag{4.5}$$

$$x(A(S)) \leq |S| - 1 \quad \forall S \subseteq P \cup D, 2 \leq |S| \leq |N| \tag{4.6}$$

$$x(A(S)) \leq |S| - 2 \quad \forall S \in \Gamma \tag{4.7}$$

$$x(i,S) + x(A(S)) + x(S, n+i) \leq |S| \quad \forall S \in \Omega, \forall i, n+i \notin S, i \in P \tag{4.8}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A \tag{4.9}$$

The objective function seeks to minimize the total transportation cost. Constraints (4.2) and (4.3) ensure that each pickup and delivery node is visited exactly once. Constraints (4.4) and (4.5) satisfy capacity restrictions for the vehicle. Constraints (4.4) link the load variables and arc variables, whereas Constraints (4.5) ensure that the vehicle does not exceed its capacity on any arc. So, Formulation 4.2.1 considers vehicle capacity restriction. However, the test instances for our computational experiments are uncapacitated (refer Chapter VI). So, we adapt the formulation to uncapacitated version, by setting $q_i = 1$ and $q_{n+i} = -1, \forall i \in P$. We also set the vehicle capacity as $Q = n$ to handle the uncapcitated instances. Note that Constraints (4.5) are non-linear. However, we can linearize the product of two variables using some standard linearization techniques as presented in Appendix A.

Constraints (5.23) are well-known Dantzig-Fulkerson-Johnson (DFJ) constraints for sub-tour elimination. Constraints (4.7) are to ensure that pickup node is visited before delivery node for each customer request. These precedence constraints were introduced for PDTSP by Ruland and Rodin [37]. An illustration of these constraints are shown in Figure 4.2. Let us consider a vehicle path with a precedence violation $(n + i \prec i)$ for a customer request $i \in P$. In this path, we can identify a node set $S \subset N$ such that $0, n + i \in S$ and $2n + 1, i \notin S$, and $x(A(S)) = |S| - 1$. We enforce $x(A(S)) \leq |S| - 2$ to remove at least one more arc with both endpoints in S and ensure that this path does not exist.

LIFO loading/unloading order is violated by a vehicle route, if there are two pickup nodes $i \in P$ and $j \in P \setminus \{i\}$ such that:

(1) j is in the path between i and $n + i$

(2) $n + j$ is not in the path between i and $n + i$

Constraints (4.8) are LIFO loading constraints which ensure that no such pickup

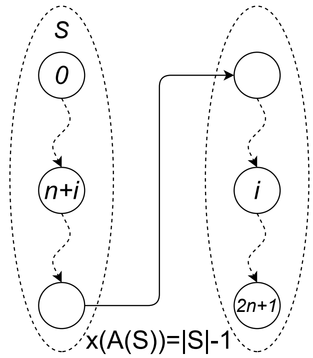


Figure 4.2: Precedence constraints illustrations

nodes exist.

Note: Constraints (4.2)-(4.7) are pervasively used in many formulations and discussions throughout this chapter. So, we will be referring back to these constraints where ever necessary to avoid repetition.

4.2.2 Inequalities for SPDPL

In this section, we explain the existing inequalities available in the SPDPL literature that we used in our implementations for runtime improvement.

Incompatible predecessor and successor inequalities

Consider two pickup nodes $i, j \in P$ such that $i \neq j$. Cordeau et al. [18] presented the incompatible successor inequalities by considering $x_{ij} = 1$ and identifying possible successors to $n + j \in D$ in a LIFO enforced vehicle route. If $x_{ij} = 1$, then the set of possible immediate successors to $n + j$ is either $n + i$ or $P \setminus \{i, j\}$. Let us denote this set of successors to $n + j$ as $S_{n+j}(i, j) = \{n + i\} \cup (P \setminus \{i, j\})$. For each node pair $i, j \in P$, the incompatible successor inequality is

$$x_{ij} + \sum_{l \notin S_{n+j}(i,j)} x_{n+j,l} \leq 1 \tag{4.10}$$

Similarly, for $i, j \in P$, Cordeau et al. [18] proposed the incompatible predecessor inequalities by considering $x_{n+i,n+j} = 1$. In this case, the set of possible predecessors to $i \in P$ is $\{j\} \cup (D \setminus \{n+i, n+j\})$ which we denote as $P_i(n+i, n+j)$. The incompatible predecessor inequality for each node pair $i, j \in P$ is

$$x_{n+i,n+j} + \sum_{l \notin P_i(n+i,n+j)} x_{l,i} \leq 1 \tag{4.11}$$

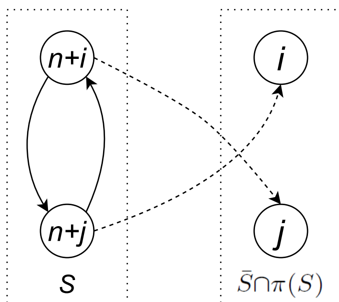
Incompatible arc set inequalities

Cordeau et al. [18] presented these inequalities which are based on a set of pairwise incompatible arcs for LIFO enforced vehicle route. For all node pairs $i, j \in P$, the following four arcs are pairwise incompatible: (i, j), $(n+i, n+j)$, $(n+i, j)$ and $(n+j, i)$. This leads to the following family of inequalities.

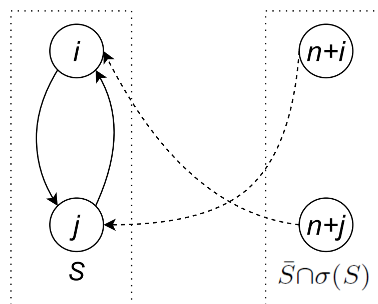
$$x_{ij} + x_{n+i,n+j} + x_{n+i,j} + x_{n+j,i} \leq 1 \tag{4.12}$$

Predecessor and successor inequalities

Let $\pi(S) = \{i \in P | n+i \in S\}$ and $\sigma(S) = \{n+i \in D | i \in S\}$ be the sets of predecessors and successors for a node subset $S \subset P \cup D$. The sub-tour elimination constraints (5.23) can be strengthened by considering the precedence relationship between pickup and delivery nodes for different customer requests. For example, let us consider a node subset $S = \{n+i, n+j\} \subseteq D$ as shown in Figure 4.3(a). The sub-tour elimination constraint for S is $x_{n+i,n+j} + x_{n+j,n+i} \leq 1$. Furthermore, we know that $i \prec n+i$ and $j \prec n+j$ in the vehicle route. If $x_{n+i,n+j} = 1$, then $x_{n+j,i}$ must be 0, otherwise we will have $n+i \prec i$ in the solution. Similarly, if $x_{n+j,n+i} = 1$, then $x_{n+i,j}$ must be 0. So, the sub-tour elimination constraint for S can be strengthened with inequality $x_{n+i,n+j} + x_{n+j,n+i} + x_{n+i,j} + x_{n+j,i} \leq 1$.



(a) Predecessor inequality for $S = \{n+i, n+j\}$



(b) Successor inequality for $S = \{i, j\}$

Figure 4.3: Predecessor and successor inequalities

Similarly, let us consider another node subset $S = \{i, j\} \subseteq P$ as shown in Figure 4.3(b). Sub-tour elimination constraint for S is $x_{ij} + x_{ji} \leq 1$. Furthermore, we know that $i \prec n+i$ and $j \prec n+j$ in the vehicle route. If $x_{ij} = 1$, then $x_{n+j,i}$ must be 0, otherwise we will have $n+i \prec i$ in the solution. Similarly, if $x_{ji} = 1$, then $x_{n+i,j}$ must be 0. By the above arguments, the sub-tour elimination constraint for S can be strengthened with inequality $x_{ij} + x_{ji} + x_{n+i,j} + x_{n+j,i} \leq 1$. Generalizing these ideas, Balas et al. [5] presented the following inequalities.

$$x(A(S)) + \sum_{i \in S} \sum_{j \in \bar{S} \cap \pi(S)} x_{ij} + \sum_{i \in S \cap \pi(S)} \sum_{j \in \bar{S} \setminus \pi(S)} x_{ij} \leq |S| - 1 \tag{4.13}$$

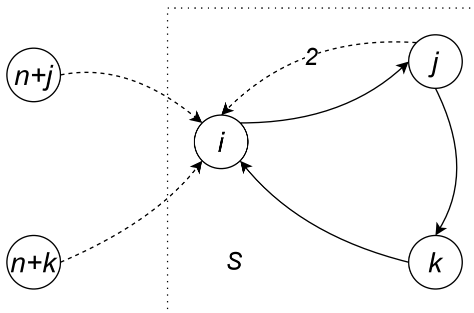
$$x(A(S)) + \sum_{i \in \bar{S} \cap \sigma(S)} \sum_{j \in S} x_{ij} + \sum_{i \in \bar{S} \setminus \sigma(S)} \sum_{j \in S \cap \sigma(S)} x_{ij} \leq |S| - 1 \tag{4.14}$$

Strengthened cycle inequalities

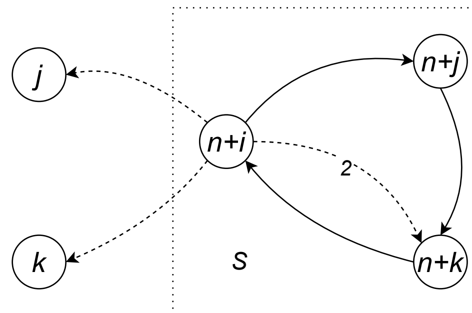
Consider a node subset $S = \{i, j, k\} \subseteq P$ as illustrated in Figure 4.4(a). The classical cycle inequality for S is $x_{ij} + x_{jk} + x_{ki} \leq 2$. Cycle arcs are (i, j), (j, k) and (k, i). Note that, if $x_{ji} = 1$, then none of the cycle arcs can be in the solution. Therefore, the cycle inequality for S can be strengthened with $x_{ij} + x_{jk} + x_{ki} + 2x_{ji} \leq 2$. Furthermore, we know that $i \prec n + i$ and $j \prec n + j$ in the vehicle route. So, if two of the three cycle arcs are in the solution, then neither $(n + j, i)$ nor $(n + k, i)$ can be in the solution. Otherwise, they will violate either the precedence requirements or the degree constraints. Therefore, the cycle inequality for S can be further strengthened as $x_{ij} + x_{jk} + x_{ki} + 2x_{ji} + x_{n+j,i} + x_{n+k,i} \leq 2$. Similarly, for an ordered node subset $S = \{n + i, n + j, n + k\} \subseteq D$ as shown in Figure 4.4(b), the classical cycle inequality $x_{n+i,n+j} + x_{n+j,n+k} + x_{n+k,n+i} \leq 2$ can be strengthened with $x_{n+i,n+j} + x_{n+j,n+k} + x_{n+k,n+i} + 2x_{n+i,n+k} + x_{n+i,j} + x_{n+i,k} \leq 2$. Cordeau [17] generalized the above idea for an ordered node subset $S = \{i_1, i_2, \ldots, i_k\} \subset N$ and presented the following inequalities.

$$\sum_{h=1}^{k-1} x_{i_h, i_{h+1}} + x_{i_k, i_1} + \sum_{h=2}^{k-1} x_{i_h, i_1} + \sum_{h=3}^{k-1} \sum_{l=2}^{h-1} x_{i_h, i_l} + \sum_{n+i_p \in \bar{S} \cap \sigma(S)} x_{n+i_p, i_1} \leq k - 1 \tag{4.15}$$

$$\sum_{h=1}^{k-1} x_{i_h,i_{h+1}} + x_{i_k,i_1} + \sum_{h=3}^{k} x_{i_1,i_h} + \sum_{h=4}^{k}\sum_{l=3}^{h-1} x_{i_h,i_l} + \sum_{i_p \in \bar{S}\cap\pi(S)} x_{i_1,i_p} \leq k-1 \tag{4.16}$$



(a) Inequality (4.15) for $S = \{i, j, k\}$



(b) Inequality (4.16) for $S = \{n+i, n+j, n+k\}$

Figure 4.4: Strengthened cycle inequalities

4.2.3 Branch-and-cut algorithm - Fractional separation (SPDPL-F)

Notice that Constraints (5.23)-(4.8) are exponential in number. Therefore, building the model for direct implementation of Formulation 4.2.1 is computationally expensive. So, Cordeau et al [18] presented a branch-and-cut algorithm in which a master relaxation problem was solved and node sets violating Constraints (5.23)-(4.8) were identified by solving maximum flow problems. Before presenting the procedure, we present the following relaxations and separation problems which are essential for our discussion.

Master Relaxation Problem for SPDPL-F

Formulation 4.2.2 (MRP-F).

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{4.17}$$

subject to:

Constraints (4.2) *and* (4.5) *from Formulation 4.2.1 (degree constraints)*

$$0 \leq x_{ij} \leq 1 \quad \forall (i,j) \in A \tag{4.18}$$

Let F be the feasible solution set for MRP-F. Cordeau et al [18] proposed a branch-and-cut algorithm (SPDPL-F) which starts by identifying a solution feasible to MRP-F. Three fractional separation problems were solved on the aforementioned solution and cutting planes were sequentially added. This procedure was repeated until a solution feasible to the original formulation (Formulation 4.2.1) was identified. The separation problems associated with SPDPL-F algorithm are presented below.

Fractional Separation Problem 1 (FSP1)- Sub-tours

Input: A directed graph $G = (N, A)$ as described in Section 4.1, and fractional values $x^* \in F$.

Problem: To identify a set of nodes S^*, such that $S^* \subseteq P \cup D$, $2 \leq |S^*| \leq N$ and $x(A(S^*)) > |S^*| - 1$, or determine that no such set exists.

Cordeau et al [18] solved FSP1 in a classical way as shown below:

(1) Create a supporting graph $G^* = (N, A^*)$ where each arc $(i,j) \in A^*$ has flow capacity equal to x^*_{ij}.

(2) Solve maximum flow problem from 0 to i, for each node $i \in N \setminus \{0, 2n+1\}$.

(3) If the max-flow value is less than 1, then search the residual graph and identify a node set S^*.

After solving FSP1, Cordeau et al [18] added inequality (5.23) for S^* as a cutting plane.

Fractional Separation Problem 2 (FSP2)- Precedence

Input: A directed graph $G = (N, A)$ as described in Section 4.1, and fractional values $x^* \in F$.

Problem: For each node $i \in P$, identify a node subset $S^* \subset N$, such that $0 \in S^*$, $n+i \in S^*$, $2n+1 \notin S^*$, $i \notin S^*$, and $x(A(S^*)) > |S^*| - 2$, or determine that no such set exists.

Cordeau et al [18] solved FSP2 for each customer request $i \in P$ as shown below:

(1) Create a supporting graph $G^* = (N, A^*)$ where each arc $(i, j) \in A^*$ has flow capacity equal to x^*_{ij}.

(2) Create two new arcs in G^*: $(0, n+i)$ and $(i, 2n+1)$ each with arc capacity 2.

(3) Solve a maximum flow problem from origin depot 0 to destination depot $2n+1$.

(4) If the max-flow value is lesser than 2, then search the residual graph and identify a node set $S^* \in \Gamma$.

(5) Note that $0, n+i \in S^*$ and $i, 2n+1 \notin S^*$ (by definition of set Γ and Constraints (4.7)).

After solving FSP2, Cordeau et al [18] added inequality (4.7) for S^* as a cutting plane.

Fractional Separation Problem 3 (FSP3)- LIFO

For separating node sets violating Constraints (4.8), Cordeau et al [18] solved separation problems for each pair of nodes $i, j \in P$ by performing two searches:

1. For all sets $S \in \Omega$, such that $j \in S$, $n+j \notin S$, $i \notin S$ and $n+i \notin S$
2. For all sets $S \in \Omega$, such that $n+j \in S$, $j \notin S$, $i \notin S$ and $n+i \notin S$

The following problem and procedure corresponds to the first search. The separation problem and procedure for second search is similar to the first one.

Input: A directed graph $G = (N, A)$ as described in Section 4.1, and fractional values $x^* \in F$.

Problem: For each pair of nodes $i, j \in P$, identify a node subset $S^* \subset P \cup D$, such that $j \in S^*$, $n+j \notin S^*$, $i \notin S^*$, $n+i \notin S^*$, and $x(i, S^*)+x(A(S^*))+x(S^*, n+i) > |S^*|$, or determine that no such set exists.

Cordeau et al [18] solved FSP3 for each pair of requests $i, j \in P$ as shown below:

(1) Create a supporting graph $G^* = (N, A^*)$ where each arc $(i, j) \in A^*$ has flow capacity equal to x^*_{ij}.

(2) Increase the capacity of following arcs to 2 in G^*: $(i, n+i)$, $(i, n+j)$, $(n+i, i)$ and $(n+j, i)$.

(3) For each node $k \in P \cup D$, add $x^*_{ik} + x^*_{k,n+i}$ to the current capacity.

(4) Solve a maximum flow problem from j to i.

(5) If the max-flow value is lesser than 2, then search the residual graph and identify a node set $S^* \in \Omega$.

(6) Note that by set definitions and Constraints (4.8), $j \in S^*$ and $i, n{+}i, n{+}j \notin S^*$.

After solving FSP3, Cordeau et al [18] added inequality (4.8) for S^* as a cutting plane.

Given a fractional solution x^*_{ij}, Table 4.2 shows the number of max flow problems solved in each separation procedure. We solve max-flow problems using Edmonds-Karp algorithm [25] with Breadth-First-Search (BFS) for graph traversal. The time complexity of each max-flow implementation is $O(n^5)$. Our implementation structures for BFS function and max-flow algorithm are presented in Appendices 2.1 and 2.2 respectively. For the readers' recollection, n denotes the number of customer requests.

Table 4.2: Constraints and #Max flow problems for SPDPL-F separations

Problem name	Constraints	Type	#Max flow problems
FSP1	5.23	Sub-tours	$2n$
FSP2	4.7	Precedence	n
FSP3	4.8	LIFO violation	$2(n^2 - n)$

SPDPL-F Algorithm Structure (Cordeau et al. [18])

Formulation 4.2.2 is solved in a Branch-and-Bound (BB) framework. Each node of the BB tree may represent one of the following cases:

- An infeasible LP relaxation, in which case we prune by infeasibility.
- A precedence abiding Hamiltonian path with no LIFO violations, in which case we update the incumbent solution accordingly.
- A fractional or integral solution not feasible to the original problem, in which case the following steps are executed:
  (1) FSP1 is solved and node sub-sets violating Constraints (5.23) (SECs), if any exists, are identified. Cutting planes corresponding to aforementioned node sub-sets are added to the current formulation.
  (2) Similarly, FSP2 and FSP3 are solved and node sub-sets violating Constraints (4.7) (precedence) and (4.8) (LIFO), if any exists, are identified. Cutting planes are added to current formulation for resolving.

This algorithm terminates when there are no nodes left in the BB tree to branch. At that point, the incumbent solution is an optimal vehicle route.

4.2.4 Branch-and-cut algorithm - Integral separation (SPDPL-I)

In this section, we introduce a new branch-and-cut algorithm with integral separation procedures to solve SPDPL. Before presenting the procedure, we present the following relaxations and separation problems.

Master Relaxation Problem for SPDPL-I

MRP-I is an assignment problem formulation

Formulation 4.2.3 (MRP-I).

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{4.19}$$

subject to:

Constraints (4.2) *and* (4.5) *from Formulation 4.2.1 (degree constraints)*

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A \tag{4.20}$$

Let I_1 be the feasible solution set for Formulation 4.2.3. We propose a branch-and-cut algorithm (SPDPL-I) which starts by identifying a solution feasible to MRP-I. We then solve three integral separation problems on the aforementioned solution. Consequently, violated inequalities are identified and added as lazy cuts to MRP-I. This process is repeated until a solution feasible to the original formulation (Formulation 4.2.1) is identified. The integral separation problems associated with SPDPL-I algorithm are presented below.

Integral Separation Problem 1 (ISP1)- Sub-tours

Input: A directed graph $G = (N, A)$ as described in Section 4.1, and binary values $x^* \in I_1$.

Problem: To identify a set of nodes S^*, such that $S^* \subseteq P \cup D$, $2 \leq |S^*| \leq N$ and $xA((S^*)) > |S^*| - 1$, or determine that no such set exists.

We solve ISP1 with a simple graph traversal from origin depot 0 to destination depot $2n+1$ with runtime complexity of $O(n^2)$. After a search in x^*, if there are unreachable nodes from the origin depot, then they are a part of sub-tour(s). This is because $x^* \in I_1$ is either a Hamiltonian path (all nodes are reachable from the source node) or a PS-Tuple (pairwise node-disjoint path and sub-tours). If sub-tours are identified, then we add Constraints (5.23) as lazy cuts. Our implementation structure to solve ISP1 is presented in Appendix 2.3.

Before presenting the next separation problem, we add Constraints (5.23) (sub-tour elimination) to Formulation 4.2.3 and denote the new formulation as RP1. Let I_2 be the feasible solution set for Formulation RP1. The following separation problem seeks to identify precedence violations in a Hamiltonian path.

Integral Separation Problem 2 (ISP2)- Precedence

Input: A directed graph $G = (N, A)$ as described in Section 4.1, and binary values $x^* \in I_2$.

Problem: For each node $i \in P$, identify a node subset $S^* \subset N$, such that $0 \in S^*$, $n + i \in S^*$, $2n + 1 \notin S^*$, $i \notin S^*$, and $x(A(S^*)) > |S^*| - 2$, or determine that no such set exists.

We solve ISP2 by performing a graph traversal from origin depot 0 to destination depot $2n + 1$ and numbering the nodes in their order of visit. The time complexity for aforementioned traversal is $O(n^2)$. All nodes are reachable from the origin because $x^* \in I_2$ is a Hamiltonian path. Let h_i be the position (order of visit) of node $i \in N$ in the path form origin to destination depot. If we can identify a node $i \in P$ such that $h_{n+i} < h_i$, then delivery node $n + i$ was visited before the pickup node i. We construct a node set S^* containing nodes corresponding to path positions $h_0, \ldots, h_{n+i}, \ldots, (h_i - 1)$. After that, we add inequality (4.7) for S^* as a lazy cut.

Our implementation structure for ISP2 is presented in Appendix 2.4.

Before presenting the LIFO separation problem, we add Constraints (4.7) (precedence) to RP1 and denote the new formulation as RP2. Let I_3 be the feasible solution set for RP2. The following separation problem seeks to identify LIFO violations in a Hamiltonian path with no precedence violations.

Similar to FSP3 (fractional separation procedure for LIFO violations), integral separation procedures for LIFO violations should also be solved by performing two searches.

(1) For all sets $S \in \Omega$, such that $j \in S$, $n+j \notin S$, $i \notin S$ and $n+i \notin S$

(2) For all sets $S \in \Omega$, such that $n+j \in S$, $j \notin S$, $i \notin S$ and $n+i \notin S$

The following problem and procedure corresponds to the first search. The separation problem for second search is equivalent to the first one.

Integral Separation Problem 3 (ISP3)- LIFO violation

Input: A directed graph $G = (N, A)$ as described in Section 4.1, and binary values $x^* \in I_3$.

Problem: For each node pair $i, j \in P$, $i \neq j$, identify a node subset $S^* \subset P \cup D$, such that $i \notin S^*$, $n+i \notin S^*$, $j \in S^*$, $n+j \notin S^*$, $x(i, S^*) + x(A(S^*)) + x(S^*, n+i) > |S^*|$ or determine that no such set exists.

Since, $x^* \in I_3$, we know the input to ISP3 is a precedence abiding Hamiltonian path. We solve ISP3 with a procedure similar to ISP2 (graph traversal and numbering the nodes by order of visit). Let h_i be the position (order of visit) of node $i \in N$ in the path form origin to destination depot. There is a LIFO violation, if i and j are located in the path such that

- $h_i < h_j$ (i is visited before j)
- $h_j < h_{n+i}$ (j is visited before $n+i$) and
- $h_{n+i} < h_{n+j}$($n+i$ is visited before $n+j$)

We then construct a node set S^* containing nodes corresponding to path positions $(h_i+1), \ldots, h_j, \ldots, (h_{n+i}-1)$. After that, inequality (4.8) is added for S^* as a lazy cut. Our implementation structure is presented in Appendix 2.5.

Notice that ISP1, ISP2 and ISP3 are defined and solved on integral solutions. Also, LIFO violations are separated on a solution only if it is a Hamiltonian path with no precedence violations. With this remark, we present our branch-and-cut algorithm with nested integral separation procedures.

SPDPL-I Algorithm Structure

Formulation MRP-I is solved in a Branch-and-Bound (BB) framework. Each node of the BB tree may represent one of the following cases:

- A fractional solution, in which case we continue branching.
- An infeasible LP relaxation, in which case we prune by infeasibility.
- An integral solution with PS-Tuple, in which case the following steps are executed:

(1) ISP1 is solved and the sub-tours are detected.

(2) Constraints (5.23) (SECs) are added to the current formulation as lazy cuts.

- A Hamiltonian path with precedence violations for at least one customer request, in which case the following steps are executed:

  (1) ISP2 is solved and node sets violating Constraints (4.7) (precedence) are detected.

  (2) Constraints (4.7) are added to the current formulation as lazy cuts.

- A precedence abiding Hamiltonian path with at least one LIFO violation, in which case the following steps are executed:

  (1) ISP3 is solved and node sets violating Constraints (4.8) (LIFO) are detected.

  (2) Constraints (4.8) are added to the current formulation as lazy cuts.

- A precedence abiding Hamiltonian path with no LIFO violations, in which case we update the incumbent solution accordingly.

An interesting aspect of this algorithm structure is the nested separation procedure. We start with assignment relaxation and once we have an integral solution, we solve ISP1 to add SECs as lazy cuts. From there, we identify a Hamiltonian path with precedence violations (integral solution for RP1). Then, we solve ISP2 to add precedence constraints as lazy cuts. As a result, we may identify an integral solution

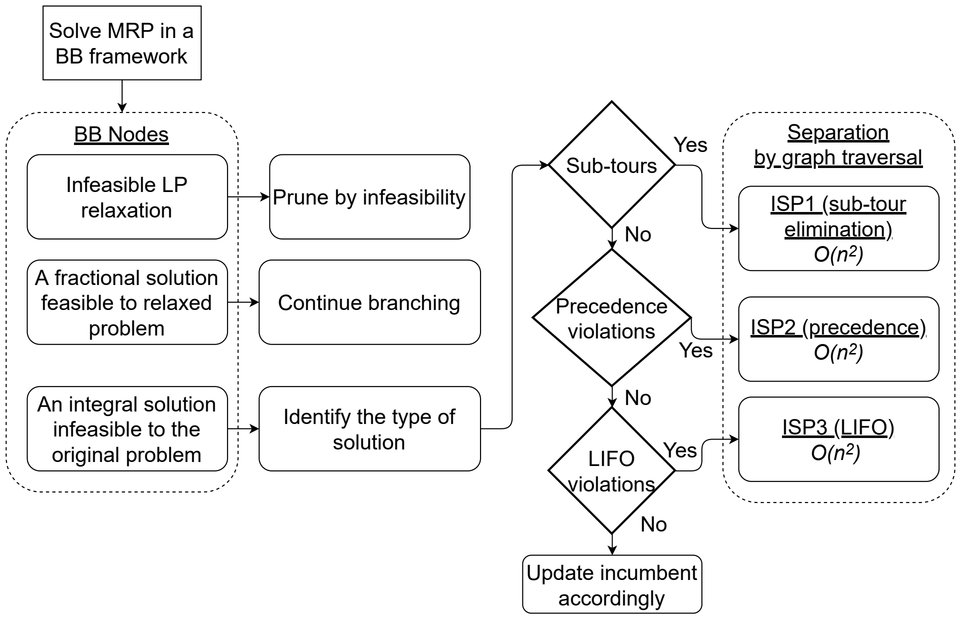


Figure 4.5: SPDPL-I algorithm structure

for RP2, at which point we solve ISP3 and find a feasible solution to our original problem. This implies that we solve one separation problem based on the output of another one.

The algorithm terminates when there are no nodes left in the BB tree to branch. At that point, the incumbent solution is an optimal vehicle route. A high-level illustration of our SPDPL-I algorithm structure is shown in Figure 4.5.

4.2.5 Other runtime improvements

In this section, we present other runtime improvements that we implemented in our branch-and-cut algorithms. Some of the preprocessing techniques and cut pool implementations were presented by Cordeau et al. [18].

Preprocessing

We remove the following arcs from the arc set (A) of the graph (G)

- Arcs of form $(n+i, i)$ $\forall i \in P$. This is because the pickup node cannot immediately succeed the delivery node for any customer request.

- Arcs of form $(0, n+i)$ $\forall n+i \in D$. This is because a delivery node cannot immediately succeed the origin depot in the vehicle path. On the similar note, we remove all arcs of form $(i, 2n+1)$ $\forall i \in P$. Furthermore, we also remove the arc $(2n+1, 0)$.

- Consider two pickup nodes $i, j \in P$ such that $i \neq j$. On a LIFO enforced route, $n+j$ cannot immediately succeed i. So, we remove all arcs of form $(i, n+j)$ $\forall i, j \in P$.

Cut pool

Before starting the algorithm, we include the following inequalities to the assignment formulation:

- Sub-tour elimination constraints (5.23) for all node sub-sets S such that $|S| = 2$.

- Incompatible successor (4.10) and predecessor inequalities (4.11) because they are quadratic in number.

- Incompatible arc set inequalities (4.12) because they are also quadratic in number.

- For each pickup node pair $i, j \in P$, successor inequalities (4.14) after setting node subset S to $\{i, j\}$, $\{i, n+j\}$ and $\{i, n+i, j\}$. Also, predecessor inequalities (4.13) after setting node subset S to $\{n+i, n+j\}$, $\{i, n+j\}$ and $\{i, n+i, n+j\}$.

- For each pickup node pair $i, j \in P$, D_k^+ inequality (4.15) for ordered set $S = \{n+i, j, i, n+j\}$ and D_k^- inequality (4.16) for ordered set $S = \{i, n+i, n+j, j\}$.

Warm start

We provide a feasible solution of SPDPL to the solver for a warm start. If our solution is better than the solvers' initial solution, then our solution will be used as a warm start. We developed a greedy heuristic to identify the warm start solution. The basic idea is to start with a feasible solution, and remove and insert nodes repeatedly such that LIFO loading order and precedence for pickups are not violated. We discuss more about our heuristic structure in the next section.

4.2.6 SPDPL warm start heuristic structure

Initial solution

We obtain a feasible SPDPL solution by performing the following steps.

(1) Scan all outgoing arcs from origin depot 0 and identify the lowest cost arc $(0, i)$, such that $i \in P$. This is because a delivery node cannot immediately succeed 0 in the solution.

(2) The possible successor of $i \in P$ in a LIFO enforced route is either $n + i$ or a pickup node different from i, because i is the last visited pickup node and

visiting any other delivery node except $n+i$ violates LIFO. So, we scan the possible successor set and select the lowest cost arc. The node at the tail end of the lowest cost arc is added to the path and the process is repeated.

(3) We continue by selecting the lowest cost arc (or one of the lowest cost arcs in case of a tie) from the possible successor set for each newly added node in the path. In any given iteration, the successor set is the delivery node corresponding to the last visited pickup node or a pickup node not already in the partial path. It follows that the precedence will be respected for all customer requests because a delivery node will not be selected unless the corresponding pickup node is already in the route.

(4) Repeat this process until all pickup and delivery nodes are in the route. Append destination depot $2n+1$ to the route.

Removal and insertion of nodes

After identifying an initial solution, for each pickup node $i \in P$, we remove i and $n+i$ from the path and insert them in all possible positions such that the precedence ($i \prec n+i$) and LIFO order is respected. After multiple removals and insertions, the path with lowest objective value is selected as our warm start solution. A simple example of our removal and insertion process is shown in Figure 4.6.

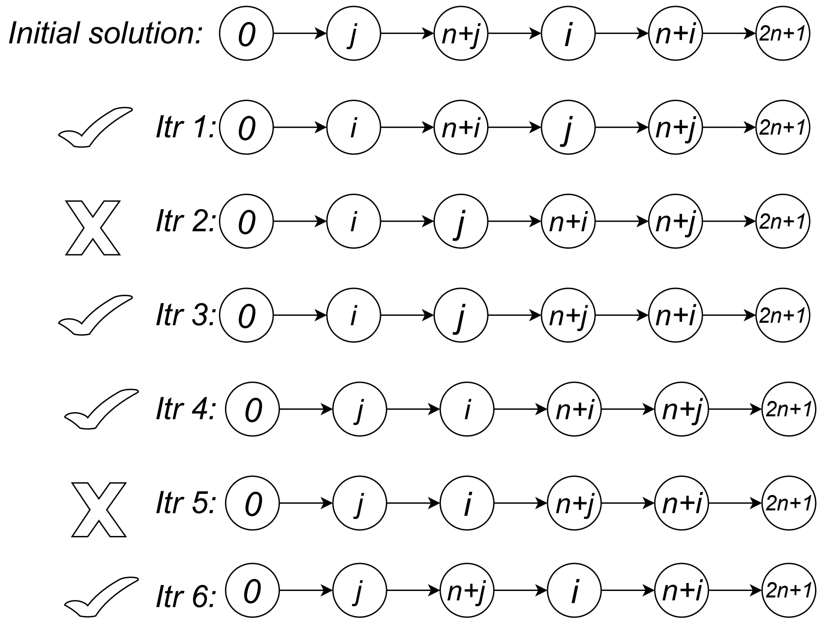


Figure 4.6: SPDPL warm start heuristic- nodes removal and insertion

The procedure for our example is described below:

(1) In iteration 1, i and $n+i$ are removed from the initial solution and inserted in positions 2 and 3 in the path. The path respects the LIFO order, so the objective value of the new path is calculated and the lowest cost is updated.

(2) In iteration 2, $n+i$ is removed from the iteration 1 path and inserted in position 4, whereas i is held in the same position. The new path violates the LIFO order. So it is not considered for update.

(3) In iteration 3, $n+i$ is removed from the iteration 2 path and inserted in position 5, whereas i is held in the same position. This path respects the LIFO order for all shipment requests. So it is considered for cost update.

(4) Precedence violations will not occur in any iteration because $n+i$ is inserted in positions after i. We continue this process until all possible positions for i and $n+i$ without precedence and LIFO order violations are explored. In our example, it took 6 iterations among which 4 were feasible solutions.

(5) Path positions for i and $n+i$ are fixed based on the lowest cost route from the aforementioned 4 solutions and we repeat the same process by removing and inserting j and $n+j$.

The algorithm structure for our warm start heuristic is shown in Appendix 2.7.

4.2.7 Upper bound tightening

In our SPDPL-I algorithm, we perform an upper bound tightening procedure when we update the incumbent solution. Let U_{new} be the objective of a new incumbent solution. We use the incumbent vehicle path as the initial solution and perform the removal and insertion of nodes as described in the previous section. Notice that there are no precedence violations after performing the removal and insertion procedure on the incumbent solution. Let U_{ri} be the objective value after a removal and insertion operation. If $U_{ri} < U_{new}$, then we add a lazy cut stating that the objective of the solution should be less than or equal to U_{ri}.

4.3 Single Vehicle Pickup-and-Delivery Problem with Handling Costs

A single-vehicle should serve all customer demands. The objective of SPDPH is to find a minimum cost Hamiltonian path from origin depot 0 to destination depot $2n+1$. A feasible route must satisfy the following conditions:

- For each shipment $i = 1, \ldots, n$, the pickup node $i \in P$ must be visited before the delivery node $n + i \in D$.
- Load picked from a location should be placed at the rear end of the vehicle (top of the stack).
- LIFO violation penalty (handling cost) is incurred once for each additional load handling operation at delivery nodes.

We assume that the reshuffling of cargo at delivery nodes is not permitted. So, additionally handled loads are reloaded back into the vehicle in the same order they were unloaded.

We explore three MIP models for SPDPH.

1. **SPDPH1** is a compact formulation presented by Veenstra et al. [42]. One of this dissertation objectives is to compare the computational performance of this formulation against our solution methodologies.
2. **SPDPH2** is a compact formulation introduced in this dissertation.
3. **SPDPH3** is a cut-based formulation with exponential number of constraints introduced in this dissertation. We also present two branch-and-cut algorithms to implement SPDPH3.

We use the graph structure and notations presented in Section 4.1 to introduce the formulations.

4.3.1 SPDPH Formulation 1 (SPDPH1)

The first formulation we discuss here was presented by Veenstra et al. [42].

Decision variables

f, x and z variables as defined in Section Table 4.1

Sets

A' as defined in Table 4.1

Formulation 4.3.1 (SPDPH1 by Veenstra et al. [42]).

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} + v\sum_{i\in P}\sum_{\substack{j\in P\\ j\neq i}} z_{ij} \tag{4.21}$$

subject to:

Constraints (4.2) - (4.5) *from Formulation 4.2.1*

$$\sum_{j:(i,j)\in A} f^1_{ijk} - \sum_{j:(j,i)\in A} f^1_{jik} = \begin{cases} 1, & \textit{if } i = 0 \\ -1, & \textit{if } i = k \\ 0, & \textit{otherwise} \end{cases} \qquad \forall i \in N, k \in P \tag{4.22}$$

$$\sum_{j:(i,j)\in A'} f^2_{ijk} - \sum_{j:(j,i)\in A'} f^2_{jik} = \begin{cases} 1, & \textit{if } i = k \\ -1, & \textit{if } i = n+k \\ 0, & \textit{otherwise} \end{cases} \qquad \forall i \in P\cup D, k \in P \tag{4.23}$$

$$\sum_{j:(i,j)\in A} f^3_{ijk} - \sum_{j:(j,i)\in A} f^3_{jik} = \begin{cases} 1, & \textit{if } i = n+k \\ -1, & \textit{if } i = 2n+1 \\ 0, & \textit{otherwise} \end{cases} \quad \forall i \in N, k \in P \tag{4.24}$$

$$f^1_{ijk} + f^3_{ijk} = x_{ij} \quad \forall (i,j) \in A \setminus A', k \in P \tag{4.25}$$

$$f^1_{ijk} + f^2_{ijk} + f^3_{ijk} = x_{ij} \quad \forall (i,j) \in A', k \in P \tag{4.26}$$

$$z_{ij} \geq \sum_{k:(k,j)\in A'} f^2_{k,j,i} - \sum_{k:(n+j,k)\in A'} f^2_{n+j,k,i} \quad \forall i,j \in P, i \neq j \tag{4.27}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A \tag{4.28}$$

$$z_{ij} \in \{0,1\} \quad \forall i,j \in P, i \neq j \tag{4.29}$$

$$f^1_{ijk}, f^3_{ijk} \in \{0,1\} \quad \forall (i,j) \in A, k \in P \tag{4.30}$$

$$f^2_{ijk} \in \{0,1\} \quad \forall (i,j) \in A', k \in P \tag{4.31}$$

The objective function (4.21) minimizes the total transportation and handling cost. Constraints (4.2) - (4.5) are degree and capacity constraints. Constraints (4.22) ensure a path from origin depot 0 to each pickup node. Constraints (4.23) ensure a path from pickup node to delivery node for each customer request. Constraints (4.24) ensure a path from each delivery node to destination depot $2n+1$. Constraints (4.25) and (4.26) link flow variables (f) with arc variables (x). Constraints (4.27) enforce a penalty, if node $j \in P$ is in the route between i and $n+i$, and $n+j$ is not in the route between i and $n+i$.

4.3.2 SPDPH Formulation 2 (SPDPH2)

The second formulation is a compact model introduced in this dissertation.

Decision variables

$\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$ variables as defined in Table 4.1

Formulation 4.3.2 (SPDPH2)**.**

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} + v\sum_{i\in P}\sum_{\substack{j\in P\\ j\neq i}} z_{ij} \tag{4.32}$$

subject to:

Constraints (4.2) - (4.5) *from Formulation 4.2.1*

$$y_{i,n+i} = 1 \quad \forall i \in P \tag{4.33}$$

$$y_{ij} \geq x_{ij} \quad \forall (i,j) \in A \tag{4.34}$$

$$y_{ij} + y_{ji} = 1 \quad \forall i,j \in N, i \neq j \tag{4.35}$$

$$y_{ij} + y_{jk} + y_{ki} \leq 2 \quad \forall i,j,k \in N, i \neq j \neq k \tag{4.36}$$

$$z_{ij} \geq y_{ij} + y_{n+i,n+j} + y_{j,n+i} - 2 \quad \forall i,j \in P, i \neq j \tag{4.37}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A \tag{4.38}$$

$$0 \leq y_{ij} \leq 1 \quad \forall i,j \in N, i \neq j \tag{4.39}$$

$$0 \leq z_{ij} \leq 1 \quad \forall i,j \in P, i \neq j \tag{4.40}$$

The objective function (4.32) calls for the minimization of the total transportation and handling cost. Constraints (4.2) - (4.5) are degree and capacity constraints. Constraints (4.33) are precedence constraints ensuring that pickup is visited before delivery for all customer requests. Constraints (4.34)-(4.36) are Sub-Tour Elimination Constraints (SEC) introduced by Sarin et al. [38] for Precedence Constrained Asymmetric Traveling Salesman Problem. Constraints (4.37) ensure that $z_{ij} = 1$ for customer requests i and j, if:

- j is visited after i
- $n + i$ is visited after j and
- $n + j$ is visited after $n + i$

4.3.3 SPDPH Formulation 3 (SPDPH3)

Our third formulation is a cut-based MIP with exponential number of constraints.

Decision variables

x and z variables as described in Table 4.1

Sets

Γ and Υ_j as defined in Table 4.1

Formulation 4.3.3 (SPDPH3)**.**

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} + v \sum_{i\in P} \sum_{\substack{j\in P \\ j\neq i}} z_{ij} \tag{4.41}$$

subject to:

Constraints (4.2) - (4.7) *from Formulation 4.2.1*

$$z_{ij} \geq \big[x(i,S) + x(A(S)) + x(S,n+i)\big] - |S| \quad \forall S \in \Upsilon_j, \forall i, n+i \notin S, \forall i,j \in P \tag{4.42}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A \tag{4.43}$$

$$0 \leq z_{ij} \leq 1 \quad \forall i,j \in P \tag{4.44}$$

The objective function (4.41) seeks to minimize the total transportation and handling cost. Constraints (4.2) - (4.7) are degree constraints, SECs, precedence and vehicle capacity constraints for PDP which have been discussed before in Formulation 4.2.1. Constraints (4.42) enforce handling cost in case of LIFO loading order violation as discussed below.

Let us consider two customer requests $i \in P$ and $j \in P \setminus \{i\}$. Also, consider a node subset $S \subset P \cup D$, such that $j \in S$ and $i, n+i, n+j \notin S$. Note that $S \in \Upsilon_j$ (by set definition). Let x^* and z^* be a feasible solution to SPDPH3. Note that:

- $x^*(i,S) \leq 1$ (maximum out-degree of i- Constraints (4.2))
- $x^*(S,n+i) \leq 1$ (maximum in-degree of $n+i$- Constraints (4.3))
- $x^*(A(S)) \leq |S|-1$ (sub-tour elimination- Constraints (5.23))

Now, z^*_{ij} will assume a value of 1 only if S was selected such that $x^*(i,S) = 1$, $x^*(S,n+i) = 1$ and $x^*(A(S)) = |S|-1$. As illustrated in Figure 4.7, this would mean that

- j is in the path between i and $n+i$
- $n+j$ is not in the path between i and $n+i$

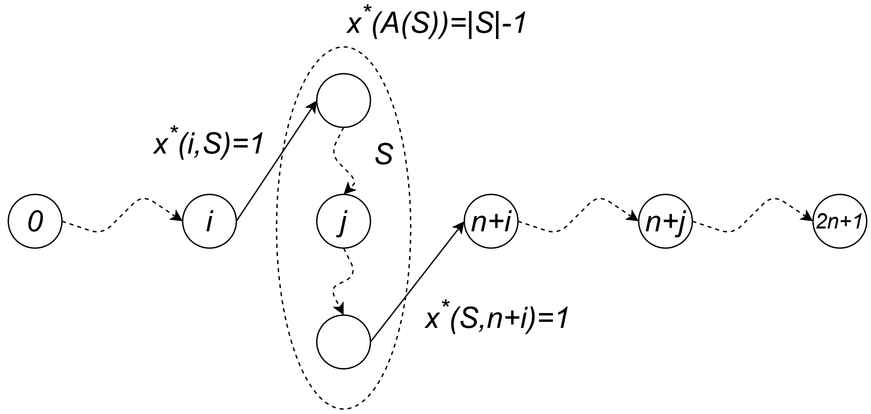


Figure 4.7: An illustration of handling cost constraints

The aforementioned node subset structure is the only case where z^*_{ij} can be 1. Any other node subset in Υ_j will result in the right hand side of equation (4.42) yielding values lesser than equal to zero, hence trivializing the constraints.

Inequalities for SPDPH3

As mentioned in the literature review, SPDPH is a relatively new problem with very sparse literature. So, there are no existing inequalities for SPDPH specifically available in the literature. However, precedence constrained ATSP inequalities are applicable for SPDPH. Therefore, inequalities (4.13)-(4.16) are applicable for SPDPH. In this dissertation, we present the following inequalities for SPDPH.

Handling cost enforcing arc pair inequalities

We present this family of inequalities based on the following notion. For each node pair $i, j \in P$, handling cost has to be enforced if two of the following arcs are in the vehicle route: (i, j), $(j, n+i)$ and $(n+i, n+j)$. This is because it would mean j is in the path between i and $n+i$, and $n+j$ is not in the path between i and $n+i$ as shown in Figure 4.8.

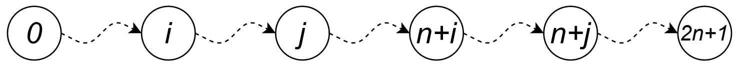


Figure 4.8: An illustration HC enforcing arc pair inequalities

Proposition 1. *For each node pair $i, j \in P$, the following inequalities are valid for SPDPH3*

$$z_{ij} \geq x_{ij} + x_{j,n+i} - 1 \tag{4.45}$$

$$z_{ij} \geq x_{j,n+i} + x_{n+i,n+j} - 1 \tag{4.46}$$

$$z_{ij} \geq x_{ij} + x_{n+i,n+j} - 1 \tag{4.47}$$

Proof: Suppose that $C \subseteq A$ represents a Hamiltonian path from node 0 to $2n+1$ that satisfies precedence for all customer requests ($i \prec n+i$ and $j \prec n+j$). Let X^c be its characteristic vector, and suppose

$$z_{ij}^c = \begin{cases} 1, \text{if } j \in P \text{ is picked up between } i \in P \text{ and } n+i \in D\text{, and delivered after } n+i \\ 0, \text{otherwise} \end{cases}$$

we want to show

$$z_{ij}^c \geq x_{ij}^c + x_{j,n+i}^c - 1 \tag{4.48}$$

$$z_{ij}^c \geq x_{j,n+i}^c + x_{n+i,n+j}^c - 1 \tag{4.49}$$

$$z_{ij}^c \geq x_{ij}^c + x_{n+i,n+j}^c - 1 \tag{4.50}$$

Inequality (4.48): This can be obtained by setting $S = \{j\}$ in Constraints (4.42).

Inequality (4.49): By the definition of z^c_{ij}, we know that $z^c_{ij} = 1$, if and only if j is in the path between i and $n+i$, and $n+j$ is not in the path between i and $n+i$. This in turn means $z^c_{ij} = 1$, if and only if:

1. $i \prec j$
2. $j \prec n+i$ and
3. $n+i \prec n+j$

Notice that the above conditions hold, if $x^c_{j,n+i} = x^c_{n+i,n+j} = 1$ because

- $x^c_{j,n+i} = 1 \implies j \prec n+i$
- $x^c_{n+i,n+j} = 1 \implies n+i \prec n+j$ and
- $i \prec n+i$ (precedence for customer requests) and $x^c_{j,n+i} = 1 \implies i \prec j$

Therefore,

$z^c_{ij} = 1$, if $x^c_{j,n+i} = x^c_{n+i,n+j} = 1 \implies z^c_{ij} \geq x^c_{j,n+i} + x^c_{n+i,n+j} - 1$.

Inequality (4.50): Notice that conditions $1-3$ also hold, if $x^c_{ij} = x^c_{n+i,n+j} = 1$ because

- $x^c_{ij} = 1 \implies i \prec j$
- $x^c_{n+i,n+j} = 1 \implies n+i \prec n+j$ and
- $x^c_{ij} = 1$ and $i \prec n+i$ (precedence for customer requests) $\implies j \prec n+i$

Therefore,

$z_{ij}^{c} = 1$, if $x_{ij}^{c} = x_{n+i,n+j}^{c} = 1 \implies z_{ij}^{c} \geq x_{ij}^{c} + x_{n+i,n+j}^{c} - 1$.

■

Size comparison of SPDPH formulations

Table 4.3 shows the number of constraints and variables for the three SPDPH formulations in terms of the number of customer requests n. To reiterate the formulation labels, SPDPH1 is a flow-based formulation proposed by Veenstra et al. [42], SPDPH2 is a compact formulation and SPDPH3 is a cut-based formulation with an exponential number of constraints.

Table 4.3: SPDPH formulations size comparison

Formulation	#Variables	#Constraints
SPDPH1	$2n^3 + 5n^2$	$3n^3 + 15n^2 + 6n$
SPDPH2	$6n(n+1) + 2$	$11n^3 + 19n^2 + 18n$
SPDPH3	$2n^2$	Exponential

Branch-and-cut algorithms

We present two branch-and-cut algorithms to solve SPDPH. Formulation SPDPH3 has exponentially many sub-tour elimination Constraints (5.23), precedence Constraints (4.7) and handling cost Constraints (4.42). So, similar to SPDPL Formulation 4.2.1, we solve this problem with two branch-and-cut algorithms: one with fractional separation procedures and another with integral separation procedures.

Branch-and-cut algorithm - Fractional separation (SPDPH3-F)

The algorithm structure is similar to the branch-and-cut approach discussed in Section 4.2.3. We start by identifying a solution feasible to MRP-F. After that, we solve three fractional separation problems and sequentially add cutting planes. Two of the fractional separation problems (FSP1 for sub-tour elimination and FSP2 for precedence enforcement) are similar to SPDPL-F. Therefore, we solve FSP1 and FSP2 as described in Section 4.2.3. However, the third separation problem (handling cost) is slightly different from SPDPL-F.

Fractional Separation Problem 3 (FSP3H)- Handling cost

Input: A directed graph $G = (N, A)$ as described in Section 4.1, fractional values $x^* \in F$ and handling costs z^*.

Problem: For each pair of nodes $i, j \in P$, identify a node subset $S^* \subset N$, such that $j \in S^*$, $n+j \notin S^*$, $i \notin S^*$, $n+i \notin S^*$, and $[x(i, S^*)+x(A(S^*))+x(S^*, n+i)]-|S^*| > z^*_{ij}$, or determine that no such set exists.

We solve FSP3H as follows. FSP3 (LIFO violation- fractional separation problem) is solved for pair of customer requests $i, j \in P$ as shown in Section 4.2.3. We do this to identify a LIFO violating node set $S^* \in \Upsilon_j$ and check if $[x(i, S^*) + x(A(S^*)) + x(S^*, n + i)] - |S^*| > z^*_{ij}$. If yes, then we add inequality (4.42) for S^* as a cutting plane.

Branch-and-cut algorithm - Integral separation (SPDPH3-I)

The algorithm structure is similar to the branch-and-cut approach discussed in Section 4.2.4. We start by identifying a solution feasible to MRP-I. After that, we solve three separation problems and add lazy cuts. Two of the separation problems (ISP1 for sub-tour elimination and ISP2 for precedence enforcement) are similar to SPDPL-I. Therefore, we solve ISP1 and ISP2 as described in Section 4.2.4. However, the third separation problem (handling cost) is slightly different from SPDPL-I.

Integral Separation Problem 3 (ISP3H)- Handling cost

Input: A directed graph $G = (N, A)$ as described in Section 4.1, binary values $x^* \in I_3$ and handling costs z^*.

Problem: For each node pair $i, j \in P$, $i \neq j$, identify a node subset $S^* \subset P \cup D$, such that $i \notin S^*$, $n+i \notin S^*$, $j \in S^*$, $n+j \notin S^*$, $[x(i, S^*)+x(A(S^*))+x(S^*, n+i)]-|S^*| > z^*_{ij}$ or determine that no such set exists.

ISP3H is defined in a Hamiltonian path with no precedence violations. We handle this problem by solving ISP3 (LIFO violation- integral separation problem) for node pair $i, j \in P$ as shown in Section 4.2.4. If we identify a LIFO violating node set $S^* \in \Upsilon_j$, then we check if $[x(i, S^*) + x(A(S^*)) + x(S^*, n + i)] - |S^*| > z^*_{ij}$. If yes, then we add inequality (4.42) for S^* as a lazy cut. A high-level illustration of our SPDPH3-I algorithm structure is shown in Figure 4.9.

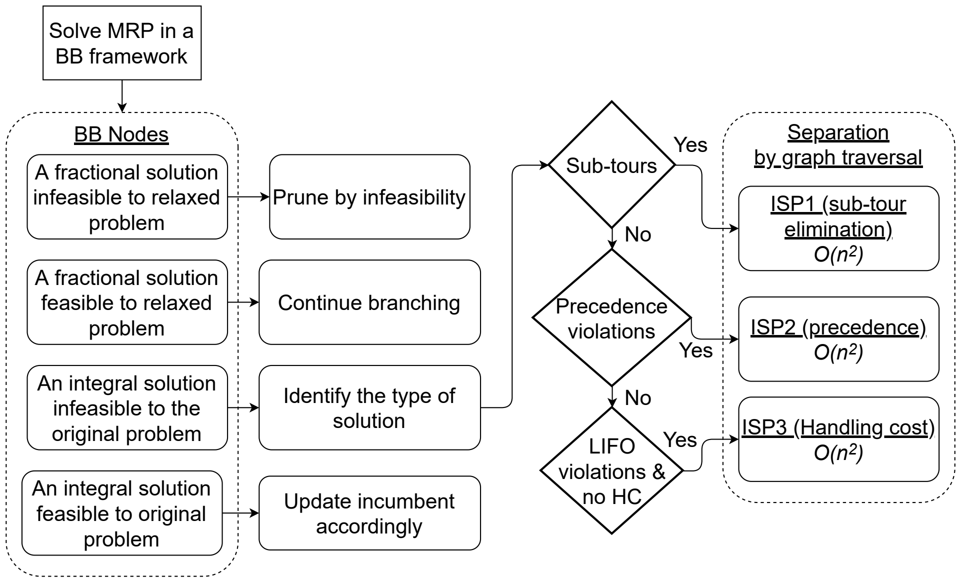


Figure 4.9: SPDPH3-I algorithm structure

4.3.4 Other runtime improvements

In this section, we present other runtime improvements that we implemented for SPDPH branch-and-cut algorithms.

Preprocessing

We remove the following arcs from the arc set (A) of the graph (G)

- Arcs of form $(n+i, i)$ $\forall i \in P$. This is because the pickup node cannot immediately succeed the delivery node for any customer request.

- Arcs of form $(0, n+i)$ $\forall n+i \in D$. This is because a delivery node cannot immediately succeed the origin depot in the vehicle path. On the similar note, we remove all arcs of form $(i, 2n+1)$ $\forall i \in P$. Furthermore, we also remove the arc $(2n+1, 0)$.

Cut pool

Before starting the algorithm, we include the following inequalities to the assignment formulation:

- Sub-tour elimination constraints (5.23) for all node sub-sets S such that $|S| = 2$.
- For each pickup node pair $i, j \in P$, successor inequalities (4.14) after setting node subset S to $\{i, j\}$, $\{i, n+j\}$ and $\{i, n+i, j\}$. Also, predecessor inequalities (4.13) after setting node subset S to $\{n+i, n+j\}$, $\{i, n+j\}$ and $\{i, n+i, n+j\}$.
- For each pickup node pair $i, j \in P$, we add D_k^+ inequality (4.15) for ordered set $S = \{n+i, j, i, n+j\}$ and D_k^- inequality for ordered set $S = \{i, n+i, n+j, j\}$.
- Handling cost enforcing arc pair inequalities (5.25)-(5.27) for each node pair $i, j \in P$.
- For each pickup node pair $i, j \in P$, we add Constraints (4.42) after setting $S = \{j, k\}$, for all $k \in N \setminus \{i, j, n+i, n+j, 0, 2n+1\}$.

Warm start

We developed a greedy heuristic to identify a warm start solution for our branch-and-cut algorithms. The basic idea is to start with an infeasible solution, and remove and insert nodes repeatedly until we find a reasonably good feasible solution. We discuss more about our heuristic structure in the next section.

4.3.5 SPDPH warm start heuristic structure

Initial solution

We obtain an initial solution which could be infeasible to SPDPH by implementing the savings algorithm presented in the seminal paper by Clark and Wright [15]. This algorithm seeks to find vehicle routes for VRP with capacity constraints. We modify this for a single vehicle problem as shown in Appendix 2.6.

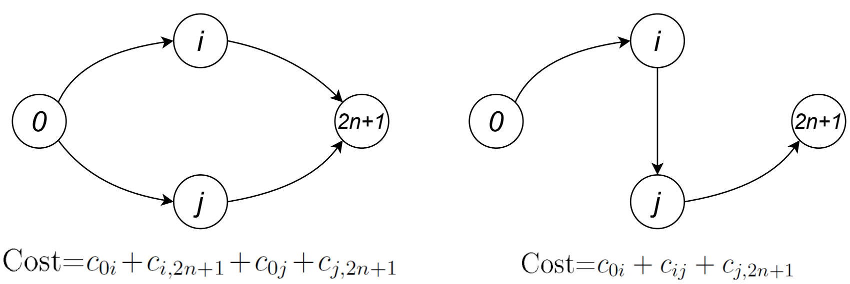


Figure 4.10: Cost incurred by two routes

The savings algorithm is based on the following premise. Without loss of generality, assume that multiple vehicles are available at origin depot 0 and our task is to visit customers i and j. A novice decision might be to visit i and j using two separate vehicles as shown in Figure 4.10 (left). The total cost for this strategy is $c_{0i} + c_{i,2n+1} + c_{0j} + c_{j,2n+1}$. However, if we choose to travel on arc (i, j) as shown in Figure 4.10 (right), then the total cost would be $c_{0i} + c_{ij} + c_{j,2n+1}$. So, the savings s_{ij} from combining two customers in a single truck and traveling on arc (i, j) is $c_{i,2n+1} + c_{0j} - c_{ij}$ (difference between the aforementioned route costs). The idea is to calculate savings for each arc and construct a solution so that the total savings is maximized. We obtain our initial solution with the following steps:

1. Calculate savings s_{ij} for each arc $(i, j) \in A$ such that $i, j \in P \cup D$ and arrange them in a list by descending order.

2. Scan each arc (i, j) in the list starting from the top.

   - If i and j are not in any vehicle path, then create a new vehicle path with i and j as endpoints.
   - If i is at one end of a vehicle path and j is not in any vehicle path, then append j to that endpoint and vice versa.
   - If node i or j is in the middle of a vehicle path, then ignore the arc and move to the next one.
   - If i is at one end of a vehicle path and j is at one end of another vehicle path, then append the two paths together by merging the endpoints (if necessary, reverse a vehicle path to make the endpoints meet).

3. Repeat step 2 until all pickup and delivery nodes are in a single-vehicle path.

4. Note that the path should start at 0 and end at $2n + 1$. So, append 0 to the beginning of the path and $2n + 1$ to the end of the path.

The initial solution might have precedence violations for some customer requests. So, it might not be a feasible solution to SPDPH. However, we remove and insert nodes repeatedly from this initial solution until we obtain a good feasible solution to SPDPH.

Removal and insertion of nodes

The removal and insertion procedure is similar to that of SPDPL heuristic discussed in Section 4.2.6, except for the following differences:

- Routes violating LIFO loading/unloading are considered.
- Incumbent route updates are done based on objective value calculation which includes handling costs.
- A route will be considered for the incumbent update only if it does not violate precedence for any customer request.

Even though we might start with an infeasible initial solution, the final route after removal and insertion of nodes will be feasible to SPDPH. This is because the insertion procedure in SPDPL heuristic is performed in such a way that the pickup node will precede the delivery node for each customer request. Therefore, the precedence constraints will not be violated for any customer request.

CHAPTER V

MULTI VEHICLE PROBLEMS

In this chapter, we present our formulations and methodologies for multi-vehicle problems. As mentioned in Chapter III, we address two problems under the multi-vehicle category: Pickup-and-Delivery Problem with Time windows and Loading constraints (MPDPTL) and Pickup-and-Delivery Problem with Time windows and Handling costs (MPDPTH).

5.1 Notations

We use the same graph structure as mentioned in Section 4.1.1. However, in addition to the single-vehicle problem notations, we introduce new notations in this chapter. For readers' convenience, a centralized table for decision variables, set definitions, and other notations has been created (Table 5.1). We refer to this table from the formulations as per necessity.

Table 5.1: Notations for MIP models- Multi vehicle

Type	Notation	Definition
Set definitions	K	Set of homogeneous (vehicles with same capacity) fleet of vehicles
Decision variables	B_i^k	Time by which vehicle $k \in K$ begins service at node $i \in N$
	f_{ijl}^{1k}	Equal to 1 if arc $(i, j) \in A$ is in the path of vehicle k from node 0 to node l; 0 otherwise
	f_{ijl}^{2k}	Equal to 1 if arc $(i, j) \in A$ is in the path of vehicle k from node l to node $n + l$; 0 otherwise
	f_{ijl}^{3k}	Equal to 1 if arc $(i, j) \in A$ is in the path of vehicle k from node $n + l$ to node $2n + 1$; 0 otherwise
	Q_i^k	Load on vehicle $k \in K$ upon leaving node $i \in N$
	u_{ij}^k	Load carried by vehicle $k \in K$ on arc $(i, j) \in A$
	x_{ij}^k	Equal to 1 if arc $(i, j) \in A$ is in the route of vehicle k and 0 otherwise
	z_{ij}^k	Equal to 1 if $j \in P$ is picked up between $i \in P$ and $n + i \in D$, and delivered after $n + i$ by vehicle k; 0 otherwise
Time factors	a_i	Earliest time at which service can start at node $i \in N$
	b_i	Latest time at which service can start at node $i \in N$
	t_{ij}	Travel duration on arc $(i, j) \in A$
Other	$x^k(A(S))$	$\sum_{i,j \in S} x_{ij}^k$ for vehicle $k \in K$
	$x^k(i, S)$	$\sum_{j \in S} x_{ij}^k$ for vehicle $k \in K$
	$x^k(S, i)$	$\sum_{j \in S} x_{ji}^k$ for vehicle $k \in K$

For origin depot, a_0 and b_0 each represents earliest and latest times at which vehicles can leave respectively. Similarly, a_{2n+1} and b_{2n+1} each represents earliest and latest time for vehicle arrival at the destination depot respectively. We assume that $q_0 = q_{2n+1} = 0$, and $q_i = -q_{n+i}$ for each request $i \in P$.

5.2 Multi Vehicle Pickup-and-Delivery Problem with Time Windows and Handling Costs

In this section, we: (1) Present a compact formulation and a cut-based formulation for MPDPTH which we denote as MPDPTH-C and MPDPTH-E respectively; (2) Explore a BC algorithm for MPDPTH-E; (3) Explore runtime improvements including families of inequalities and a warm start heuristic, which turns out be very efficient. The objective of MPDPTH is to find minimum cost Hamiltonian path(s) from origin depot 0 to destination depot $2n+1$. A feasible solution must satisfy the following conditions:

- Each node $i \in P \cup D$ should be visited exactly once by one vehicle.
- For each customer request, the pickup and delivery must be visited by the same vehicle.
- The number of routes must not exceed the number of available vehicles.
- For each shipment and vehicle, the pickup node must be visited before the delivery node.
- Each node can be visited only within the associated time window.
- For each vehicle $k \in K$, the capacity should not exceed Q on any arc.

- Handling cost must be imposed for additional cargo handling at delivery points (LIFO violation).

5.2.1 Compact formulation

In this section, we present our compact formulation with a polynomial number of constraints for MPDPTH. Our formulation is the multiple vehicle extension of SPDPH formulation introduced by Veenstra et al. [42].

Decision variables

B, f, Q, x and z variables as defined in Table 5.1

Sets

A' as defined in Table 5.1

Formulation 5.2.1 (MPDPTH-C)**.**

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ij}^k + h \sum_{k \in K} \sum_{i \in P} \sum_{\substack{j \in P \\ j \neq i}} z_{ij}^k \tag{5.1}$$

subject to:

$$\sum_{k \in K} \sum_{j:(i,j) \in A} x_{ij}^k = 1 \quad \forall i \in P \tag{5.2}$$

$$\sum_{j:(i,j) \in A} x_{ij}^k - \sum_{j:(n+i,j) \in A} x_{n+i,j}^k = 0 \quad \forall i \in P, k \in K \tag{5.3}$$

$$Q_j^k \geq (Q_i^k + q_j) x_{ij}^k \quad \forall (i,j) \in A, \forall k \in K \tag{5.4}$$

$$max\{0, q_i\} \leq Q_i^k \leq min\{Q, Q + q_i\} \quad \forall i \in N, \forall k \in K \tag{5.5}$$

$$B_j^k \geq (B_i^k + t_{ij}) x_{ij}^k \quad \forall (i,j) \in A, \forall k \in K \tag{5.6}$$

$$B_i^k + t_{i,n+i} \leq B_{n+i}^k \quad \forall i \in P, \forall k \in K \tag{5.7}$$

$$a_i \leq B_i^k \leq b_i \quad \forall i \in N, \forall k \in K \tag{5.8}$$

$$\sum_{k \in K} \sum_{j:(i,j) \in A} f_{ijl}^{1k} - \sum_{k \in K} \sum_{j:(j,i) \in A} f_{jil}^{1k} = \begin{cases} 1, & \mathit{if}\ i = 0 \\ -1, & \mathit{if}\ i = l \\ 0, & otherwise \end{cases} \qquad \forall i \in N, l \in P \tag{5.9}$$

$$\sum_{k \in K} \sum_{j:(i,j) \in A'} f_{ijl}^{2k} - \sum_{k \in K} \sum_{j:(j,i) \in A'} f_{jil}^{2k} = \begin{cases} 1, & \mathit{if}\ i = l \\ -1, & \mathit{if}\ i = n + l \\ 0, & otherwise \end{cases} \qquad \forall i \in P \cup D, l \in P \tag{5.10}$$

$$\sum_{k \in K} \sum_{j:(i,j) \in A} f_{ijl}^{3k} - \sum_{k \in K} \sum_{j:(j,i) \in A} f_{jil}^{3k} = \begin{cases} 1, & \mathit{if}\ i = n + l \\ -1, & \mathit{if}\ i = 2n + 1 \\ 0, & otherwise \end{cases} \qquad \forall i \in N, l \in P \tag{5.11}$$

$$nx_{ij}^k \geq \sum_{l \in P} f_{ijl}^{1k} + \sum_{l \in P} f_{ijl}^{3k} \quad \forall (i,j) \in A \setminus A', k \in K \tag{5.12}$$

$$nx_{ij}^k \geq \sum_{l \in P} f_{ijl}^{1k} + \sum_{l \in P} f_{ijl}^{2k} + \sum_{l \in P} f_{ijl}^{3k} \quad \forall (i,j) \in A', k \in K \tag{5.13}$$

$$z_{ij}^k \geq \sum_{l:(l,j) \in A'} f_{l,j,i}^{2k} - \sum_{l:(n+j,l) \in A'} f_{n+j,l,i}^{2k} \quad \forall i,j \in P, i \neq j, k \in K \tag{5.14}$$

$$f_{ijl}^{1k}, f_{ijl}^{3k} \in \{0,1\} \quad \forall (i,j) \in A, l \in P, k \in K \tag{5.15}$$

$$f_{ijl}^{2k} \in \{0,1\} \quad \forall (i,j) \in A', l \in P, k \in K \tag{5.16}$$

$$x_{ij}^k \in \{0,1\} \quad \forall (i,j) \in A, k \in K \tag{5.17}$$

$$z_{ij}^k \in \{0,1\} \quad \forall i,j \in P, i \neq j, k \in K \tag{5.18}$$

Objective function and constraints

The objective function seeks to minimize the total transportation and handling cost. Constraints (5.2) ensure that each customer request is served by exactly one vehicle. Constraints (5.3) ensure that the pickup and delivery for each customer request is serviced by the same vehicle. Constraints (5.4) and (5.5) satisfy capacity restrictions for all vehicles. Constraints (5.6) and (5.8) satisfy time window restrictions for each node. Precedence requirement that the pickup node has to be visited before the delivery node for each customer request is enforced by Constraints (5.7). Constraints (5.9) ensure a path from origin depot 0 to each pickup node. Constraints (5.10) ensure a path from pickup node to delivery node for each customer request. Constraints (5.11) ensure a path from each delivery node to destination depot $2n+1$. Constraints (5.12) and (5.13) link flow variables with arc variables for each vehicle. Constraints (5.14) enforce a handling cost for vehicle $k \in K$, if node $j \in P$ is in the route between i and $n+i$, and $n+j$ is not in the route between i and $n+i$. Note that Constraints (5.4) and (5.6) are non-linear. However, we can linearize the product of two variables using some standard linearization techniques as presented in Appendix A, but with an additional index for each load and arc variable corresponding to vehicle $k \in K$. Except for Constraints (5.9) - (5.16), remainder of the formulation was proposed by Ropke et al. [36] for PDP with time windows. We introduce Constraints (5.9) - (5.16) as the multiple vehicle extension of PDTSPH formulation introduced by Veenstra et al. [42].

5.2.2 Cut-based formulation

In this section, we present our cut-based formulation with an exponential number of constraints. We remove the flow variables (f) from Formulation MPDPTH-C for our cut-based formulation and replace them with an exponential number of constraints.

Formulation 5.2.2 (MPDPTH-E)**.**

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ij}^k + h \sum_{k \in K} \sum_{i \in P} \sum_{\substack{j \in P \\ j \neq i}} z_{ij}^k \tag{5.19}$$

subject to:

Constraints (5.2) - (5.8)*,* (5.17) *and* (5.18) *from Formulation MPDPTH-C*

$$\sum_{j:(0,j) \in A} x_{0j}^k = 1 \quad \forall k \in K \tag{5.20}$$

$$\sum_{j:(j,i) \in A} x_{ji}^k - \sum_{j:(i,j) \in A} x_{ij}^k = 0 \quad \forall i \in P \cup D, k \in K \tag{5.21}$$

$$\sum_{j:(j,2n+1) \in A} x_{j,2n+1}^k = 1 \quad \forall k \in K \tag{5.22}$$

$$x^k(A(S)) \leq |S| - 1 \quad \forall S \subseteq P \cup D, 2 \leq |S| \leq |N|, k \in K \tag{5.23}$$

$$z_{ij}^k \geq \left[x^k(i,S) + x^k(A(S)) + x^k(S, n+i) \right] - |S| \quad \forall S \in \Upsilon_j, \forall k \in K, \forall i, n+i \notin S, \forall i,j \in P \tag{5.24}$$

Constraints (5.23) are the well-known Dantzig-Fulkerson-Johnson (DFJ) sub-tour elimination constraints for each vehicle $k \in K$. LIFO violations are penalized with handling cost h with Constraints (5.24). These constraints are the multi-vehicle version of the SPDPH handling cost constraints from Formulation 4.3.3, which in turn were obtained by penalizing LIFO constraints presented by Cordeau et al. [18].

5.2.3 Families of inequalities

In this section, we present the inequalities that we used in our implementations for reducing the runtime.

Multi Vehicle handling cost enforcing arc pair inequalities

We introduce a new family of inequalities which is the multi vehicle variation of the handling cost enforcing arc pair inequalities for SPDPH introduced in Section 4.3.3. For each pickup node pair $i, j \in P$, handling cost has to enforced for vehicle $k \in K$, if at least two of the following three arcs are on the vehicle path: (i, j), $(j, n+i)$ and $(n+i, n+j)$. This means that j is between i and $n+i$, and $n+j$ is not between i and $n+i$ on the path of vehicle k. With that notion, we present the following inequalities for each pickup node pair $i, j \in P$ and vehicle $k \in K$.

$$z_{ij}^k \geq x_{ij}^k + x_{j,n+i}^k - 1 \tag{5.25}$$

$$z_{ij}^k \geq x_{j,n+i}^k + x_{n+i,n+j}^k - 1 \tag{5.26}$$

$$z_{ij}^k \geq x_{ij}^k + x_{n+i,n+j}^k - 1 \tag{5.27}$$

The proof of validity of these inequalities for SPDPH is the same as the proof presented in Section 4.3.3, but with one additional index on variables for vehicle k.

Symmetry breaking inequalities

In multi-vehicle routing problem with a homogeneous fleet, there is a symmetry issue that could increase the runtime. The route assigned to a vehicle can be swapped with any other vehicle in the fleet. So, there are $|K|!$ options to swap the routes assigned to a vehicle in the fleet. For example, customer requests $i, j \in P$ being assigned to vehicle $k \in K$ is equivalent to $|K|-1$ other solutions in which the same customer requests are assigned to other vehicles in the fleet. This could increase the runtime of our implementation due to search in a space of equivalent solutions. Sherali and Smith [39] discussed how such symmetry issues could increase the runtime in branch-and-bound implementations. Coelho et al. [16] and Adulyasak et al. [1] presented symmetry breaking inequalities for homogeneous fleet assignment in Multi-Vehicle Inventory Routing Problem (MVIRP) where vehicle routing and inventory management are solved as a single model. We use the following inequalities which are similar to their symmetry breaking inequalities. We assume that the vehicles are indexed from 1 to $|K|$ and use the following inequalities for each customer request $i \in P$.

$$\sum_{k=1}^{\min(i,|K|)} \sum_{j=0}^{2n} x_{ji}^{k} = 1 \tag{5.28}$$

Let us consider customer request 1 as an example for which the inequality is $\sum_{j=0}^{2n} x_{j1}^{1} = 1$. We eliminate the complexity of request 1 being assigned to $k-1$ other vehicles by assigning that request to vehicle 1. Similarly, request 2 can be assigned to vehicle 1 or 2 by the inequality $\sum_{j=0}^{2n} x_{j2}^{1} + \sum_{j=0}^{2n} x_{j2}^{2} = 1$. Therefore, we eliminate the complexity of request $\boldsymbol{2}$ being assigned to $\boldsymbol{k-2}$ other vehicles. Similarly, inequalities (5.28) assign a customer request $\boldsymbol{i} \in \boldsymbol{P}$ to a vehicle only if the vehicle index $\boldsymbol{k}$ is smaller than or equal to $\boldsymbol{i}$. By doing this, we eliminate the complexity of assigning request $\boldsymbol{i}$ to vehicles with $\boldsymbol{k} > \boldsymbol{i}$. If $\boldsymbol{i} \geq |K|$, then request $\boldsymbol{i}$ is assigned to one of the vehicles in the fleet without any restriction.

5.2.4 Preprocessing

We rearrange the customer requests before building the model such that the geographically proximal pickup locations are listed far from each other. We do this to maximize the potential of symmetry breaking inequalities. As Figure 5.1 illustrates, arranging the customer requests such that proximal pickup locations are listed to each other could reduce the potential of symmetry breaking and cluster consecutive pickup locations in a single vehicle. This preprocessing technique turned out to be very effective when used in tandem with symmetry breaking inequalities.

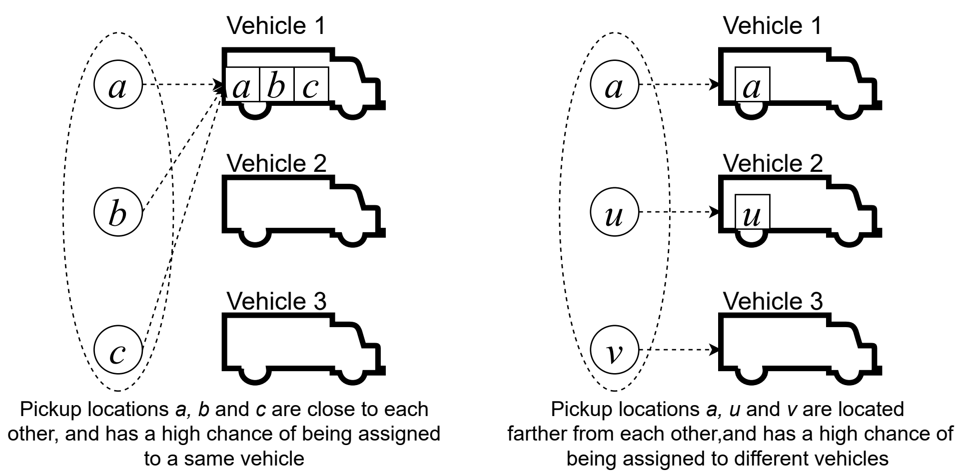


Figure 5.1: Preprocessing illustration

5.2.5 Warm start heuristic

In this section, we present a warm start heuristic that we implemented in our models for identifying a good starting solution. The heuristic has five stages: savings calculation, vehicle assignments, route generation, feasibility check, and cost update. Except for the first step, the remaining steps are iterative. In the vehicle assignment step, we assign customer requests to vehicles such that the cost savings calculated in the first step are maximized. In route generation, we explore different possible routes based on assignments done in the previous step. In feasibility check, we check for vehicle capacity and time window violations in the routes. Finally, in the cost update step, the route with the minimum cost is selected. We explain each stage in detail below.

Savings calculation

The objective of this step is to group customer requests based on savings opportunities. The savings calculation is based on the savings algorithm proposed in the seminal paper by Clark and Wright [15] for VRP. We modify their algorithm for the savings calculation step in our warm start. Let us consider two customer requests $i, j \in P$ for an example. All possible paths for routing these two requests are shown in Figure 5.2.

Route r_1 is a novice choice where both requests are routed using two separate vehicles. There might be a savings opportunity by routing them together using a single vehicle. Routes r_2 to r_7 are all possible options to route the two requests on a single vehicle. Let c_k be the cost of the route r_k. The savings we obtain by routing

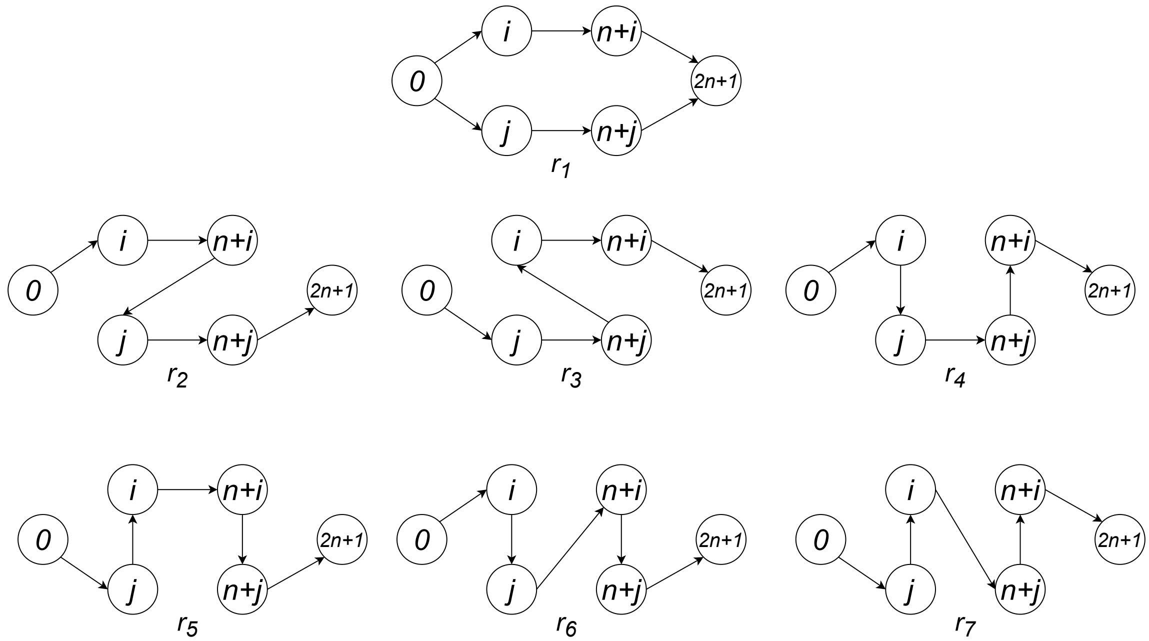


Figure 5.2: All possible routes for $i, j \in P$

requests i and j on a single vehicle is $s_{ij} = c_1 - min(c_2, c_3, c_4, c_5, c_6, c_7)$. Notice that routes r_2 to r_5 are LIFO enforced routes, whereas r_6 and r_7 have LIFO violations. So, their route costs include transportation and handling costs. As a result, we capture the trade-off between enforcing LIFO and allowing handling costs for each customer request pair in the savings. Similarly, we calculate savings s_{ij} for each customer request pair $i, j \in P$. After that, we arrange the customer request pairs in descending order based on the savings value and create a *savings list*. This savings list ranks the customer requests which are most beneficial to be paired together on a route. The intuition behind our heuristic is to iteratively visit the entries in the savings list and generating routes such that the total savings are maximized.

Vehicle assignment

Consider an iteration with a customer request pair $i, j \in P$ from the savings list (note that each entry in the savings list is a customer request pair). One of the following scenarios is possible.

- Requests i and j are already assigned to vehicles, in which case we move to the next entry in the savings list (next pair of customer requests).
- Requests i and j are not on any vehicle route, in which case we assign i and j to vehicle k.
- Request i is already assigned to vehicle k, in which case we assign j to vehicle k. Similarly, if j is already assigned to vehicle k, then we assign i to vehicle k.

Route generation

After assigning customer requests to vehicles, we generate routes based on the latest vehicle assignment from the previous step. This step is similar to node removal and insertion procedure in SPDPH warm start heuristic proposed in Section 4.3.5. Let us consider requests i, j, and $l \in P$ that have been assigned to vehicle k. We create a route $i \rightarrow n+i \rightarrow j \rightarrow n+j \rightarrow l \rightarrow n+l$. After that, we remove pickup and delivery nodes for customer request i (i and $n+i$) from the path. We know i always precedes $n+i$ in the route due to precedence requirement. So, we iteratively generate routes by holding other nodes in their positions, and inserting i and $n+i$ in all possible positions such that precedence is not violated for request i. We show some of the route generation steps with removal and insertion of i and $n+i$ nodes in Figure 5.3.

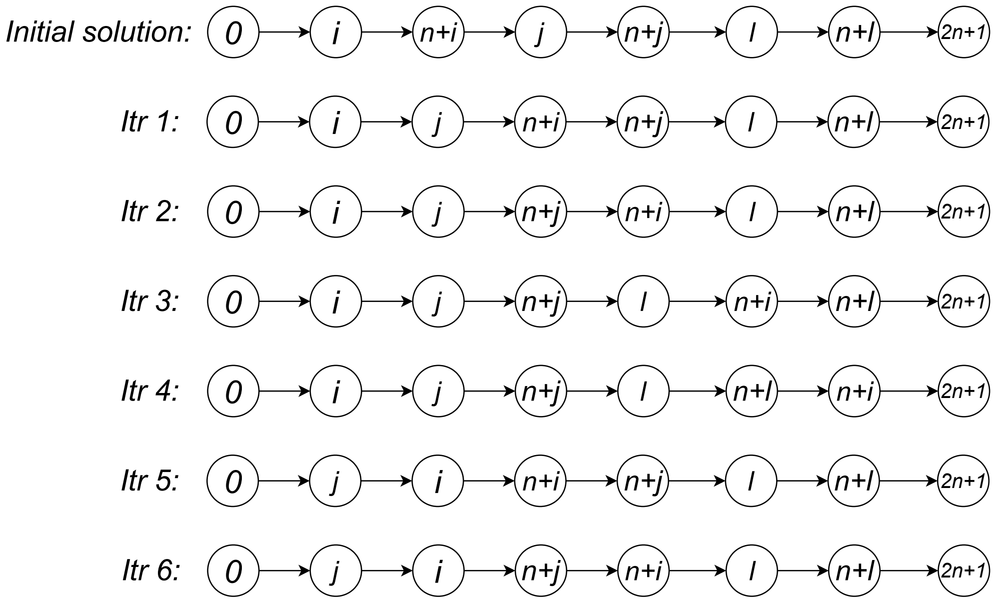


Figure 5.3: Some iterations in the route generation procedure

After that, we repeat the same steps for customer requests j (remove and insert j and $n + j$ iteratively) and l (remove and insert l and $n + l$ iteratively). For each route, we check route feasibility and incumbent cost update before finalizing a route and moving forward to the next iteration.

Feasibility check and cost update

After generating a route, we check it for capacity and time window violations to check feasibility. If a route passes the feasibility tests, then we calculate the objective function for that route. If the route objective value is cheaper than the objective of previous routes, then we update the incumbent cost value in the cost update step. However, if the objective value is not cheaper, then we generate the next route and repeat the feasibility check and cost update.

Let u_k and v_k be the dynamic time and capacity labels on the route of vehicle k. Initially, we set $u_k = v_k = 0$, and we increment the labels as we scan through the route, and checking for violations after visiting each node in the route. For example, while scanning the arc $(i, j) \in A$, the labels are updated as follows: $u_k = u_k + t_{ij}$ and $v_k = v_k + q_j$. We do not consider the route for incumbent cost update, if $u_k > b_j$ or $v_k > Q$. This is because these conditions mean capacity or time window violations in the route. If the route passes the feasibility checks and the route cost is cheaper than other routes for the same vehicle assignment set, then the best route cost is updated. However, it is possible for all the routes for a vehicle assignment to fail the feasibility checks. In that case, the latest vehicle assignment is undone, and we move to the next entry in the savings list.

5.2.6 Branch-and-cut algorithm for MPDPTH-E

In our Formulation MPDPTH-E, there are exponential number of sub-tour elimination Constraints (5.23) and handling cost Constraints (5.24). So, a direct implementation of MPDPTH-E in a commercial solver is computationally expensive. Therefore, we present a BC algorithm with integral separation procedures in this paper for MPDPTH-E implementation. We relax Constraints (5.23) and (5.24) in MPDPTH-E and denote the new formulation as Master Relaxation Problem (MRP). Our algorithm is initiated by finding an integral solution feasible to MRP. Therefore, the resulting solution might have subtours and LIFO violations with zero handling costs. So, we implement separation procedures which are used to identify the node sets that violate Constraints (5.23) and (5.24). Our BC algorithm structure and separation procedures are similar to our SPDPH approach. The following are our separation problems.

Separation Problem 1 (SP1)- To identify subtours

Input: A directed graph $G = (N, A)$ as described in Section 4.1.1, and a vector X^k for a vehicle $k \in K$ containing binary values $x_{ij}^k \ \forall (i,j) \in A$ which are feasible to MRP.

Problem: To identify a node set S, such that $S \subseteq P \cup D$, $2 \leq |S| \leq N$ and $\sum_{i,j \in S} x_{ij}^k > |S| - 1$, or determine that no such set exists.

We can solve SP1 with a simple graph traversal from 0 to $2n+1$. The procedure and complexity are presented in Section 4.2.4. If subtours are identified, then we add inequalities (5.23) as lazy cuts.

Separation Problem 2 (SP2) - To identify LIFO violations and add HC

Input: A directed graph $G = (N, A)$ as described in Section 4.1.1, a solution vector (B, Q, X, Z) feasible to MRP, such that B is the vector containing continuous values $B_i^k \ \forall i \in N, k \in K$, Q is the vector containing continuous values $Q_i^k \ \forall i \in N, k \in K$, X is the arc variable vector containing binary values $x_{ij}^k \ \forall (i,j) \in A, k \in K$ and Z is the LIFO violation vector containing binary values $z_{ij}^k \ \forall i,j \in P, k \in K$.

Problem: For each node pair $i, j \in P$ and $k \in K$, identify a node set $S \in \Upsilon_j$, such that $i \notin S$, $n+i \notin S$, $j \in S$, $n+j \notin S$, $\left[\sum_{u \in S} x_{iu}^k + \sum_{u,v \in S} x_{uv}^k + \sum_{u \in S} x_{u,n+i}^k\right] - |S| > z_{ij}^k$ or determine that no such set exists.

We solve SP2 with a simple graph traversal from 0 to $2n+1$ for each vehicle

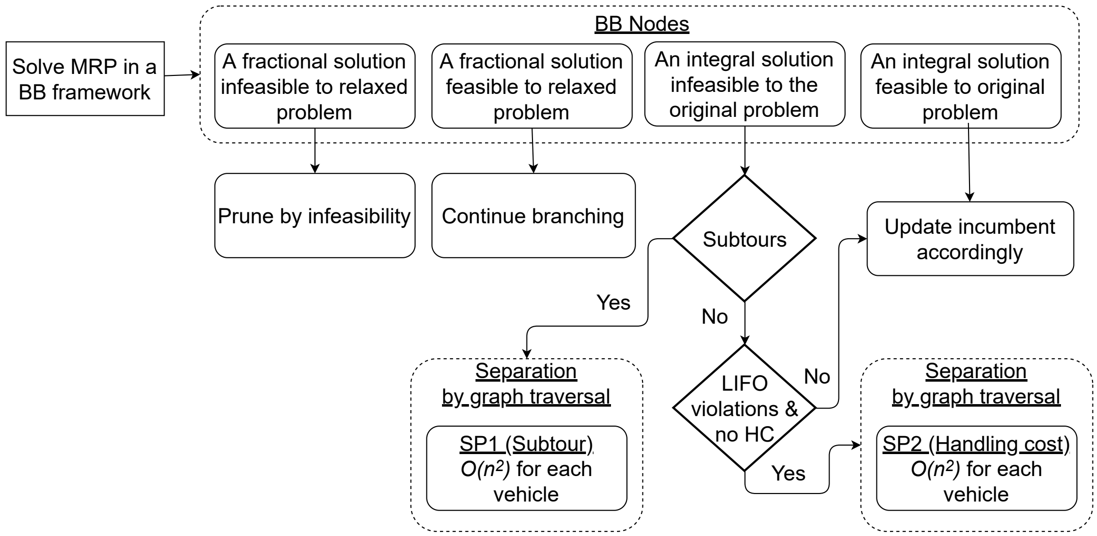


Figure 5.4: BC algorithm structure MPDPTH-E

$k \in K$, and indexing the node positions on the path based on their order of visits. The procedure and complexity are presented in Section 4.2.4. If LIFO violations are identified, then we add inequality (5.24) as a lazy cut.

BC algorithm structure

The structure of our BC algorithm is shown in Figure 5.4. We start by solving MRP in a Branch-and-Bound framework. If a BB node corresponds to a fractional solution, then we handle them in a traditional BB approach. If the fractional solution is infeasible to the MRP, then we prune the BB node by infeasibility. On the other hand, if the fractional solution is feasible to MRP, then we continue branching. If a BB node contains an integral solution feasible to the original formulation, then we update the incumbent accordingly. However, if we have an integral solution infeasible to the original problem, then we solve the separation problems to identify subtours and LIFO violations for each vehicle route. After that, we add inequalities (5.23) and (5.24) as lazy cuts.

5.3 Multi Vehicle Pickup-and-Delivery Problem with Time Windows and Loading Constraints

In this section, we: (1) Present two formulations with exponentially many constraints for MPDPTL; (2) Explore BC algorithms for the two formulations; (4) Explore runtime improvements including families of inequalities and a warm start heuristic. Similar to MPDPTH, the objective of MPDPTL is to find minimum cost Hamiltonian path(s) from origin depot $\mathbf{0}$ to destination depot $\mathbf{2n+1}$. However, LIFO violations are not permitted. So, a delivery location can be visited only if the corresponding shipment is at the access end of the vehicle.

5.3.1 Formulation MPDPTL1

Decision variables

$\boldsymbol{B}$, $\boldsymbol{Q}$, $\boldsymbol{u}$ and $\boldsymbol{x}$ variables as defined in Table 5.1

Formulation 5.3.1.

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ij}^k \tag{5.29}$$

subject to:

Constraints (5.2)-(5.8) *and* (5.17) *from Formulation 5.2.1*

Constraints (5.20)-(5.23) *from Formulation 5.2.2*

$$\sum_{k\in K} Q_{n+i}^k = \sum_{k\in K} Q_i^k - q_i \quad \forall i \in P \tag{5.30}$$

Objective function and constraints

The objective function seeks to minimize the total transportation cost. Except for LIFO Constraints (5.30), the remainder of the formulation is similar to PDP with time windows formulation proposed by Ropke et al. [36]. Those constraints are explained in Formulations 5.2.1 and 5.2.2. LIFO constraints (5.30) were designed based on a nice property of SPDPL solution presented by Cordeau et al. [18] for single vehicle problems. This property is explained in the proposition below.

Proposition 2 (Cordeau et al. [18])**.** *The net amount delivered between each pair* $i \in P$, $n+i \in D$ *is equal to 0 for a SPDPL solution*

This proposition is based on the following observation. Consider two customer requests: (pickup at i and delivery at $n+i$) and (pickup at j and delivery at $n+j$). Figure 5.5 shows a route respecting LIFO loading order. For both customer requests, the vehicle weight entering the pickup node is equal to the vehicle weight leaving the corresponding delivery node. For instance, the total weight entering pickup node i is 135 lbs, which is equal to the total weight leaving the corresponding delivery node $n+i$.

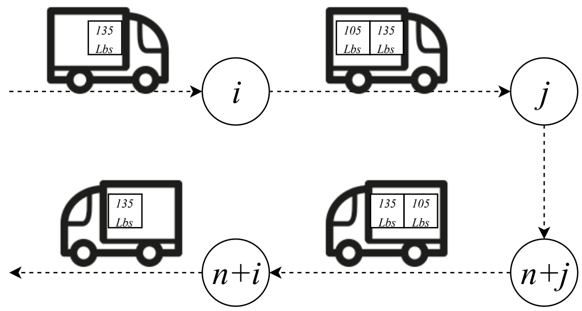


Figure 5.5: An illustration of LIFO loading property

BC algorithm for MPDPTL1

There are exponentially many subtour elimination constraints in MPDPTL1. So, we implement it using a BC algorithm. The algorithm structure is similar to the MPDPTH BC algorithm presented in Section 5.2.6. However, we do not solve the handling cost separation problem (SP2) because it is handled by Constraints (5.30).

5.3.2 Formulation MPDPTL2

Decision variables

$\boldsymbol{B}$, $\boldsymbol{Q}$, $\boldsymbol{u}$ and $\boldsymbol{x}$ variables as defined in Table 5.1

Formulation 5.3.2.

$$\min \sum_{k\in K} \sum_{(i,j)\in A} c_{ij} x_{ij}^k \tag{5.31}$$

subject to:

Constraints (5.2)-(5.8) *and* (5.17) *from Formulation 5.2.1*

Constraints (5.20)-(5.23) *from Formulation 5.2.2*

$$x^k(i,S) + x^k(A(S)) + x^k(S,n+i) \leq |S| \quad \forall S \in \Upsilon_j, \forall k \in K, \forall i, n+i \notin S, \forall i,j \in P \tag{5.32}$$

Objective function and constraints

The objective function seeks to minimize the total transportation cost. Except for LIFO Constraints (5.32), the remaining constraints are explained in Formulations 5.2.1 and 5.2.2. LIFO Constraints (5.32) were introduced by Cherkesly et al. [14] for multi-vehicle PDP with time windows and LIFO , which in turn is the multi-vehicle extension of LIFO constraints introduced by Cordeau et al. [18] for SPDPL.

BC algorithm for MPDPTL2

There are exponentially many subtour elimination and LIFO constraints in MPDPTL2. So, we implement it using a BC algorithm whose structure is similar to the MPDPTH BC algorithm presented in Section 5.2.6. However, we add Constraints (5.32) as lazy cuts after solving SP2 because we do not permit LIFO violation in MPDPTL.

Other runtime improvements

We add symmetry breaking inequalities (Section 5.2.3) to enhance the computational scalability of our MPDPTL implementations. We also use a warm start heuristic to provide a starting solution for our MPDPTL algorithms. The structure of this warm start heuristic is the same as the MPDPTH algorithm discussed in Section 5.2.5 with one small modification. We reject LIFO violating solutions in the route generation step of the heuristic.

CHAPTER VI

COMPUTATIONAL RESULTS

In this chapter, we present the computational results for the problems that we address in this dissertation.

6.1 Single Vehicle Pickup-and-Delivery Problem with Loading Constraints

In this section, we present our experimental set-up, test-bed details, and computational results for SPDPL methodologies.

SPDPL test-bed details: SPDPL solution approaches were tested on instances from Carrabs et al. [11]. They solved SPDPL using a variable neighborhood heuristics. This test-bed has 32 instances ranging from 9 to 21 shipment orders. The instances were posted for public access by Chair in Logistics and Transportation research program, HEC Montréal business school [13].

SPDPL implementation details: The SPDPL solution approaches were implemented using C++ and Gurobi™ 7.5.2 on dual Intel® Xeon E5-2620 Sandy Bridge hex core 2.0 GHz CPU, with 32 GB RAM. A time limit of 2 hours was imposed in all instances.

Remarks on computational results

Table 6.1 compares the performance of our branch-and-cut algorithm with integral separation procedures (SPDPL-I) against branch-and-cut algorithm with fractional separation procedures (SPDPL-F) by Cordeau et al. [18] on 16 small instances. Column header n denotes the number of customer requests. UB denotes the best upper bound on the objective function value identified by us. We have also reported the integrality %gap $\left(\frac{|UB|-|LB|}{|UB|}\right)$ between the upper and lower bound of objective value. For instances that were not solved to optimality within the time limit, the %gap at the end of 2 hours has been reported. We have indicated the algorithm with shorter runtime and smaller %gaps in bold font. The number of Branch-and-Bound nodes (#BB nodes) has been reported for both approaches. The table also shows the number of separation problems solved for sub-tour elimination (SEC), precedence (PRE), and Last-In-First-Out (LIFO) constraints.

Some remarks from the Table 6.1 are as follows:

- SPDPL-I solved 14 out of 16 instances to optimality within the time limit, whereas SPDPL-F solved 12 out of 16 instances to optimality within the time limit.

- SPDPL-I and SPDPL-F timed out in two and four instances respectively. For those instances, SPDPL-I terminated with a smaller %gap than SPDPL-F (brd14015 and nrw1379 with $n = 13$).

- Except for att532, d15112, and fnl4461 with $n = 9$, SPDPL-I has a higher number of BB nodes on all instances.

Table 6.1: SPDPL-F and SPDPL-I computational results

Instance	n	Upper bound	Time (Secs)		Gap		#BB Nodes		#SEC		#PRE		#LIFO	
			SPDPL-F	SPDPL-I	SPDPL-F	SPDPL-I	SPDPL-F	SPDPL-I	SPDPL-F	SPDPL-I	SPDPL-F	SPDPL-I	SPDPL-F	SPDPL-I
att532	9	4,250	866.8	**51.2**	0%	0%	11,694	9,075	4,684	119	9,463	151	13,718	240
brd14051		4,555	484.7	**120.3**	0%	0%	7,819	25,311	3,055	118	7,439	135	6,919	516
d15112		76,203	115.5	**12.6**	0%	0%	4,435	4,314	1,556	53	2,741	39	4,682	364
d18512		4,446	77.3	**76.9**	0%	0%	3,951	7,426	2,733	88	4,148	105	1,675	382
fnl4461		1,866	2.4	**1.6**	0%	0%	20	1	74	8	30	1	92	62
nrw1379		2,691	190.7	**157.1**	0%	0%	10,172	27,927	3,709	89	11,668	232	2,019	142
pr1002		12,947	2.1	**1.2**	0%	0%	1	1	113	7	25	1	29	26
ts225		21,000	10.5	**5.6**	0%	0%	160	281	248	16	116	6	151	58
att532	13	6,495	305.1	**208.8**	0%	0%	1,814	322,354	7,941	888	4,234	1,738	480	2,118
brd14051		5,097	>7,200	**>7,200**	46%	12%	304,208	873,545	132,912	1,356	574,713	2,074	279,172	4,902
d15112		101,858	>7,200	**740.8**	12%	0%	4,733	1,092,430	8,632	1,136	6,335	1,458	3,340	11,730
d18512		4,704	1,944.3	**338.9**	0%	0%	108,392	839,507	44,124	624	70,761	1,235	234,927	5,212
fnl4461		2,483	>7,200	**6,405.4**	35%	0%	276,967	2,129,970	60,103	596	100,013	569	653,986	12,396
nrw1379		3,641	>7,200	**>7,200**	44%	20%	274,073	776,191	228,645	1,487	839,381	5,207	141,575	2,838
pr1002		15,566	16.4	**8.4**	0%	0%	126	386	397	25	123	15	82	30
ts225		32,395	46.4	**45.4**	0%	0%	539	5,722	1,125	94	568	35	449	396

- For SPDPL-I, the number of LIFO enforcing cuts are higher than sub-tour elimination and precedence enforcing cuts for around 88% of instances.

From Table 6.1 results in small instances, our branch-and-cut algorithm with integral separation procedures (SPDPL-I) is the clear winner. With this observation, we implemented SPDPL-I on larger test instances to measure its scalability. SPDPL-I performance on 16 large instances with 17 and 21 customer requests is shown in Table 6.2.

Table 6.2: SPDPL-I computational results on large instances

Instance	n	Upper bound	Time (secs)	Gap	#BB Nodes	#Separation problems SEC	PRE	LIFO
att532	17	6,365	>7,200	4%	550,584	1,470	4,329	3,644
brd14051		11,650	>7,200	63%	96,273	3,216	8,074	-
d15112		138,157	>7,200	29%	365,121	4,042	6,615	7,680
d18512		6,617	>7,200	31%	281,929	3,677	7,836	4,802
fnl4461		3,926	>7,200	37%	383,003	3,808	6,379	12,500
nrw1379		5,760	>7,200	48%	269,617	3,753	8,848	2,418
pr1002		17,564	24	0%	654	60	9	58
ts225		36,703	142.7	0%	6,096	58	23	182
att532	21	13,067	>7,200	27%	193,083	4,091	10,253	758
brd14051		9,209	>7,200	51%	226,327	3,680	8,691	804
d15112		154,535	>7,200	35%	253,111	4,149	7,739	1,422
d18512		7,693	>7,200	39%	203,327	3,652	7,677	1,284
fnl4461		4,385	>7,200	37%	323,094	3,690	5,721	9,890
nrw1379		8,364	>7,200	61%	175,334	3,865	12,364	322
pr1002		20,173	105.2	0%	3,713	164	34	144
ts225		43,082	255.2	0	97,156	867	1,099	8,298

Some remarks from the Table 6.2 are as follows:

- Across all instances (from Tables 6.1 and 6.2), around 57% of instances were solved to optimality within the 2-hour time limit by SPDPL-I.

- All timed-out instances terminated with a reasonable upper bound on objective function due to a decent warm start. For instances with **17** requests, brd14051 terminated without solving any LIFO violation separation problem. So, the upper bound for that instance is completely due to the warm start heuristic.

- The number of precedence cuts are higher than other cuts for large timed-out instances (9 out of 16). Since our separation procedures are nested, the solver expended more time on the precedence stage before proceeding to LIFO violation separation problems.

In summary, SPDPL-I (branch-and-cut algorithm introduced in this dissertation) outperforms SPDPL-F (branch-and-cut algorithm by Cordeau et al. [18]) on our test-bed. However, our approach expends more time on separation problems for precedence enforcement in large instances. Our runtime can probably be improved by focusing on some heuristic separation procedures for precedence enforcement.

6.2 Single Vehicle Pickup-and-Delivery Problem with Handling Costs

In this section, we present our experimental set-up, test-bed details, and computational results of SPDPH methodologies.

SPDPH test-bed details: We implement our SPDPH algorithms on the same testbed as SPDPL. For handling costs, we use the values proposed by Veenstra et al. [42]. All customer requests will be forced to respect LIFO with very high handling cost values. With that observation, Veenstra et al. [42] presented two different handling costs for each instance from Carrabs et al. [11] such that LIFO violations will be permitted. We explore 60 instances ranging from 9 to 25 shipments, with handling costs ranging from \$1 to \$1000.

SPDPH implementation details: Single vehicle algorithms were implemented using C++ and Gurobi™ 8.1.1 on Intel® Xeon W3670 3.20 GHz CPU, with 8 GB RAM and Windows® 7 Professional operating system. A time limit of 2 hours was imposed on all instances.

Remarks on computational results

Table 6.3 shows the performance of formulation introduced by Veenstra et al. [42] (SPDPH1), our compact formulation (SPDPH2) and our branch-and-cut algorithms with fractional (SPDPH3-F) and integral separation procedures (SPDPH3-I) in 32 instances.

Some remarks from Table 6.3 are as follows:

- From the runtime results, SPDPH3-I outperformed SPDPH1, SPDPH2 and SPDPH3-F on 22 out of 32 instances.
- Comparing the root node relaxation of compact formulations, SPDPH1 is the clear winner across all instances with tighter root node relaxations.
- A noticeable issue with our branch-and-cut algorithm, when compared against SPDPH1, is the higher number of BB nodes across all instances.

Table 6.3: SPDPH results on small instances (all approaches)

Instance	n	Handling cost	Upper bound	Root node SPDPH1	Root node SPDPH2	Runtime (Secs) SPDPH1	Runtime (Secs) SPDPH2	Runtime (Secs) SPDPH3-F	Runtime (Secs) SPDPH3-I	#BB Nodes SPDPH1	#BB Nodes SPDPH2	#BB Nodes SPDPH3-F	#BB Nodes SPDPH3-I
att532	9	10	3,911	3,671	3,346	21.6	14.1	237	**7.6**	233	1,006	98	1,240
		50	4,122	3,673	3,404	63.6	60.0	294	**35.3**	1,776	40,089	100	22,956
brd14051	9	10	4,389	4,211	1,224	64.4	1,053.7	1,255	**9.8**	3,772	8,816	804	4,584
		50	4,528	4,214	1,266	97.3	661.7	>7,200 (6%)	**35.5**	4,584	19,495	3,718	19,839
d15112	9	500	74,452	68,193	59,454	43.3	91.2	2,575	**37.3**	2,397	87,820	2,142	13,832
		1,000	76,040	68,194	59,462	**43.6**	72.4	>7,200 (11%)	88.4	2,336	16,556	3,724	63,560
d18512	9	1	4,245	4,208	1,145	**4.5**	1,159.5	151	8.1	1	79,532	71	9,180
		10	4,288	4,221	1,159	**14.9**	835.9	2,688	21.6	1	284,002	1,513	5,152
fnl4461	9	1	1,804	1,800	776	**2.3**	5,007.9	70	2.6	1	145,807	25	1
		10	1,847	1,841	808	**2.8**	4827.3	2,306	6.1	1	2,517	1,625	14
nrw1379	9	10	2,553	2,491	875	20.8	>7,200 (31%)	252	**6.9**	295	616,399	199	7,817
		50	2,622	2,494	912	38.5	>7,200 (22%)	1,707	**12.3**	715	490,266	1,125	7,768
pr1002	9	50	12,749	12,622	9,516	**2.9**	85.1	89	3.7	1	11,874	1	38
		100	12,947	12,787	9,661	**3.3**	55.3	273	4.5	1	15,286	120	1
ts225	9	500	19,000	19,000	14,361	3.5	2.2	91	**2.0**	1	158	1	52
		1,000	20,000	19,000	14,392	5.1	3.3	698	**3.3**	1	172	284	353
att532	13	10	5,037	4,967	4,527	27.0	170.7	594.2	**19.1**	1	24,625	87	1,103
		50	5,354	4,979	4,615	619.4	784.3	1,519.5	**182.2**	894	79,640	409	93,789
brd14051	13	10	4,526	4,261	1,238	>7,200 (3%)	>7,200 (6%)	>7,200 (10%)	**>7,200 (2%)**	25,208	2,891,714	76,980	2,719,500
		50	4,762	4,261	1,280	>7,200 (7%)	>7,200 (17%)	>7,200 (20%)	**>7,200 (5%)**	19,840	1,437,947	170,785	911,295
d15112	13	500	85,761	76,781	66,941	2757.5	5,807.8	>7,200 (5%)	**911.7**	15,143	554,801	21,457	138,558
		1,000	88,806	76,849	67,008	>7,200 (5%)	>7,200 (11%)	>7,200 (24%)	**>7,200 (2%)**	22,803	161,610	30,822	526,058
d18512	13	1	4,305	4,261	1,159	**27.7**	>7,200 (18%)	565.0	30.4	1	391,730	21	2,680
		10	4,348	4,279	1,175	386.3	7,078.4	>7,200 (6%)	**134.8**	3,928	1,115,559	28,050	95,516
fnl4461	13	1	2,055	2,050	884	**9.2**	>7,200 (23%)	498.7	18.4	1	1,559,070	1,575	63
		10	2,156	2,102	922	**62.4**	>7,200 (21%)	>7,200 (5%)	91.8	310	780,372	90,411	2,502
nrw1379	13	10	3,187	2,773	974	>7,200 (10%)	>7,200 (47%)	>7,200 (13%)	**>7,200 (10%)**	17,847	1,729,112	13,339	523,975
		50	3,586	2,775	1,015	>7,200 (19%)	>7,200 (41%)	>7,200 (21%)	**>7,200 (16%)**	9,908	2,793,796	43,686	301,649
pr1002	13	50	15,368	15,126	11,403	63.7	1,900.9	399.5	**16.4**	1	31,882	8	311
		100	15,566	15,283	11,547	68.2	1,129.3	1,442.1	**24.0**	1	20,389	52,680	439
ts225	13	500	30,395	30,032	22,700	37.9	23.5	776.5	**17.2**	1	1,421	4	212
		1,000	31,395	30,032	22,748	132.9	85.4	5,960.4	**28.0**	1	1,372	2,478	3,080

- SPDPH1 and SPDPH3-I solved 27 out of 32 instances to optimality within the 2-hour time limit. However, SPDPH2 and SPDPH3-F timed out on 10 instances each. The %gap $\left(\frac{|UB|-|LB|}{|UB|}\right)$ for those timed out instances are provided in the runtime results within parenthesis.

- SPDPH3-F has exhibited the longest runtime for 20 out of 32 instances. Whereas, SPDPH2 has the highest number of BB nodes for 28 out of 32 instances.

- SPDPH1 has the lowest number of BB nodes for 24 out of 32 instances. Furthermore, SPDPH1 was able to solve 15 instances with just 1 BB node.

From our computational results, SPDPH1 and SPDPH3-I were identified as the top-performing approaches. Table 6.4 shows the impact of the problem size on these approaches, with an overview of the termination status in four instance groups based on the number of customer requests (n). The status results for both approaches are identical in instances with 9 or 13 requests. Specific runtime details for these small instances are shown in Table 6.5. For instances with 17 requests, SPDPH1 solved 25% of instances to optimality within the 2-hour time limit, whereas SPDPH3-I solved 37.5% of instances to optimality. Similarly, SPDPH3-I solved more instances to optimality than SPDPH1 for instances with 17 or 21 customer requests. Furthermore, SPDPH3-I outperformed SPDPH1 by the runtime in all optimally solved instances with 17 or 21 customer requests. More importantly, SPDPH1 terminated with an out-of-memory status without a lower bound in 25% and 75% instances with $n = 17$ and 21, respectively, whereas SPDPH3-I never terminated with that status in any instance. SPDPH3-I solved those instances either within or beyond the 2-hour time limit. For those timed-out instances, SPDPH3-I still managed to obtain reasonable integrality gaps as shown in Table 6.6. Therefore, SPDPH3-I is the clear winner in large instances. Since SPDPH1 terminated with an out-of-memory status in many large instances, we only present the performance of SPDPH3-I in large instances in Table 6.6.

Table 6.4: Percentage breakdown of termination status for top two approaches

Status	$n = 9$ (16 instances)		$n = 13$ (16 instances)		$n = 17$ (16 instances)		$n = 21$ (16 instances)	
	SPDPH1	SPDPH3-I	SPDPH1	SPDPH3-I	SPDPH1	SPDPH3-I	SPDPH1	SPDPH3-I
Optimal	100%	100%	68.8%	68.8%	25%	37.5%	12.5%	18.8%
Timed-out	0%	0%	31.2%	31.2%	50%	62.5%	12.5%	81.2%
Out-of-memory	0%	0%	0%	0%	25%	0%	75%	0%

Table 6.5 shows the performance of SPDPH1 and SPDPH3-I in 32 small instances comprising of 9 or 13 shipment requests. This table shows the number of LIFO violations and the best upper bound of the solution, root node relaxation for SPDPH1, runtime, integrality gap between the best Upper Bound (UB) and the best Lower Bound (LB), the number of BB nodes, and the number of separation problems in SPDPH3-I corresponding to subtour elimination (SEC), precedence (PRE) and handling cost (HC) enforcements. From the runtime results, SPDPH3-I outperformed SPDPH1 in 69% of instances (22 instances). Among them, both approaches timed-out in 5 instances, namely brd14051, d15112, and nrw1379. However, SPDPH3-I terminated with smaller integrality gap for those instances. For the optimally resolved instances where SPDPH3-I was faster than SPDPH1, the average runtime improvement was around 57%. For 31% of instances in which SPDPH1 runtime was smaller than SPDPH3-I, the number of handling cost separation problems (SP3) is higher than other problems on average. This indicates that the solver spent more

Table 6.5: SPDPH results on small instances (top two approaches)

Instance	n	Handling cost	#LIFO violations	Upper bound	Root node SPDPH1	Time (Secs) SPDPH1	Time (Secs) SPDPH3-I	Gap SPDPH1	Gap SPDPH3-I	#BB Nodes SPDPH1	#BB Nodes SPDPH3-I	#Separation problems SEC	PRE	HC
att532	9	10	8	3,911	3,671	21.6	**7.6**	0%	0%	233	1,240	36	56	77
		50	3	4,122	3,673	63.6	**35.3**	0%	0%	1,776	22,956	91	101	242
brd14051	9	10	4	4,389	4,211	64.4	**9.8**	0%	0%	3,772	4,584	52	49	53
		50	2	4,528	4,214	97.3	**35.5**	0%	0%	4,584	19,839	91	105	226
d15112	9	500	9	74,452	68,193	43.3	**37.3**	0%	0%	2,397	13,832	49	50	367
		1,000	2	76,040	68,194	**43.6**	88.4	0%	0%	2,336	63,560	68	67	497
d18512	9	1	5	4,245	4,208	**4.5**	8.1	0%	0%	1	9,180	46	47	36
		10	4	4,288	4,221	**14.9**	21.6	0%	0%	1	5,152	32	31	194
fnl4461	9	1	7	1,804	1,800	**2.3**	2.6	0%	0%	1	1	6	4	5
		10	4	1,847	1,841	**2.8**	6.1	0%	0%	1	14	7	4	42
nrw1379	9	10	2	2,553	2,491	20.8	**6.9**	0%	0%	295	7,817	39	52	22
		50	1	2,622	2,494	38.5	**12.3**	0%	0%	715	7,768	59	147	73
pr1002	9	50	6	12,749	12,622	**2.9**	3.7	0%	0%	1	38	8	4	17
		100	0	12,947	12,787	**3.3**	4.5	0%	0%	1	1	10	4	13
ts225	9	500	2	19,000	19,000	3.5	**3.0**	0%	0%	1	52	17	4	17
		1,000	1	20,000	19,000	5.1	**3.3**	0%	0%	1	353	17	4	17
att532	13	10	10	5,037	4,967	27.0	**19.1**	0%	0%	1	1,103	26	42	36
		50	5	5,354	4,979	619.4	**182.2**	0%	0%	894	93,789	79	133	163
brd14051	13	10	8	4,526	4,261	>7,200	**>7,200**	3%	2%	25,208	2,719,500	132	456	318
		50	5	4,762	4,261	>7,200	**>7,200**	7%	5%	19,840	911,295	378	890	1,523
d15112	13	500	16	85,761	76,781	2757.5	**911.7**	0%	0%	15,143	138,558	168	236	1,455
		1,000	5	88,806	76,849	>7,200	**>7,200**	5%	2%	22,803	526,058	331	452	3,136
d18512	13	1	14	4,305	4,261	**27.7**	30.4	0%	0%	1	2,680	49	41	71
		10	4	4,348	4,279	386.3	**134.8**	0%	0%	3,928	95,516	55	59	216
fnl4461	13	1	15	2,055	2,050	**9.2**	18.4	0%	0%	1	63	12	9	60
		10	11	2,156	2,102	**62.4**	91.8	0%	0%	310	2,502	30	30	524
nrw1379	13	10	13	3,187	2,773	>7,200	**>7,200**	10%	10%	17,847	523,975	540	3,250	891
		50	1	3,586	2,775	>7,200	**>7,200**	19%	16%	9,908	301,649	482	2,097	914
pr1002	13	50	6	15,368	15,126	63.7	**16.4**	0%	0%	1	311	32	16	14
		100	0	15,566	15,283	68.2	**24.0**	0%	0%	1	439	23	19	22
ts225	13	500	2	30,395	30,032	37.9	**17.2**	0%	0%	1	212	20	2	49
		1,000	2	31,395	30,032	132.9	**28.0**	0%	0%	1	3,080	26	3	87

time in identifying the LIFO violations for those instances.

A noticeable issue of SPDPH3-I, when compared with SPDPH1, is the higher number of BB nodes across all instances. Especially the numbers of BB nodes for timed-out instances brd14051, d15112, and nrw1379 are very high, which indicates a hard effort by the solver. In contrast, SPDPH1 solved 15 instances including d18512, fnl4461, pr1002, and ts225 with just one BB node. This is because SPDPH1 exhibits tight root node relaxation values.

Table 6.6 shows the performance of SPDPH3-I in 32 large instances comprising of 17 or 21 shipment requests. In this table, only 28.1% of instances were solved to optimality within the 2-hour time limit. However, the integrality gaps of SPDPH3-I in the timed-out instances at the end of the run were within reasonable levels. This was primarily due to the decent warm start because we noticed that many of the large timed-out instances terminated without an upper bound if a warm start was not included. The performance of SPDPH3-I in fnl4461, pr1002 and ts225 is better than in other instances with the same number of customer requests. Especially for pr1002 and ts225, the short runtime can be attributed to the low numbers of separation problems and BB nodes. Comparing the number of separation problems in large instances, PRE is higher than SEC and HC on average. Therefore, more focus should be directed towards the precedence constraints for further reduction in the runtime.

The worst performing instances for our implementation are brd14051, nrw1379, and d18512. After a close inspection of their solution structures, we identified the pathological characteristics of our SPDPH3-I algorithm. To understand these char-

Table 6.6: SPDPH3 computational results

Instance	n	Handling cost	#LIFO Violations	Upper bound	Time (secs)	Gap	#BB Nodes	#Separation problems SEC	PRE	HC
att532	17	10	22	5,514	111.8	0%	8,102	108	27	145
		50	10	6,047	>7,200	6%	1,139,290	338	574	857
brd14051	17	10	22	5,418	>7,200	21%	46,319	1,299	5,547	19
		50	6	5,673	>7,200	24%	31,193	1,306	5,882	5
d15112	17	500	11	126,534	>7,200	25%	150,125	744	1,653	5,919
		1,000	11	132,034	>7,200	28%	120,967	823	1,818	3,997
d18512	17	1	21	4,674	>7,200	5%	1,118,090	184	3,412	111
		10	12	4,812	>7,200	6%	247,356	579	3,727	802
fnl4461	17	1	34	2,311	86.5	0%	62	22	15	158
		10	14	2,593	>7,200	5%	801,272	131	278	1,751
nrw1379	17	10	15	3,572	>7,200	17%	171,787	979	5,363	627
		50	26	4,015	>7,200	7%	116,871	1,699	6,588	1,981
pr1002	17	50	7	17,138	52.9	0%	550	42	72	19
		100	1	17,386	66.6	0%	1,761	42	60	26
ts225	17	500	2	35,703	95.3	0%	5,483	83	13	73
		1,000	0	36,703	145.3	0%	20,415	89	35	73
att532	21	10	21	9,836	>7,200	5%	398,513	446	2,538	265
		50	16	10,485	>7,200	11%	92,307	2,025	8,225	1,675
brd14051	21	10	17	6,420	>7,200	30%	82,293	1,920	7,559	135
		50	4	6,811	>7,200	34%	92,982	1,994	6,430	1,178
d15112	21	500	20	137,894	>7,200	30%	85,145	1,566	3,931	4,045
		1,000	16	146,491	>7,200	34%	104,925	1,175	2,393	746
d18512	21	1	40	6,371	>7,200	29%	195,211	766	4,139	383
		10	19	6,602	>7,200	30%	143,500	814	3,081	286
fnl4461	21	1	50	2,595	603.9	0%	9,480	53	107	1,002
		10	30	2,899	>7,200	9%	229,498	479	1,245	3,119
nrw1379	21	10	19	4,162	>7,200	24%	65,809	1,817	8,965	95
		50	5	4,607	>7,200	31%	63,859	2,157	10,195	-
pr1002	21	50	8	19,665	163.7	0%	2,380	84	203	53
		100	2	19,963	164.3	0%	6,426	66	37	34
ts225	21	500	4	45,541	>7,200	11%	303,590	216	518	1,097
		1,000	1	43,082	>7,200	3%	290,470	376	557	1,669

acteristics it is necessary to understand LIFO enforced route structures. So, we discuss the two types of LIFO route structures before presenting the pathological characteristics. Figure 6.1(a) shows a sequential structure in which the pickup nodes of some customer requests are close to their delivery nodes in the solution sequence.

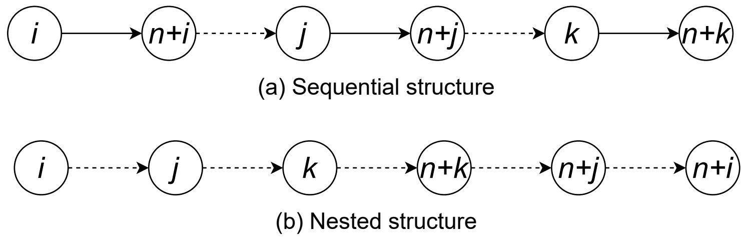


Figure 6.1: Two types of LIFO route structures

On the other hand, Figure 6.1(b) shows a nested structure in which the pickup nodes of some customer requests are located far away from their delivery nodes in the solution sequence. For example, node $i \in P$ is located far away from $n + i \in D$ in the solution sequence. Instances with a solution containing nested structure for customer requests affect our SPDPH3-I implementation negatively. For example, the feasible solutions of brd14051 had nested structure for many customer requests. In the solutions for these instances, many nodes were in the vehicle route between the pickup and the delivery nodes of other customer requests. This prompted the addition of a large number of precedence and handling cost lazy cuts. In contrast, pr1002 and ts225 results do not have a nested structure for most of the customer requests. The pickup and delivery nodes for those instances were close to each other in the solution sequence, which prompted less effort for precedence and handling cost enforcement.

6.3 Multi Vehicle Pickup-and-Delivery Problem with Time Windows and Handling Cost

In this section, we present our experimental set-up, test-bed details and computational results of our MPDPTH methodologies.

MPDPTH test-bed details: We implemented our algorithms in real-world instances procured from a logistics company for 10 days. We tested our algorithm performance on 50 instances, each with 9, 13, 17, 21, or 25 customer requests of the same commodity class. All shipments were delivered using LTL trucks. The maximum on-road time for the driver was set as 35 hours. This time was set considering that there were some cross-country shipments with 55 hours delivery deadline and the maximum consecutive hours a truck driver can work is up to 11 hours after which an off-duty time of 10 hours is recommended [28]. The vehicle capacity was set to 20,000 lbs per truck. All commodities in the customer requests fall under the same LTL weight class category and would reach maximum weight capacity before volume capacity while loading into the trucks. From the shipment data, we calculated distances between all sites using a Google® Maps xml plugin in Microsoft® Excel VBA. For each arc, we assigned a transportation cost of $1.38 per mile (based on estimates from [40]). For all network sites, the lower limit of time windows was set as 0. This is because the trucks can be dispatched from the depot by 7 am in our instances, which is the same time by which the network locations start their daily operations. For all instances, the handling cost was assumed to be $30 for unloading and reloading one shipment. This assumption is based on the average handling cost calculation across the customer requests for 10 days from which our instances were extracted.

MPDPTH implementation details: The MPDPTH solution approaches were implemented using C++ and Gurobi™ 7.5.2 on dual Intel® Xeon E5-2620 Sandy Bridge hex core 2.0 GHz CPU, with 32 GB RAM. A time limit of 4 hours was imposed in all instances.

Remarks on computational results

Table 6.7 shows the performance of MPDPTH-C (our compact formulation) and MPDPTH-E (out cut-based formulation) in 30 instances comprising of 9, 13, or 17 customer requests. We denote MPDPTH-C and MPDPTH-E as MV-C and MV-E respectively in the table for a compact presentation. We have also presented the results from our Warm Start (WS) heuristic. Even though the primary purpose of this heuristic is to provide a warm start for other methodologies, we presented the details here because it identified decent results for MPDPTH with very short runtime. Table 6.7 shows the following WS heuristic results: best objective cost, % difference against the best UB (from exact approaches), and runtime. The table also shows the following results for MPDPTH-C and MPDPTH-E: the best objective cost or Upper Bound (UB), the best Lower Bound (LB), % gap between UB and LB, runtime, the number of BB nodes, and the number of lazy cuts added in MPDPTH-E after solving the separation problems.

The WS heuristic identified good results with very small runtime. The heuristic solutions are within 20% of the best UB solution for 90% of instances (27 out of

Table 6.7: MPDPTH results in instances with 9, 13 and 17 requests (three approaches)

Instance	n	WS Heuristic			Cost		Lower Bound (LB)		Gap		Secs		#BB Nodes		#cuts
		Cost	Diff	Secs	MV-C	MV-E	MV-C	MV-E	MV-C	MV-E	MV-C	MV-E	MV-C	MV-E	
S274	9	5,604	14%	0.02	4,913	4,913	4,913	4,913	0%	0%	1.6	**0.2**	1	1	0
S284		6,071	12%	0.02	5,417	5,417	5,417	5,417	0%	0%	1.5	**0.2**	1	1	0
S155		5,078	12%	0.02	4,550	4,550	4,550	4,550	0%	0%	1.2	**0.2**	1	1	0
S165		6,160	8%	0.02	5,712	5,712	5,712	5,712	0%	0%	1.2	**0.2**	1	1	0
S175		4,715	17%	0.02	4,014	4,014	4,014	4,014	0%	0%	5.5	**0.3**	1	1	0
S185		4,171	11%	0.02	3,770	3,770	3,770	3,770	0%	0%	1.2	**0.2**	1	1	0
S195		5,378	0%	0.02	5,378	5,378	5,378	5,378	0%	0%	1.2	**0.2**	1	1	0
S225		5,157	12%	0.02	4,602	4,602	4,602	4,602	0%	0%	1.5	**0.2**	1	1	0
S245		5,378	5%	0.01	5,098	5,098	5,098	5,098	0%	0%	1.9	**0.2**	1	1	0
S255		3,608	4%	0.02	3,483	3,483	3,483	3,483	0%	0%	1.3	**0.2**	1	1	0
S274	13	7,238	28%	0.13	5,667	5,667	5,667	5,667	0%	0%	810.6	**56.8**	1	2,946	1
S284		6,384	9%	0.11	5,837	5,837	5,837	5,837	0%	0%	976.9	**34.7**	1	652	1
S155		6,748	20%	0.16	5,637	5,637	5,637	5,637	0%	0%	973.8	**689.5**	1	12,354	0
S165		7,654	9%	0.15	7,004	7,004	7,004	7,004	0%	0%	1,192.2	**695.5**	1	5,310	1
S175		6,538	25%	0.13	5,245	5,245	5,245	5,245	0%	0%	1,563.8	**318.1**	12,247	7,988	8
S185		5,191	10%	0.18	4,709	4,709	4,709	4,709	0%	0%	1,743.9	**58.2**	1	1,954	3
S195		6,320	6%	0.17	5,938	5,938	5,938	5,938	0%	0%	2,499.7	**111.9**	1	7,009	0
S225		6,081	11%	0.16	5,465	5,465	5,465	5,465	0%	0%	3,124.5	**158.8**	3,991	3,857	0
S245		6,499	5%	0.18	6,190	6,190	6,190	6,190	0%	0%	1,567.1	**412.7**	4,227	9,224	0
S255		6,932	14%	0.11	6,087	6,087	6,087	6,087	0%	0%	2,323.2	**233.5**	1,627	5,888	0
S274	17	10,088	31%	0.61	7,911	7,723	4,789	7,723	39%	0%	>14,400	**13,637.8**	506	124,299	6
S284		6,635	6%	0.70	6,258	6,258	4,646	6,258	26%	0%	>14,400	**6,476.2**	2,310	28,626	7
S155		8,657	17%	0.60	7,912	7,374	3,889	7,374	51%	0%	>14,400	**2,679.8**	1,768	512,559	60
S165		8,311	9%	0.84	7,654	7,654	3,998	7,654	48%	0%	>14,400	**3,963.9**	504	209,028	0
S175		9,208	18%	0.61	8,729	7,773	4,612	7,773	47%	0%	>14,400	**3,991.2**	591	418,295	1
S185		6,548	8%	1.04	6,073	6,073	3,251	6,073	46%	0%	>14,400	**4,022.7**	1,152	805,601	72
S195		7,426	8%	1.03	6,892	6,892	3,724	6,892	46%	0%	>14,400	**13,264.3**	1,183	1,730,630	7
S225		7,139	9%	0.94	6,795	6,522	2,890	6,522	57%	0%	>14,400	**13,703.3**	363	165,228	108
S245		6,727	5%	1.02	6,679	6,418	4,254	5,263	36%	18%	>14,400	**>14,400**	816	2,480,610	8
S255		7,879	20%	0.71	7,260	6,565	3,222	6,565	56%	0%	>14,400	**7,581.6**	74	683,801	124

30 instances) in Table 6.7. This indicates that the heuristic reduced the solvers' effort to identify an optimal solution for the exact approaches considerably. Even for medium-sized instances with 17 customer requests, the runtime did not exceed 2 seconds.

Comparing the results between the other two exact approaches, MPDPTH-E outperformed MPDPTH-C in all 30 instances. Out of the 30 instances, MPDPTH-E and MPDPTH-C solved 97% and 67% of instances to optimality within the 4-hour time limit respectively. For the optimally resolved instances, MPDPTH-E was pervasively faster than MPDPTH-C with an average runtime savings of 85%. MPDPTH-E and MPDPTH-C timed-out in 1 and 10 instances respectively. Among them, all instances had 17 customer requests. So, the runtime of the two approaches increased significantly in the mid-size instances with 17 requests. However, MPDPTH-E terminated with smaller integrality gap than MPDPTH-E in the one timed-out instance.

An issue of MPDPTH-E, when compared with MPDPTH-C, is the higher number of BB nodes across 18 out of 30 instances. Especially the numbers of BB nodes for MPDPTH-E in instances S185, S195, and S245 with $n = 17$ are very high, which indicates a hard effort by the solver during the branch-and-cut procedure. In contrast, MPDPTH-C solved 16 instances with just one BB node.

After a close inspection of their solution structures, we identified instance characteristics that reduce the runtime for our branch-and-cut algorithm. MPDPTH-E runtime was relatively short for S155, S165, and S175 ($n = 17$). These instances had a lot of short-haul customer requests (lesser than 250 miles) with low cargo weights ($<$15,000 lbs), and the solution identified short-haul routes that did not exceed 250 miles. In contrast, instances S245 and S195 ($n = 17$) have long runtime and identified routes with very long distances. This was because the solver was able to find effective consolidation options for the short-haul requests with low weights, whereas long-haul requests were typically high weight and the solver spent more effort to find effective consolidated routes.

Table 6.8: MPDPTH results in instances with 21 ans 25 requests (two approaches)

Instance	n	WS Heuristic			MPDPTH-E				
		Cost	Diff	Secs	Cost	LB	Gap	#BB Nodes	#cuts
S274	21	11,947	29%	2.47	9,297	5,443	41%	372,968	11
S284		9,110	24%	1.75	7,333	5,814	21%	535,910	89
S155		9,299	4%	2.68	8,966	4,776	47%	130,701	19
S165		9,830	6%	1.56	9,267	5,773	38%	216,981	2
S175		10,580	11%	2.66	9,508	6,857	28%	490,827	102
S185		7,657	14%	3.95	6,730	4,019	40%	271,529	54
S195		8,486	7%	3.05	7,932	4,834	39%	473,485	73
S225		9,287	12%	2.95	8,295	4,113	50%	361,789	7
S245		7,561	0.1%	4.34	7,551	4,619	39%	401,610	61
S255		8,812	13%	2.62	7,817	4,565	42%	384,944	24
S274	25	13,648	9%	4.38	12,530	4,231	66%	136,387	41
S284		10,052	3%	3.69	9,751	5,475	44%	121,031	58
S155		11,596	3%	4.96	11,263	4,715	58%	45,450	39
S165		12,078	3%	3.13	11,724	4,941	58%	37,248	10
S175		11,679	4%	4.77	11,206	6,172	45%	126,254	79
S185		8,816	12%	6.84	7,848	4,837	38%	135,184	60
S195		9,256	8%	5.81	8,587	4,458	48%	313,366	76
S225		10,222	3%	5.1	9,908	3,902	61%	36,810	23
S245		7,909	1%	7.86	7,857	4,760	39%	270,123	17
S255		10,196	13%	4.88	9,012	4,791	47%	221,207	35

Since MPDPTH-E exhibited better performance than MPDPTH-C on small and medium sized instances, we measured the scalability of MPDPTH-E alone on larger instances with 21 and 25 customer requests in Table 6.8. This table presents the following details for MPDPTH-E: the best UB, integrality gap between the best UB and the best LB, the number of BB nodes, and the number of lazy cuts added after solving the separation problems. The table also shows the following WS heuristic results: best objective cost, % difference against the best UB identified by MPDPTH-E and runtime. We have not presented the runtime for MPDPTH-E here because all large instances timed-out (>14,400 secs) in that approach. However, the MPDPTH-E integrality gap is within 50% for 80% of the instances (16 out of 20). One way to reduce the MPDPTH-E runtime is to explore the linear relaxation tightening techniques. This could help to close the integrality gap faster by tightening the lower bound in the branch-and-bound framework. Similar to Table 6.7 results, the high number of BB nodes for MPDPTH-E is an issue for all instances. The number of handling cost lazy cuts is low even for the large instances. One potential reason for this is that the handling cost enforcing inequalities (5.25)-(5.27) enforced many handling cost cuts in the initial cut pool. This reduced the number of handling cost lazy cuts later in the branch-and-cut approach.

The WS heuristic performance is very effective even in large instances. From Table 6.8 results, we can see that the heuristic runtime did not exceed 8 seconds for instances with $n = 25$. Moreover, the WS objective is within 15% of the best UB for 90% of the large instances (18 out of 20). This means that the WS heuristic provided a decent starting solution. Furthermore, the exact approaches did not find a UB which was drastically different than the WS heuristic solution. Especially for instance S245, WS performed so well that the exact approach solution was barely better than the WS solution.

6.4 Multi Vehicle Pickup-and-Delivery Problem with Time Windows and Loading Constraints

In this section, we present our experimental set-up, test-bed details and computational results of our MPDPTL methodologies. The MPDPTL implementation details are similar to MPDPTH.

Remarks on computational results

Table 6.9 shows the performance of MPDPTL1 and MPDPTL2 in 30 instances comprising of 9, 13 or 17 customer requests. We have also presented the results from our MPDPTL Warm Start (WS) heuristic. Similar to MPDPTH, we presented the WS details here because it identified decent results for MPDPTL with very short runtime. Table 6.9 shows the following WS heuristic results: best objective cost, % difference against the best upper bound (from exact approaches), and runtime. The table also shows the following results for MPDPTL1 and MPDPTL2: the best objective cost or Upper Bound (UB), the best Lower Bound (LB), % gap between UB and LB, runtime, and the number of BB nodes.

The efficiency of MPDPTL WS heuristic is very close to the MPDPTH WS heuristic. It identified good results with very small runtime. The heuristic solutions are within 20% of the best UB solution for 87% of instances (26 out of 30 instances) in Table 6.9. The runtime did not exceed 2 seconds even for medium-sized instances with 17 customer requests.

Table 6.9: MPDPTL results in instances with 9, 13 and 17 requests (three approaches)

Instance	n	WS Heuristic			Cost		Lower Bound (LB)		Gap		Secs		#BB Nodes	
		Cost	Diff	Secs	MPDPTL1	MPDPTL2	MPDPTL1	MPDPTL2	MPDPTL1	MPDPTL2	MPDPTL1	MPDPTL2	MPDPTL1	MPDPTL2
SL274	9	5,914	14%	0.02	5,185	5,185	5,185	5,185	0%	0%	0.2	**0.2**	1	1
SL284		6,492	12%	0.02	5,793	5,793	5,793	5,793	0%	0%	0.2	**0.2**	1	1
SL155		5,564	12%	0.02	4,986	4,986	4,986	4,986	0%	0%	0.2	**0.2**	1	1
SL165		6,650	8%	0.02	6,166	6,166	6,166	6,166	0%	0%	0.2	**0.2**	1	1
SL175		5,054	17%	0.01	4,303	4,303	4,303	4,303	0%	0%	0.2	**0.2**	1	1
SL185		4,366	11%	0.02	3,946	3,946	3,946	3,946	0%	0%	0.2	**0.2**	1	1
SL195		5,690	0%	0.01	5,690	5,690	5,690	5,690	0%	0%	0.2	**0.2**	1	1
SL225		5,516	12%	0.02	4,922	4,922	4,922	4,922	0%	0%	0.2	**0.2**	1	1
SL245		5,931	5%	0.02	5,622	5,622	5,622	5,622	0%	0%	0.2	**0.2**	1	8
SL255		3,832	4%	0.01	3,699	3,699	3,699	3,699	0%	0%	0.2	**0.2**	1	1
SL274	13	7,757	28%	0.16	6,073	6,073	6,073	6,073	0%	0%	160.2	**82.7**	6,535	21,272
SL284		6,967	9%	0.14	6,370	6,370	6,370	6,370	0%	0%	**26.0**	30.7	747	2,584
SL155		6,990	20%	0.16	5,839	5,839	5,839	5,839	0%	0%	**197.0**	394.0	7,149	7,477
SL165		8,093	9%	0.15	7,406	7,406	7,406	7,406	0%	0%	396.1	**309.7**	4,901	15,198
SL175		6,785	25%	0.17	5,443	5,443	5,443	5,443	0%	0%	**281.8**	285.4	9,587	7,420
SL185		5,369	10%	0.17	4,871	4,871	4,871	4,871	0%	0%	213.6	**78.8**	5,392	8,933
SL195		6,743	6%	0.17	6,336	6,336	6,336	6,336	0%	0%	141.1	**65.2**	10,256	80,132
SL225		6,514	11%	0.15	5,854	5,854	5,854	5,854	0%	0%	245.8	**133.8**	15,739	9,008
SL245		6,895	5%	0.18	6,568	6,568	6,568	6,568	0%	0%	**545.1**	835.8	6,918	141,605
SL255		7,535	14%	0.14	6,616	6,616	6,616	6,616	0%	0%	157.6	**110.4**	4,350	4,921
SL274	17	10,697	31%	0.74	8,189	8,189	8,189	7,812	0%	5%	**1,347.4**	1,351.7	189,621	628,694
SL284		7,166	6%	0.81	6,759	6,759	6,759	6,759	0%	0%	**3,133.5**	5,777.6	15,987	42,461
SL155		9,401	17%	0.65	8,008	8,008	8,008	8,008	0%	0%	**3,278.4**	3,617.2	564,744	782,763
SL165		8,722	9%	0.85	8,032	8,032	8,032	8,032	0%	0%	5,502.3	**2,167.1**	876,171	773,562
SL175		9,937	18%	0.75	8,389	8,389	8,255	8,389	2%	0%	>14,400	**5,935.8**	3,553,410	1,236,180
SL185		6,871	8%	1.03	6,372	6,372	6,372	6,372	0%	0%	3,730.5	**3,534.2**	677,898	1,513,680
SL195		7,788	8%	0.98	7,228	7,228	7,228	7,228	0%	0%	12,485.8	**12,043.9**	2,317,270	1,102,040
SL225		7,867	9%	0.92	7,187	7,187	7,187	7,187	0%	0%	**11,160.5**	11,245.7	73,865	915,503
SL245		7,356	5%	1.06	7,018	7,018	5,928	7,018	16%	0%	>14,400	**4,976.5**	3,208,040	724,386
SL255		8,439	20%	0.75	7,031	7,031	7,031	7,031	0%	0%	4,093.2	**3,435.2**	531,349	525,147

Comparing the results between the two exact approaches, MPDPTL2 outperformed MPDPTL1 in 73% of instances. Out of the 30 instances, MPDPTL2 and MPDPTL1 solved 100% and 93% of instances to optimality within the 4-hour time limit respectively. For the optimally resolved instances in which MPDPTL2 was faster than MPDPTL1, the average runtime savings was 28%. MPDPTL1 timed-out in 2 instances. Both instances had 17 customer requests. The runtime of the two approaches increased significantly in the mid-size instances with 17 requests. With these observations, MPDPTL2 is the clear winner among exact approaches based on the runtime.

The number of BB nodes is very low for both approaches in small instances with $n = 9$. For remaining instances in Table 6.9, MPDPTL2 has higher number of BB nodes in 13 instances and MPDPTL2 has higher number of BB nodes in the remaining 7 instances. So, a comparison between the two approaches based on the number of BB nodes does not yield an obvious winner. However, the number of nodes is high for both approaches which suggest a hard branching effort by the solver. Similar to MPDPTH approaches, MPDPTL2 runtime was relatively short in instances that had a lot of short-haul customer requests with low weights (SL155, S165, and SL274). In contrast, instances that identified routes with very long distances had relatively long runtime (for example-SL225 and SL195).

Since MPDPTL2 exhibited better performance than MPDPTL1 in Table 6.9, we measured the scalability of MPDPTL2 alone on larger instances with 21 and 25 customer requests in Table 6.10. This table presents the following details for MPDPTL2: the best UB, integrality gap between the best UB and the best LB, the

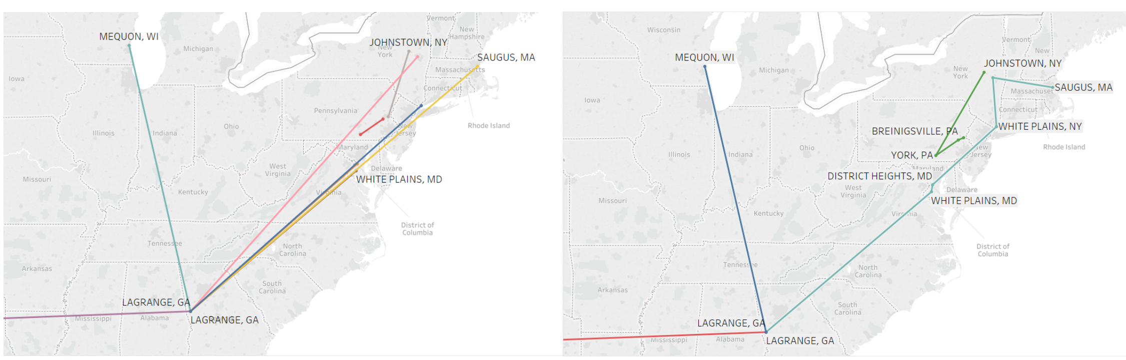
Table 6.10: MPDPTL results in instances with 21 ans 25 requests (two approaches)

Instance	n	WS Heuristic			MPDPTL2				
		Cost	Diff	Secs	Cost	LB	Gap	#BB Nodes	#cuts
SL274	21	12,406	29%	2.98	9,654	5,652	41%	372,968	11
SL284		9,809	24%	2.87	7,896	6,260	21%	535,910	89
SL155		10,091	4%	2.83	9,730	4,989	49%	130,701	19
SL165		10,508	6%	2.14	9,906	6,171	38%	216,981	2
SL175		11,290	11%	3.27	10,146	7,317	28%	490,827	102
SL185		8,036	14%	3.75	7,063	4,218	40%	271,529	54
SL195		9,056	7%	3.47	8,465	5,159	39%	473,485	73
SL225		10,105	12%	3.22	9,026	4,475	50%	361,789	7
SL245		8,109	0.1%	4.34	8,098	4,954	39%	401,610	61
SL255		9,670	13%	2.81	8,578	5,009	42%	384,944	45
SL274	25	14,070	9%	5.64	12,916	4,361	66%	136,387	41
SL284		10,752	3%	5.52	10,430	5,856	44%	121,031	58
SL155		12,003	3%	5.49	11,659	4,881	58%	45,450	39
SL165		13,435	4%	4.45	12,862	5,421	58%	37,248	10
SL175		12,582	4%	6	12,073	6,649	45%	126,254	79
SL185		9,220	12%	6.82	8,208	5,059	38%	135,184	60
SL195		9,875	6%	6.49	9,276	4,816	48%	313,366	76
SL225		11,084	4%	5.93	10,626	4,185	61%	36,810	23
SL245		8,320	1%	7.84	8,265	5,007	39%	270,123	17
SL255		10,577	13%	5.42	9,349	4,970	47%	221,207	60

number of BB nodes, and the number of lazy cuts added after solving the separation problems. The table also shows the following WS heuristic results: best objective cost, % difference against the best UB identified by MPDPTL2 and runtime. We have not presented the runtime for MPDPTL2 here because all large instances timed-out (>14,400 secs) in that approach. However, the integrality gap is below 50% for 80% of the instances (16 out of 20). The high number of BB nodes is an issue in all large instances.

Similar to the MPDPTH WS heuristic, the MPDPTL WS performance is very effective even on the large instances. From Table 6.10 results, we can see that the heuristic runtime is less than 8 seconds for instances with n=21 or 25. Moreover, the WS objective is within 15% of the best UB for 90% of the large instances (18 out of 20). This means that the WS heuristic provided a decent starting solution. Furthermore, the exact approaches did not find a UB which was drastically different than the WS heuristic solution. Especially, in instances SL245, SL284 and SL155 WS performed so well that the exact approach solution was barely better than the WS solution.

Overall, our model consistently identified cheaper routes with a lesser number of trucks than the actual routes. Figure 6.2(a) shows actual routes along the east coast for one of the instances. Six trucks were used in total to fulfill the customer requests, whereas Figure 6.2(b) shows our model results for the same requests using only 3 trucks.



(a) Actual routes (b) Routes from our model

Figure 6.2: MPDPTL result on east coast for one instance

CHAPTER VII

CONCLUSION AND FUTURE WORKS

In this chapter, we summarize the contributions of this dissertation to the vehicle routing problem literature and some future research directions.

7.1 Research contributions

In this dissertation, we investigate four closely related problems:

- Single vehicle Pickup-and-Delivery Problem with LIFO Loading constraints
- Single vehicle Pickup-and-Delivery Problem with Handling costs
- Multi-vehicle Pickup-and-Delivery Problem with Time windows and LIFO Loading constraints
- Multi-vehicle Pickup-and-Delivery Problem with Time windows and Handling costs

Our contributions to these four problems are tabulated in Figure 7.1.

For the first problem, we introduced a new branch-and-cut algorithm. One of the notable contributions of this dissertation is the introduction of new conditional integral separation procedures to identify violated inequalities. We also proposed

	LIFO	Handling cost
Single vehicle	**From literature:** 1. A cut-based formulation 2. Branch-and cut algorithm-fractional separation procedures **Our contributions:** Branch-and cut algorithm with integral separation procedures	**From literature:** 1. A compact formulation **Our contributions:** 1. A compact formulation 2. A cut-based formulation 3. Branch-and cut algorithm with fractional separation 4. Branch-and cut algorithm with integral separation
Multi vehicle	**Unexplored in literature** **Our contributions:** 1. Two formulations 2. Branch-and cut algorithm 3. A heuristic based on potential savings from shipping requests together	**Unexplored in literature** **Our contributions:** 1. Two formulations 2. Branch-and cut algorithm 3. A heuristic based on tradeoff b/w travel distance and LIFO enforcement

Figure 7.1: Research contributions for the four problems

new inequalities to speed up the algorithm. Other runtime improvements like upper bound tightening, warm start, and preprocessing were explored. Our algorithm outperformed a branch-and-cut algorithm with fractional separation procedures from the literature in all the test instances by the runtime.

For the second problem, we introduced a compact formulation, a cut-based formulation, and two branch-and-cut algorithms (one with fractional separation procedures and another with integral separation procedures). We inspected the solutions to identify the pathological characteristics of our algorithm. Our approach was very effective in the instances in which each customer requests' delivery node was close to its pickup node in the solution sequence. This is because it prompts less effort for precedence and handling cost enforcements. In contrast, the runtime increased when the solution had a nested structure, which means the pickup and the delivery nodes for some customer requests were placed far away from each other in the solution sequence.

The third problem is newly introduced in this dissertation. The objective of this problem is to route multiple vehicles in a homogeneous fleet. The routing is subject to vehicle capacity, time windows at customer locations, maximum time on-road for a driver, and LIFO loading/unloading order. We presented two formulations and two branch-and-cut algorithms. A warm start heuristic was explored to provide a starting solution in a branch-and-bound framework for the other procedures. This heuristic identified decent solutions for all instances with a very short runtime.

The fourth problem is newly introduced in this dissertation. This problem is similar to the third problem, but we permit loading/unloading LIFO violations with handling costs. We presented a compact formulation and a formulation with an exponential number of constraints. We also presented a branch-and-cut algorithm to computationally implement the latter formulation. The branch-and-cut algorithm uses integral separation procedures to identify inequalities violating the LIFO order for loading/unloading. We presented new inequalities for handing cost enforcement which reduced our runtime. We inspected the solutions to identify the instance characteristics that could impact the runtime of our algorithm positively. Our approach was very effective in the instances where short-haul routes were identified in the solution ($<$250 miles). In contrast, the runtime increased when the solution had relatively long-distance routes.

7.2 Future Works

We solved the handling cost problems in this dissertation with the assumption that the shuffling of cargo will not be permitted. It means that the additional shipments unloaded at customer sites cannot be shuffled on the ground before getting reloaded into the vehicle. Figure 7.2 illustrates this assumption. It would be interesting to explore the handling cost problems with shuffling. A heuristic that permits shuffling in the single-vehicle handling cost problem has already been explored by Veenstra et al. [42]. However, shuffling cargo in a multi-vehicle setting has not been explored for handling cost problems.

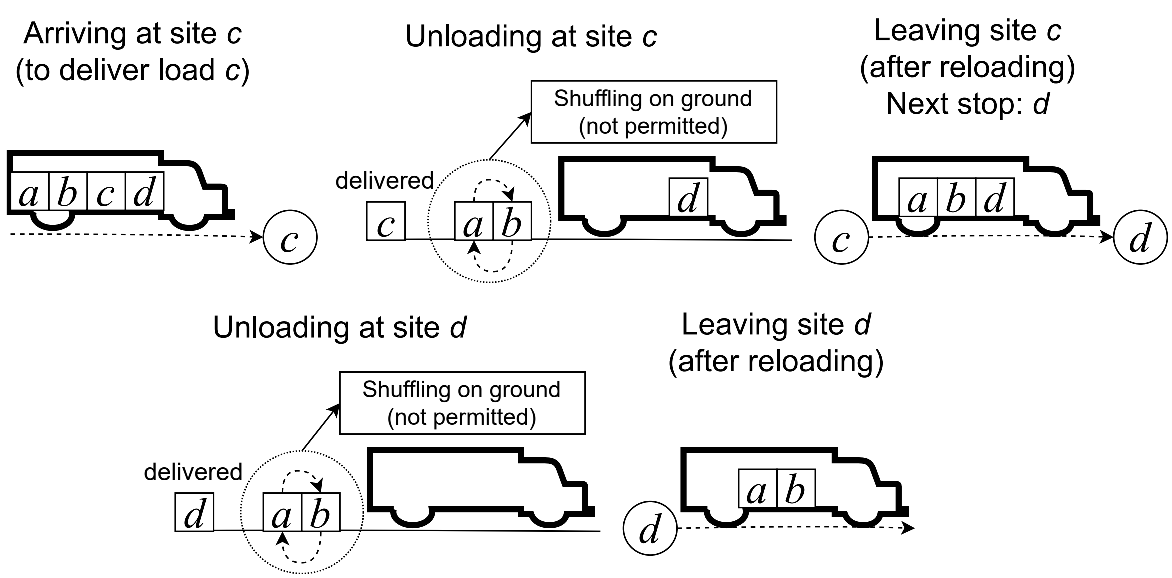


Figure 7.2: Shuffling option for cargo handling

Variety of VRP problems in the literature has been solved with branch-and-price algorithms. However, branch-and-price approaches have not been explored for the four problems in this dissertation. Although we have explored some fractional separation procedures in this dissertation, the primary focus has been on integral separation techniques that target the upper bound of the solutions. So, there are some computational results in which the upper bound of the solutions closed the integrality gap very quickly than the lower bounds. One way to resolve this issue is to explore the linear relaxation tightening techniques.

The conditional integral separation procedures introduced in the single-vehicle problem is an important contribution of this dissertation. This approach is applicable in other formulations with exponential sets of constraints. Identifying other problems where the conditional integral separation procedures are applicable is another possible future research direction.

BIBLIOGRAPHY

[1] Y. Adulyasak, J-F. Cordeau, and R. Jans. Formulations and branch-and-cut algorithms for multivehicle production and inventory routing problems. *INFORMS Journal on Computing*, 26(1):103–120, 2014.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Network flows. 1988.

[3] American Trucking Association. Reports, trends and statistics. http://www.trucking.org/News_andInformationReportsIndustryData.aspx.

[4] American Trucking Associations. New report finds trucking industry revenues topped 700 billion. https://www.trucking.org/article/New-Report-Finds-Trucking-Industry-Revenues-Topped-\$700-Billion.

[5] E. Balas, M. Fischetti, and W. Pulleyblank. The precedence-constrained asymmetric traveling salesman polytope. *Mathematical programming*, 68(1-3):241–265, 1995.

[6] M. Battarra, G. Erdoğan, G. Laporte, and D. Vigo. The traveling salesman problem with pickups, deliveries, and handling costs. *Transportation Science*, 44(3):383–399, 2010.

[7] E. Benavent, M. Landete, E. Mota, and G. Tirado. The multiple vehicle pickup and delivery problem with lifo constraints. *European Journal of Operational Research*, 243(3):752–762, 2015.

[8] Business Insider. 11 incredible facts about the $700 billion us trucking industry. "https://markets.businessinsider.com/news/stocks/trucking-industry-facts-ustruckers-2019-5-1028248577#in-2017-the-american-trucking-industry-posted-revenues-higher-\\than-the-gdp-of-more-than-150-nations-1".

[9] California Department of Transportation. Trucking industry overview. http://www.dot.ca.gov/hq/tpp/offices/ogm/keyreportsfiles/National,\%20Technical\%20studies/Truckingindustryoverview.pdf.

[10] F. Carrabs, R. Cerulli, and J-F. Cordeau. An additive branch-and-bound algorithm for the pickup and delivery traveling salesman problem with lifo or fifo loading. *INFOR: Information Systems and Operational Research*, 45(4):223–238, 2007.

[11] F. Carrabs, J-F. Cordeau, and G. Laporte. Variable neighborhood search for the pickup and delivery traveling salesman problem with lifo loading. *INFORMS Journal on Computing*, 19(4):618–632, 2007.

[12] L. Cassani. Algoritmi euristici per il tsp with rear-loading. *Degree thesis, Universita di Milano, Italy*, 2004.

[13] Chair in Logistics and Transportation, HEC Montreal. Scientific Data: TSPPDL-BB. http://chairelogistique.hec.ca/data/TSPPDL-BB/.

[14] M. Cherkesly, G. Desaulniers, and G. Laporte. Branch-price-and-cut algorithms for the pickup and delivery problem with time windows and last-in-first-out loading. *Transportation Science*, 49(4):752–766, 2014.

[15] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

[16] L.C. Coelho, J-F. Cordeau, and G. Laporte. Consistency in multi-vehicle inventory-routing. *Transportation Research Part C: Emerging Technologies*, 24:270–287, 2012.

[17] J-F Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3):573–586, 2006.

[18] J-F. Cordeau, M. Iori, G. Laporte, and J. J. Salazar González. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with lifo loading. *Networks*, 55(1):46–59, 2010.

[19] J-F. Cordeau, G. Laporte, J-Y. Potvin, and M. Savelsbergh. Transportation on demand. *Handbooks in operations research and management science*, 14:429–466, 2007.

[20] J-F. Cordeau, G. Laporte, and S. Ropke. Recent models and algorithms for one-to-one pickup and delivery problems. In *The vehicle routing problem: latest advances and new challenges*, pages 327–357. Springer, 2008.

[21] J-F. Côté, C. Archetti, M. G. Speranza, M. Gendreau, and J-Y. Potvin. A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple stacks. *Networks*, 60(4):212–226, 2012.

[22] J-F. Côté, M. Gendreau, and J-Y. Potvin. Large neighborhood search for the pickup and delivery traveling salesman problem with multiple stacks. *Networks*, 60(1):19–30, 2012.

[23] G. Dantzig and J. Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

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

[25] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

[26] Environmental Defense Fund. Green freight facts and figures. http://business.edf.org/projects/greenfreight-freight-facts-figures.

[27] G. Erdogan, M. Battarra, G. Laporte, and D. Vigo. Metaheuristics for the traveling salesman problem with pickups, deliveries and handling costs. *Computers & Operations Research*, 39(5):1074–1086, 2012.

[28] Federal Motor Carrier Safety Administration. Interstate truck driver's guide to hours of service. https://www.fmcsa.dot.gov/sites/fmcsa.dot.gov/files/docs/Drivers\%20Guide\%20to\%20HOS\%202015_508.pdf.

[29] M. Gendreau, M. Iori, G. Laporte, and S. Martello. A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. *Networks*, 51(1):4–18, 2008.

[30] S. Ladany and A. Mehrez. Optimal routing of a single vehicle with loading and unloading constraints. *Transportation Planning and Technology*, 8(4):301–306, 1984.

[31] A. Malapert, C. Guéret, N. Jussien, A. Langevin, and L-M. Rousseau. Two-dimensional pickup and delivery routing problem with loading constraints. In *First CPAIOR Workshop on Bin Packing and Placement Constraints (BPPC'08)*, page 184, 2008.

[32] Owner–Operator Independent Drivers Association. Lumping regulations. https://www.ooida.com/EducationTools/Resources/lumping-regs.asp.

[33] J Pacheco. Problemas de rutas con carga y descarga en sistemas lifo: soluciones exactas. *Estudios de economía aplicada*, (3):69–86, 1995.

[34] J. Pacheco. Heurístico para los problemas de rutas con carga y descarga en sistemas lifo, 1997.

[35] D. Radha Krishnan and T. Liu. A branch-and-cut algorithm for pickup-and-delivery traveling salesman problem with handling costs. *Manuscript submitted for publication*, 2020.

[36] S. Ropke and J-F. Cordeau. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, 43(3):267–286, 2009.

[37] K.S. Ruland and E.Y. Rodin. The pickup and delivery problem: Faces and branch-and-cut algorithm. *Computers & mathematics with applications*, 33(12):1–13, 1997.

[38] S. Sarin, H. Sherali, and A. Bhootra. New tighter polynomial length formulations for the asymmetric traveling salesman problem with and without precedence constraints. *Operations Research Letters*, 33(1):62–70, 2005.

[39] H.D. Sherali and J.C. Smith. Improving discrete model representations via symmetry considerations. *Management Science*, 47(10):1396–1407, 2001.

[40] The Truckers Report. The real cost of trucking – per mile operating cost of a commercial truck. https://www.thetruckersreport.com/infographics/cost-of-trucking/.

[41] P. Toth and D. Vigo. *Vehicle routing: problems, methods, and applications.* SIAM, 2014.

[42] M. Veenstra, K. J. Roodbergen, I. F. Vis, and L. C. Coelho. The pickup and delivery traveling salesman problem with handling costs. *European Journal of Operational Research*, 257(1):118–132, 2017.

APPENDIX A

LINEARIZATION OF PRODUCT OF TWO VARIABLES

Let us consider non-linear Constraints (4.4) from Formulation (4.2.1) for an illustration of our linearization technique. For an arc $(i, j) \in A$, inequality (4.4) is

$$Q_j \geq (Q_i + q_j)x_{ij} \tag{1.1}$$

Right hand side becomes $Q_i x_{ij} + q_j x_{ij}$, where Q_i is a continuous variable, x_{ij} is a binary variable and $Q_i x_{ij}$ is a non-linear term. From Formulation (4.2.1), note that Q_i is bounded below by 0 and above by Q. To linearize this constraint, we present new variables α_{ij} for each arc $(i, j) \in A$. Now we replace (1.1), with following inequalities

$$Q_j \geq \alpha_{ij} + q_j x_{ij}$$

$$\alpha_{ij} \leq Q x_{ij}$$

$$\alpha_{ij} \leq Q_j$$

$$\alpha_{ij} \geq Q_j - (1 - x_{ij})Q$$

$$\alpha_{ij} \geq 0$$

APPENDIX B

ALGORITHM STRUCTURES

2.1 Breadth first search

Algorithm 1 BFS Algorithm- Function

Require: A directed graph $G = (N, A)$, flow values $0 \leq r_{ij} \leq 1 \ \forall (i, j) \in A$ and source node s

```
initialize PRED[|N|] := NULL, and LIST := {s}
unmark all nodes in N, mark root node s
while LIST is not empty do
    i := LIST[0] (First entry in the LIST)
    remove i from LIST
    for j=0 to 2n + 1 do
        if node i is incident to an arc (i, j), such that r_ij > 0 then
            if node j is unmarked then
                mark j and PRED[j] := i
            end if
        end if
    end for
end while
return PRED[|N|]
```

2.2 Fractional separation problem- Maximum flow algorithm

Algorithm 2 Edmonds-Karp Algorithm

Require: A directed graph $G = (N, A)$, fractional values $x \in \{0, 1\}^{|A|}$, source node s and sink node t

initialize residual graph capacities $r_{ij} := x_{ij}\ \forall (i, j) \in A$, $MaxFlow := 0$ and $PRED[|N|] := NULL$
while sink t is reachable from s on residual graph (call **BFS** function) **do**
    obtain $PRED$ list
    $i := t$ and $flow := \infty$
    while $i \neq s$ **do**
        $j := PRED[i]$
        $flow := min\{flow, r_{ji}\}$
        $i := j$
    end while
    $i := t$
    while $i \neq s$ **do**
        $j := PRED[i]$
        $r_{ji} := r_{ji} - flow$
        $i := j$
    end while
    $MaxFlow := flow + MaxFlow$
end while
return $MaxFlow$

2.3 Integral separation problem- Sub-tour elimination

Algorithm 3 ISP1

Require: A directed graph $G = (N, A)$, binary values $x \in \{0, 1\}^{|A|}$ defining a PC-tuple

```
unmark all nodes in N, mark root node 0
i := 0; p := 0; PATH := ∅; SUBTOURS[ ] := ∅
while i ≠ 2n + 1 do
    if node i is incident at an arc (i, j), such that x_ij = 1 then
        PATH := PATH ∪ {i}, mark node i
        i := j
    end if
end while
for i=1 to 2n do
    if i is unmarked then
        if node i is incident to an arc (i, j), such that x_ij = 1 then
            SUBTOURS[p] := SUBTOURS[p] ∪{i}, mark i, k := i and i := j
            while j ≠ k do
                if node i is incident to an arc (i, j), such that x_ij = 1 then
                    SUBTOURS[p] := SUBTOURS[p] ∪{i}, mark node i
                    i := j
                end if
            end while
            p := p + 1
            break for-loop
        end if
    end if
end for
```

2.4 Integral separation problem- Precedence violation

Algorithm 4 ISP2

Require: A directed graph $G = (N, A)$, binary values $x \in \{0,1\}^{|A|}$ defining a Hamiltonian path

unmark all nodes in N, mark root node 0
$i := 0$; $p := 0$; $q := 0$; $PATH := \emptyset$; $POS[|N|] := \emptyset$; $ORDER[|N|] := \emptyset$; $PRECSET := \emptyset$
while $i \neq 2n+1$ **do**
    if node i is incident at an arc (i,j), such that $x_{ij} = 1$ **then**
        $PATH := PATH \cup \{i\}$, mark node i
        $POS[i] := p$
        $ORDER[p] := i$
        $i := j$ and $p := p+1$
    end if
end while
for i=1 to n **do**
    if $POS[n+i] < POS[i]$ **then**
        for k=$POS[0]$ to $POS[i]$-1 **do**
            $PRECSET := PRECSET \cup \{ORDER[POS[k]]\}$
        end for
    end if
end for

2.5 Integral separation problem- LIFO violation

Algorithm 5 ISP3

Require: A directed graph $G = (N, A)$, binary values $x \in \{0, 1\}^{|A|}$ defining a Hamiltonian path with no precedence violations

```
unmark all nodes in N, mark root node 0
i := 0; p := 0; q := 0; PATH := ∅; POS[|N|] := ∅; ORDER[|N|] := ∅; LIFOSET := ∅
while i ≠ 2n + 1 do
    if node i is incident at an arc (i, j), such that x_ij = 1 then
        PATH := PATH ∪ {i}, mark node i
        POS[i] := p
        ORDER[p] := i
        i := j and p := p + 1
    end if
end while
for i=1 to n do
    for j=1 to n do
        if POS[i] < POS[j] then
            if POS[j] < POS[n + i] then
                if POS[n + i] < POS[n + j] then
                    for k=POS[i]+1 to POS[n + j]-1 do
                        LIFOSET := LIFOSET ∪ {ORDER[POS[k]]}
                    end for
                end if
            end if
        end if
    end for
end for
```

2.6 Clarke-Wright algorithm

Algorithm 6 Savings algorithm

Require: A directed graph $G = (N, A)$, cost values c_{ij} for each arc $(i, j) \in A$

```
unmark all nodes in N
p := 0; PATH := ∅; ARCLIST := ∅
for each i(i, j) ∈ A do
    s_ij := c_i0 + c_0j = c_ij
end for
Sort s_ij in descending order and correspondingly store arcs in ARCLIST
for p=1 to |A| do
    if both nodes in ARCLIST[p] are unmarked then
        Create a new row in PATH and add ARCLIST[p]
        Mark nodes i and j in ARCLIST[p]
    end if
    if one of the nodes in ARCLIST[p] is unmarked then
        if marked node in ARCLIST[p] is at the edge of a row in PATH then
            Append ARCLIST[p] to corresponding row
            Mark nodes i and j in ARCLIST[p]
        end if
    end if
    if both nodes in ARCLIST[p] are marked then
        if nodes in ARCLIST[p] are at two PATH row edges then
            Merge the corresponding PATH rows
            Remove one of the two PATH rows
        end if
    end if
    if PATH[0] has 2n nodes then
        Break
    end if
end for
```

2.7 LIFO greedy search algorithm

Algorithm 7 Warm start heuristic for SPDPL

Require: A directed graph $G = (N, A)$, customer requests n, cost values c_{ij} for each arc $(i, j) \in A$

```
PATH := ∅; UNVISITED := {1, ..., 2n}
PATH[0] := 0
while UNVISITED ≠ ∅ do
    i := 0
    MINCOST := ∞
    for each (i, j) ∈ A do
        if (i, j) does not violate LIFO on partial PATH then
            if c_ij < MINCOST then
                j := i
                Remove j from UNVISITED
            end if
        end if
    end for
end while
PATH[2n + 1] := 2n + 1
for i = 1 to n do
    NEWPATH := PATH
    Remove i and n + i from PATH
    for j = 1 to 2n − 1 do
        for k = j to 2n do
            Insert i in position j of NEWPATH
            Insert n + i in position k of NEWPATH
            if NEWPATH does not violate LIFO then
                if NEWPATH objective < PATH objective then
                    PATH := NEWPATH
                end if
            end if
        end for
    end for
end for
```

VITA

Devaraja Vignesh Radha Krishnan

Candidate for the Degree of

Doctor of Philosophy

Dissertation: EFFECTIVE ALGORITHMS FOR PICKUP AND DELIVERY PROBLEM WITH LOADING RESTRICTIONS AND HANDLING COSTS

Major Field: Industrial Engineering and Management

Biographical:

Education:

Completed the requirements for the Ph.D. degree in Industrial Engineering and Management at Oklahoma State University in December, 2020

Received the M.S. degree in Industrial Engineering and Management from Oklahoma State University in December, 2015

Received the B.E. degree in Aeronautical Engineering from Anna University, Chennai, Tamil Nadu, India, 2012

Experience:

Graduate Research/Teaching Assistant - School of Industrial Engineering and Management, Oklahoma State University (January 2014 - May 2020).

Logistics Engineering Intern - Transplace Inc., Frisco, Texas (June 2018 - Aug 2018, June 2019 - Aug 2019, Aug 2020 - Dec 2020).