12-2023

Exact Models, Heuristics, and Supervised Learning Approaches for Vehicle Routing Problems

Zefeng Lyu
zlyu2@vols.utk.edu

To the Graduate Council:

I am submitting herewith a dissertation written by Zefeng Lyu entitled "Exact Models, Heuristics, and Supervised Learning Approaches for Vehicle Routing Problems." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Industrial Engineering.

Andrew J. Yu, Major Professor

We have read this dissertation and recommend its acceptance:

Mingzhou Jin, James Ostrowski, Shuai Li

Accepted for the Council:

Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Exact Models, Heuristics, and Supervised Learning Approaches for Vehicle Routing Problems

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Zefeng Lyu

December 2023

DEDICATION

To my mom

Lianhua Pan

ACKNOWLEDGEMENTS

I would like to thank my parents for their unwavering support.

I express my deepest gratitude to my supervisor, Dr. Andrew J. Yu. He has always been kind and patient with me. His consistent encouragement, support, and guidance have greatly helped me build confidence in my research. I can hardly imagine completing my PhD program successfully and smoothly without his mentorship.

A special thanks to Dr. Jianbiao Pan; without him, I would not have had the opportunity to study abroad, let alone become the person I am today.

Many thanks to all my friends at the University of Tennessee. A special mention to Hang Wang and Gaoqiang Yang for the cherished friendship and the wonderful memories we have shared together.

Lastly, I would like to express my gratitude to my wife, Yujun Zhang, who has supported me, encouraged me, and been my companion during my most challenging moments. She is my motivation for a relentless pursuit of excellence.

ABSTRACT

This dissertation presents contributions to the field of vehicle routing problems by utilizing exact methods, heuristic approaches, and the integration of machine learning with traditional algorithms. The research is organized into three main chapters, each dedicated to a specific routing problem and a unique methodology. The first chapter addresses the Pickup and Delivery Problem with Transshipments and Time Windows, a variant that permits product transfers between vehicles to enhance logistics flexibility and reduce costs. To solve this problem, we propose an efficient mixed-integer linear programming model that has been shown to outperform existing ones. The second chapter discusses a practical workforce scheduling problem, formulated as a specific type of vehicle routing problem. The objective here is to efficiently assign consultants to various clients and plan their trips. This computational challenge is addressed by using a two-stage approach: the first stage employs a mathematical model, while the second stage refines the solution with a heuristic algorithm. In the final chapter, we explore methods that integrate machine learning with traditional approaches to address the Traveling Salesman Problem, a foundational routing challenge. Our goal is to utilize supervised learning to predict information that boosts the efficiency of existing algorithms. Taken together, these three chapters offer a comprehensive overview of methodologies for addressing vehicle routing problems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1. Introduction

Vehicle routing problems (VRPs) stand as fundamental challenges within the domains of logistics and transportation. These problems entail the allocation of resources in a manner that optimizes objectives, such as minimizing transportation costs, maximizing customer satisfaction under diverse constraints, such as maximum capacity, time windows, etc. Solving VRPs efficiently is crucial for enhancing operational efficiency, reducing costs, and ensuring timely and effective delivery of goods and services. As a result, extensive research efforts have been dedicated to developing innovative methods, algorithms, and techniques to address VRPs. This dissertation undertakes a comprehensive exploration of exact methods, heuristics, and supervised learning approaches to address vehicle routing problems and their variants.

In Chapter 2, we investigate the pickup and delivery problem with transshipments (PDP-T), where requests can be transferred between vehicles, as well as the pickup and delivery problem with time windows and transshipments (PDPTW-T). We conduct an in-depth analysis of two state-of-the-art models, identify limitations of the models, and propose refined formulations. Additionally, we introduce a new formulation that tackles both PDP-T and PDPTW-T. We address 340 generated PDP-T instances and 360 open-access PDPTW-T instances. Our computational results showcase the superior performance of our proposed model in terms of solution quality and computational time. Notably, our model significantly reduces the average computational time by 96% for PDP-T and 40% for PDPTW-T instances.

In Chapter 3, we focus on a variant of the routing problem that incorporates workforce allocation into considerations. Specifically, we address the simultaneous assignment of consultant supplies to client demands while determining optimal traveling routes for consultants. Our approach accounts for skill requirements, capacity limitations, fixed demand, and a maximum number of travel legs. Furthermore, we introduce priority matching, ensuring that clients are assigned to consultants with suitable priority levels. To tackle this computational challenge, we propose a decomposition algorithm and a MIP-

based neighborhood search algorithm. Additionally, we extend an existing Mixed-Integer Linear Programming (MILP) formulation to adapt it to the specific requirements of the proposed problem and compare our algorithms against it. By evaluating on 100 synthetic instances and 12 real-life instances, our computational analysis highlights the superior solution quality and reduced computational time offered by our proposed algorithms, particularly for large-scale and real-life scenarios.

In Chapter 4, we shift our focus to the traveling salesman problem (TSP), the most basic version of vehicle routing problems. Here, we propose a novel supervised learning approach that distinguishes itself from previous methods by leveraging local information rather than global information. By introducing the concept of "anchors," nodes that should be connected to their nearest neighbors in the optimal solution, our approach demonstrates excellent scalability and generalization capacity. Experimental results illustrate the effectiveness of our proposed model, successfully identifying 87% of the anchors with a precision exceeding 95% for both generated and TSPLIB instances. By integrating the predicted anchors into established methods such as the Miller-Tucker-Zemlin (MTZ) model and insertion algorithms, we achieve substantial improvements in solution quality, reducing the average gap.

Overall, this dissertation contributes advanced methodologies to tackle vehicle routing problems and their variants. Through improvements on exact methods, heuristics, and learning-based algorithms, we enhance solution quality, reduce computational time, and pave the way for scalable and efficient routing optimizations in real-world scenarios.

Chapter 2. An Exact Model for Solving the Pickup and Delivery Problems

This chapter is based on a paper published by Zefeng Lyu and Andrew Junfang Yu:

> Lyu, Z., & Yu, A. J. (2022). The pickup and delivery problem with transshipments: Critical review of two existing models and a new formulation. *European Journal of Operational Research*.

Zefeng Lyu contributed to methodology, original draft writing, software, validation, formal analysis, and visualization, while Dr. Andrew Junfang Yu contributed to conceptualization, methodology, and provided input during the writing and editing process, as well as providing supervision.

The pickup and delivery problem with transshipments (PDP-T) is generalized from the classical pickup and delivery problem (PDP) by allowing the transfer of requests between vehicles. After considering the time window constraints, the PDP-T is further generalized to the pickup and delivery problem with time windows and transshipments (PDPTW-T). In this paper, we review two state-of-the-art models for the PDP-T and PDPTW-T. We point out the possible issues existing in the models and provide our revisions. In addition, we develop a new mixed-integer linear programming formulation to solve the PDP-T and PDPTW-T. The performance of the proposed model is evaluated by solving 340 generated PDP-T instances and 360 open-access PDPTW-T instances. Computational results show that the proposed model outperforms the existing models in terms of solution quality and computing time. PTP-T instances with up to 25 requests and 2 transfer stations are solved to optimality by using the proposed model. As a comparison, the best-known benchmarks in literature are instances with 5 requests and 1 transfer station. In addition, the average computational time for solving PDP-T is reduced by 96%. For PDPTW-T instances, the average computing time is reduced by 40%.

2.1 Introduction

The pickup and delivery problem (PDP) aims to find the optimal routes for a fleet of capacitated vehicles to satisfy customer requests. Each request is associated with a pickup location and a delivery location. The vehicles depart from the origin depots, visit the pickup

locations to receive loads, deliver the loads to corresponding delivery locations, and return to the destination depots at the end. The PDP has been broadly studied. Relevant survey papers can be found at Koç, Laporte, & Tükenmez (2020) and Berbeglia, Cordeau, & Laporte (2010). With the development of e-commerce and information technology, people are exploring innovative methods to improve the transportation system. Allowing transshipment is one of the most promising attempts, which raises the pickup and delivery problem with transshipments (PDP-T).

In PDP-T, requests can be transferred from one vehicle to another. Specifically, requests can be dropped off at designated transfer stations and be stored there temporarily. Then, other vehicles can come to pick up the requests and complete the delivery. The PDP-T improves the efficiency of the transportation system by better utilizing the capacity and available time of the vehicles. In practice, the time for receiving and delivering requests is always restricted, either in a soft or hard manner, which gives rise to the pickup and delivery problem with time windows and transshipments (PDPTW-T).

Mitrović-Minić & Laporte (2006) show that allowing the transfer of requests is beneficial as the drivers can stay in their home areas. However, considering transshipment makes the problem much harder to be solved. For the pickup and delivery problem with time windows (PDPTW), Ropke, Cordeau, & Laporte (2007) report that the branch-and-cut algorithm can solve instances with up to 96 requests. However, the solvable scale is reduced to 7 requests for PDPTW-T (see Rais, Alvelos, & Carvalho, 2014). The challenges of solving the PDPTW-T are the expanded feasible regions and the synchronization requirements.

In this paper, we review two models for the PDP-T and PDPTW-T. We point out the possible issues existing in the models. We carefully discuss the causes of the issues and suggest our revisions. In addition, we presented a new mixed-integer linear programming (MILP) formulation to solve the problems. Computational results show that the proposed model is superior to the two existing models. To the best of our knowledge, this paper is the first that solves PDP-T instances with 25 requests and 2 transfer stations to optimality.

For PDPTW-T, the proposed model increases the solvable scale from 3 requests and 4 transfer stations to 5 requests and 4 transfer stations.

Contributions of this paper are three folds. First, we revise the possible issues existing in two state-of-the-art models for the PDP-T and PDPTW-T. Second, we present a new MILP formulation that is superior to the existing models. Third, we generate 340 new PDP-T instances and present the corresponding computational results as benchmarks for future research.

2.2 Literature review

Commonly used methods for solving the PDP-T and PDPTW-T can be divided into two categories, i.e., exact methods and metaheuristics. The exact algorithms for vehicle routing problems have been widely studied. We refer interested readers to the survey presented by Costa, Contardo, & Desaulniers (2019). In contrast, exact algorithms for PDP-T and PDPTW-T received limited attention.

Cortés, Matamala, & Contardo (2010) present an arc-based model for the PDP-T and show the benefits of allowing request transfers. They split every transfer state into two nodes, one for loading requests and the other for unloading requests. The model is solved by a branch-and-cut technique based on Benders decomposition. They show that their method can reduce the computing time by 90% compared with the branch-and-bound algorithm. As the computational results, they solve instances with 6 requests, 2 vehicles, and 1 transfer station to optimality.

Rais et al. (2014) present a MILP formulation to solve the PDP-T. The model can solve the PDPTW-T after adding additional time window constraints. In addition, several variants of the PDP-T are captured by adding necessary modifications. The computational results show that the MILP model solves PDP-T instances with 5 requests and PDPTW-T instances with 7 requests to optimality. In this paper, we review the MILP model and point out the possible issues. The causes of the issues are analyzed using an illustrative example.

Related revisions are also presented. In addition, we present a new MILP formulation that solves PDP-T instances with 25 requests and 2 transfer stations to optimality.

Sampaio et al. (2020) present a MILP formulation for the PDPTW-T in urban freight delivery systems. This formulation is based on the model proposed by Rais et al. (2014). The unique feature of their problem is that the drivers are occasional. Unlike regular drivers employed by the companies, the occasional drivers provide shorter shifts, e.g., 3 to 5 hours. In addition, the capacity constraints are relaxed by assuming that the vehicle capacity is sufficient. They present an adaptive large neighborhood search (ALNS) algorithm to solve large-scale instances. They show that the benefit of allowing transshipment can be significant, especially in settings where pickup and delivery locations are far apart, and the driver shifts are short. In this paper, we identify the possible issues existing in the MILP model, present our revisions, and propose a new model. The performance of the proposed model is compared with that of the existing one using open-access data.

A dynamic pickup and delivery problem under the urban environment is studied by Arslan, Agatz, Kroon, & Zuidwijk (2019). They solve the dynamic version of PDP using a rolling horizon framework and an exact method. The pickup and delivery problem with split loads and transshipments (PDPSL-T) is also closely related to the problems studied in this paper. In PDPSLT, loads can be split and delivered by multiple vehicles. Unlike the PDPTW-T, the PDPSL-T does not consider the time window constraints. Wolfinger & Salazar-González (2021) present a branch-and-cut algorithm for PDPSL-T. The branch-and-cut algorithm solves instances with 8 requests to optimality within 24 hours. They state that the instances solved are the largest both for the PDP-T and PDPSL-T. In this paper, we solve PDP-T instances with 25 requests to optimality within 1 hour.

Exact methods can find optimal solutions but only work for small-scale instances. Metaheuristics are practical methods in solving large-scale instances. Ropke & Pisinger (2006) present an ALNS algorithm to solve the PDPTW. Since then, the ALNS algorithm is widely applied to solve the PDP-T and PDPTW-T (see for examples Qu & Bard, 2012; Masson, Lehuédé, & Péton, 2013; Sampaio, Savelsbergh, Veelenturf, & Van Woensel, 2020; Wolfinger, 2021; and Voigt & Kuhn, 2021). In addition to the ALNS algorithm,

Cortes & Suzuki (2020) present a simulated annealing algorithm. Danloup, Allaoui, & Goncalves (2018) compares a large neighborhood search algorithm with a genetic algorithm in solving the PDP-T.

The remainder of the paper is constructed as follows. In Section 3.3, we review the PDP-T model presented by Rais et al. (2014). We point out the possible issues existing in the model, i.e., the subtour elimination requirement and the synchronization requirement are not satisfied in specific instances. We discuss the causes of the issues and present our revisions. In Section 3.4, we review the PDPTW-T model presented by Sampaio et al. (2020). This model is modified from the model of Rais et al. (2014). Similar to the previous section, we point out the possible issues for this model and present revisions. In Section 3.5, we propose a new MILP formulation for solving the PDP-T and PDPTW-T. In Section 3.6, the performance of the proposed MILP is evaluated by solving 340 generated PDP-T instances and 360 open-access PDPTW-T instances. Computational results show that the proposed model outperforms the existing models. Finally, conclusions are made in the last section.

2.3 Formulation for PDP-T

This section discusses the MILP formulation proposed by Rais et al. (2014). The notations are defined as follows. Graph $G = (N, A)$ is a directed graph, where N is the set of nodes and $A = \{(i, j) | i \in N, j \in N, i \neq j\}$ is the set of arcs. There are five types of nodes, i.e., the origin depots O, the destination depots O', the pickup locations P, the delivery locations D, and the transfer stations T. The vehicles K are heterogeneous in terms of the origin depots, destination depots, and driving cost per unit distance traveled. $o(k)$ and $o'(k)$ denote the original depot and destination depot of vehicle k, respectively. R denotes the set of requests. Each request $r \in R$ is associated with a pickup location $p(r)$ and a delivery location $d(r)$. c_{ij}^{k} represents the travel cost associated with arc (i, j) and vehicle k. In order to restrict the maximum load, u_k denotes the capacity of vehicle k while q_r denotes the quantality of request r. Binary variables x_{ij}^{k}, y_{ij}^{kr} and z_{ij}^{k} are the decision variables. x_{ij}^{k}=1

if vehicle k travels through arc (i,j). $x_{ij}^k = 0$ otherwise. We can obtain the vehicle flows from the results of x_{ij}^k. Similarly, $y_{ij}^{rk} = 1$ if request r is transported by vehicle k through arc (i,j) and $y_{ij}^{rk} = 0$ otherwise. The request flows can be obtained from these variables. z_{ij}^k is used to determine the order in which the nodes are visited. Specifically, $z_{ij}^k = 1$ if node i precedes node j for vehicle k and $z_{ij}^k = 0$ otherwise. With the above notations, the MILP formulation is listed as follows.

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

s. t.
$$\sum_{(i,j)\in A} x_{ij}^k \le 1 \;\; \forall k \in K, i = o(k) \tag{2.3.1}$$

$$\sum_{(i,j)\in A} x_{ij}^k = \sum_{(j,l)\in A} x_{jl}^k \;\; \forall k \in K, i = o(k), l = o'(k) \tag{2.3.2}$$

$$\sum_{(i,j)\in A} x_{ij}^k - \sum_{(j,i)\in A} x_{ji}^k = 0 \;\; \forall k \in K, \forall i \in N\backslash\{o(k), o'(k)\} \tag{2.3.3}$$

$$\sum_{k\in K}\sum_{(i,j)\in A} y_{ij}^{kr} = 1 \;\; \forall r \in R, i = p(r) \tag{2.3.4}$$

$$\sum_{k\in K}\sum_{(j,i)\in A} y_{ji}^{kr} = 1 \;\; \forall r \in R, i = d(r) \tag{2.3.5}$$

$$\sum_{k\in K}\sum_{(i,j)\in A} y_{ij}^{kr} - \sum_{k\in K}\sum_{(j,i)\in A} y_{ji}^{kr} = 0 \;\; \forall r \in R, \forall i \in T \tag{2.3.6}$$

$$\sum_{(i,j)\in A} y_{ij}^{kr} - \sum_{(j,i)\in A} y_{ji}^{kr} = 0 \;\; \forall k \in K, \forall r \in R, \forall i \in N\backslash T \tag{2.3.7}$$

$$y_{ij}^{kr} \le x_{ij}^k \;\; \forall (i,j) \in A, \forall k \in K, \forall r \in R \tag{2.3.8}$$

$$\sum_{r\in R} q_r y_{ij}^{kr} \le u_k x_{ij}^k \;\; \forall (i,j) \in A, \forall k \in K \tag{2.3.9}$$

$$x_{ij}^k \le z_{ij}^k \;\; \forall i,j \in N, \forall k \in K, i \ne o(k), j \ne o'(k) \tag{2.3.10}$$

$$z_{ij}^k + z_{ji}^k = 1 \; \forall i,j \in N, \forall k \in K, i \ne o(k), j \ne o'(k) \tag{2.3.11}$$

$$z_{ij}^{k} + z_{jl}^{k} + z_{li}^{k} \leq 2 \ \ \forall i, j, l \in N, \forall k \in K, i, j \neq o(k), l \neq o'(k) \tag{2.3.12}$$

$$x_{ij}^{k} \in \{0, 1\} \ \ \forall (i, j) \in A, \forall k \in K \tag{2.3.13}$$

$$y_{ij}^{kr} \in \{0,1\} \ \ \forall (i, j) \in A, \forall k \in K, \forall r \in R \tag{2.3.14}$$

$$z_{ij}^{k} \in \{0,1\} \ \ \forall i, j \in N, \forall k \in K \tag{2.3.15}$$

Explanations for the MILP formulation can be found in Rais et al. (2014). We briefly review these constraints here. Constraints (2.3.1) to (2.3.3) maintain the vehicle flows while constraints (2.3.4) to (2.3.7) maintain the request flows. Constraints (2.3.8) link the vehicle flows and the request flows. Constraints (2.3.9) ensure that the loading capacities are not exceeded. Constraints (2.3.10) to (2.3.12) are used to eliminate subtours. Constraints (2.3.13) to (2.3.15) restrict the variables $x_{ij}^{k}, y_{ij}^{kr}, z_{ijk}$ to be binary.

We find it necessary to make some revisions to this model. The first revision is that the origin depots and destination depots of the requests should be excluded from constraints (2.3.7). Otherwise, the request flow is restricted to returning to the pickup location after passing through the delivery location, which makes the problem infeasible. Constraints (2.3.7) are revised as constraints (2.3.16) to resolve this issue.

$$\sum_{(i,j)\in A} y_{ij}^{kr} - \sum_{(j,i)\in A} y_{ji}^{kr} = 0 \ \ \forall k \in K, \forall r \in R, \forall i \in N \backslash \{T \cup \{p(r), d(r)\}\} \tag{2.3.16}$$

The second revision is related to subtour elimination. Constraints (2.3.10) to constraints (2.3.12) eliminate the subtours by determining the precedence that the nodes are visited. The origin depots and destination depots are excluded from the constraints. This exclusion is necessary for the traveling salesman problem (TSP) because the origin depot and destination depot coincide with each other.

However, the depots should not be excluded for PDP-T. Otherwise, the model would obtain unreasonable solutions as the vehicles can return to the original depots. An illustrative example is shown in Figure 2.3.1.

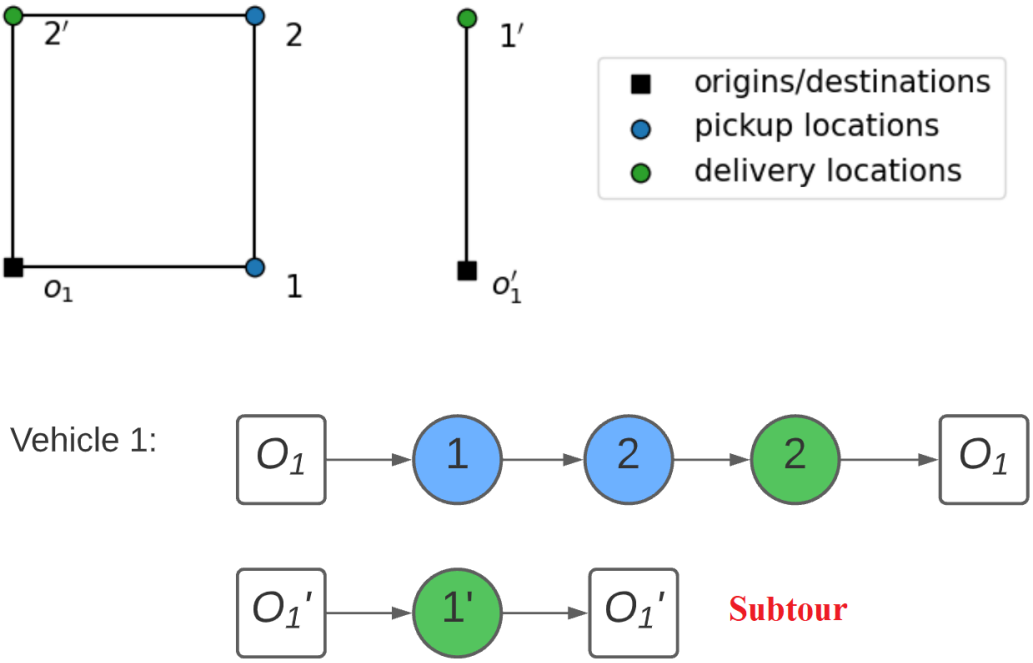

Figure 2.3.1: Illustration of the solution obtained by solving “example 1”

This figure shows a solution obtained by solving "example 1" using the revised model without adding constraints (2.3.17) to (2.3.19). The geographical locations of the nodes and the routes are shown in the top. The vehicle flows are shown in the bottom. This solution is infeasible because of the subtour. As is shown in the figure, there are subtours in the solution because the depots O and O' are excluded from the subtour elimination constraints. To address this issue, we modify the domain of constraints (2.3.10) to (2.3.12). The revised constraints are shown in constraints (2.3.17) to (2.3.19).

$$x_{ij}^k \leq z_{ij}^k \quad \forall (i,j) \in A, \forall k \in K \tag{2.3.17}$$

$$z_{ij}^k + z_{ji}^k = 1 \; \forall (i,j) \in A, \forall k \in K \tag{2.3.18}$$

$$z_{ij}^k + z_{jl}^k + z_{li}^k \leq 2 \;\; \forall i,j,l \in N, \forall k \in K, (i,j),(j,l),(l,i) \in A \tag{2.3.19}$$

The third revision is associated with the synchronization requirement. Synchronization means that if a request is transferred between two vehicles, the vehicle that drops off the request should arrive at the transfer station before the vehicle that comes to pick up the request. This issue is easily overlooked because the model obtains correct solutions for most of the PDP-T instances. We will explain in detail how ignoring the synchronizing constraints can lead to infeasible solutions. An example is shown in Figure 2.3.2.

This is an illustration of the solution obtained by solving "example 2" using the revised model without adding equations (2.3.20) to (2.3.24). The solid line represents the route of vehicle 1 while the dashed line represents the route of vehicle 2. This solution is infeasible because the two transshipments conflict with each other. As is shows, request 1 is transferred from vehicle 1 to vehicle 2 at transfer station t_2 while request 2 is transferred from vehicle 2 to vehicle 1 at transfer station t_1. However, these two transshipments conflict with each other. On the one hand, if requests 1 is transferred, vehicle 1 must arrive t_1 earlier than vehicle 2. On the other hand, if request 2 is transferred, vehicle 2 must arrive t_1 earlier than vehicle 1, leading to the conflict. To solve the synchronization issue, we add new variables and constraints as shown in equations (2.3.20) to (2.3.24). e_i^k are positive

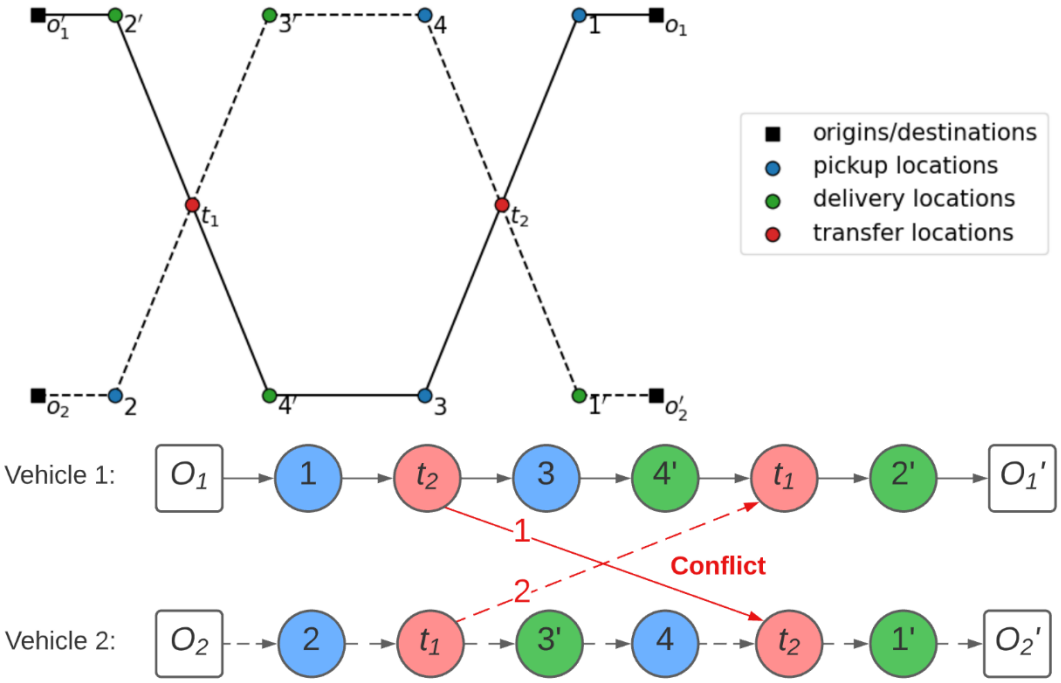

Figure 2.3.2: Illustration of the solution obtained by solving “example 2”

numbers that represent the sequence that node i is visited by vehicle k. Variables $s_{tr}^{k_1 k_2}$ indicate whether request r is transferred from vehicle k_1 to vehicle k_2 at transfer station t.

$$e_i^k + 1 - e_j^k \leq M\left(1 - x_{ij}^k\right) \forall (i,j) \in A, \forall k \in K \tag{2.3.20}$$

$$\sum_{(j,t)\in A} y_{jt}^{k_1 r} + \sum_{(t,j)\in A} y_{tj}^{k_2 r} \leq s_{tr}^{k_1 k_2} + 1 \ \ \forall r \in R, \forall t \in T, \forall k_1, k_2 \in K, k_1 \neq k_2 \tag{2.3.21}$$

$$e_t^{k_1} - e_t^{k_2} \leq M\left(1 - s_{tr}^{k_1 k_2}\right) \forall r \in R, \forall t \in T, \forall k_1, k_2 \in K, k_1 \neq k_2 \tag{2.3.22}$$

$$e_i^k \geq 0 \ \ \forall i \in N, \forall k \in K \tag{2.3.23}$$

$$s_{tr}^{k_1 k_2} \in \{0,1\} \ \ \forall t \in T, \forall r \in R, \forall k_1, k_2 \in K, k_1 \neq k_2 \tag{2.3.24}$$

Constraints (2.3.20) determine the sequence of the nodes visited by the vehicles. M is a sufficiently large number. In the experiments, we set the value to be the number of nodes in the Graph G. Constraints (2.3.21) indicate whether a request is transferred between two vehicles in the transfer stations. Constraints (2.3.22) restrict that if a request r is transferred from k_1 to k_2 at transfer station t, the vehicle k_1 should visit t before k_2. The decision variables are defined in equations (2.3.23) and (2.3.24).

$$\sum_{(i,j)\in A} x_{ij}^k = 1 \ \ \forall k \in K, i = o(k) \tag{2.3.25}$$

In addition to the three revisions above, we also suggest modifying constraints (2.3.1) to constraints (2.3.25). This modification is not mandatory but is related to how the PDP-T is defined. In the original problem, if a vehicle is not used, its travel distance between the origin depot and destination depot will not be counted into the objective function. This setting causes an issue that the vehicles with long distances between the origin depots and destination depots are not preferred. In other words, the vehicles with long origin-destination pairs may not be used even if there are requests on the way of their original trips. An illustrative example is shown in Figure 2.3.3.

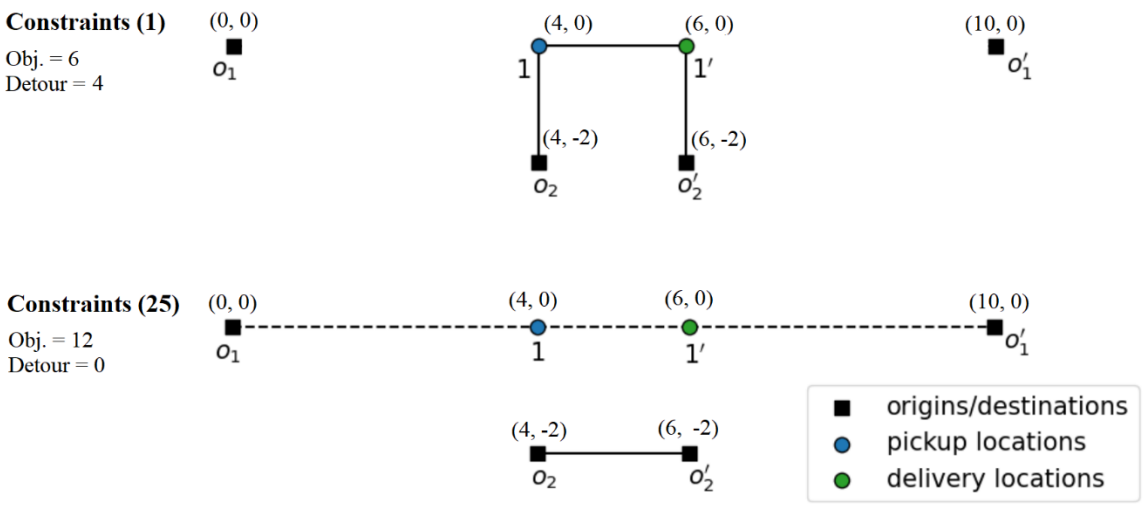

Figure 2.3.3: Illustration of the solution obtained by solving “example 3”

This figure depicts an illustration of comparing the solutions obtained by solving "example 3" using the revised model and that without replacing constraints (2.3.1) by constraints (2.3.25). The dashed line represents the route of vehicle 1 while the solid line represents the route of vehicle 2. The coordinates are noted above the nodes. There are two vehicles and one request in the example. The origin and destination of vehicle k_1 are o_1 and $o_1{}'$ respectively. Similarly, the origin and destination of vehicle k_2 are o_2 and $o_2{}'$. The solution of using constraints (2.3.1) is shown at the top of the figure. In this solution, the request is picked up and delivered by k_2. The objective function value is 6 but there is a detour of 4. Vehicle k_1 is not preferred because the distance between o_1 and $o_1{}'$ is too far. If vehicle k_1 is used, the objective function value will increase to 10, although there is no detour for k_1 to pick up and deliver the request.

If constraints (2.3.25) are used, the request would be picked up and delivered by k_1. As shown at the bottom of the figure, there is no detour in the new solution. In other words, the revised model is to minimize the detour taken by the vehicles rather than the actual driving distance. Note that our objective function value is not exactly the detour. In fact, it is equal to the detour plus a constant, which is the total distance between the origins and destinations of the vehicles. To retrieve the detour, we need to subtract this constant. The revised model for solving PDP-T is shown as follows,

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

s. t. (2.3.2) to (2.3.6), (2.3.8), (2.3.9), (2.3.13) to (2.3.25).

2.4 Formulation for PDPTW-T

This section discusses the MILP formulation presented by Sampaio et al. (2020) for solving PDPTW-T. The model is generated from the one proposed by Rais et al. (2014), which is shown in Section 2.3. Sampaio et al. (2020) focus on PDPTW-T for urban environments where crowd-shipping is considered. Specifically, the drivers are not employed by the

companies but occasionally provide services. The occasional drivers tend to provide short service times, e.g., three to five hours. Sampaio et al. (2020) make some restrictions and assumptions for the PDPTW-T. They require that the vehicles start and end their shifts at the same depots. They assume that the capacities of the vehicles are sufficient, so the capacity constraints can be released. It is also assumed that each vehicle can visit the same transfer station at most once.

The notations used in this section are the same as those in Section 2.3. Additional notations are defined to handle the time window constraints. c_{ij} represents the travel cost associated with arc (i,j). τ_{ij} represents the traveling time between node i and node j. E_i denotes the earliest time that the requests can be picked up at the location i. By contrast, L_i denotes the latest time that the requests can be delivered to i. Variables $s_{tr}^{k_1k_2} = 1$ if request r is transferred from vehicle k_1 to vehicle k_2 at transfer station t. $s_{tr}^{k_1k_2} = 0$ otherwise. a_i^k and b_i^k represent the arrival time and departure time for vehicle k at location i, respectively. The MILP formulation is listed as follows.

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

s. t. (2.3.1), (2.3.4) - (2.3.6), (2.3.8), (2.3.13), (2.3.14), (2.3.16)

$$\sum_{(i,j)\in A} x_{ij}^k = \sum_{(j,i)\in A} x_{ji}^k \quad \forall k \in K, i = o(k) \tag{2.4.1}$$

$$\sum_{(i,j)\in A} x_{ij}^k - \sum_{(j,i)\in A} x_{ji}^k = 0 \quad \forall k \in K, \forall i \in P \cup D \cup T \tag{2.4.2}$$

$$b_i^k + \tau_{ij} - a_j^k \le M\left(1 - x_{ij}^k\right) \quad \forall (i,j) \in A, \forall k \in K \tag{2.4.3}$$

$$b_{i^+}^k \ge E_{i^+}, a_{i^-}^k \le L_{i^-} \quad \forall k \in K, \forall r \in R, i^+ = p(r), i^- = d(r) \tag{2.4.4}$$

$$\sum_{(j,t)\in A} y_{jt}^{k_1r} + \sum_{(t,j)\in A} y_{tj}^{k_2r} \le s_{tr}^{k_1k_2} + 1 \quad \forall r \in R, t \in T, k_1, k_2 \in K \tag{2.4.5}$$

$$a_t^{k_1} - b_t^{k_2} \le M\left(1 - s_{tr}^{k_1k_2}\right) \quad \forall r \in R, t \in T, k_1, k_2 \in K \tag{2.4.6}$$

$$s_{tr}^{k_1k_2} \in \{0,1\} \quad \forall t \in T, \forall r \in R, \forall k_1, k_2 \in K \tag{2.4.7}$$

$$a_i^k, b_i^k \geq 0 \ \ \forall i \in N, \forall k \in K \tag{2.4.8}$$

Detailed descriptions of the formulation can be found in Sampaio et al. (2020). Here we briefly describe the differences between this model and the model described in Section 2.3. Constraints (2.4.1) and (2.4.2) are used to replace constraints (2.3.2) and (2.3.3) because Sampaio et al. (2020) assume that the starting and ending points of the vehicles coincide. Constraints (2.4.3) and (2.4.4) are added as time window constraints. Constraints (2.4.5) are almost the same as constraints (2.3.21). The only difference between them is that $k_1 = k_2$ is allowed in constraints (2.4.5). Although it does not have a significant effect on the solution quality, it is thought better to prohibit it just like in constraints (2.3.21). Constraints (2.4.6) achieve the same function as constraints (2.3.22). They make sure that if a transfer occurs, the vehicle dropping the request should arrive at the transfer location before the vehicle picking up the request. Constraints (2.4.7) and (2.4.8) define the decision variables. Next, we discuss the possible issues in the model and propose our revisions.

The first revision is related to vehicle flow conservation. The home depots are excluded from constraints (2.4.2). However, this exclusion is too broad. We should only exclude the depot of the corresponding vehicle rather than the whole set of depots. Otherwise, the vehicles can depart from a depot that does not belong to it.

A counterexample is shown in Figure 2.4.1. In this example, the solution is obtained by solving "example 4" using the revised model without replacing revising constraints (2.4.1) and (2.4.2). This solution is infeasible because the vehicle k_3 does not start and end from its own depot. Instead, k_3 utilizes the depots that belong to k_1 and k_2. To solve this issue, we suggest replacing the constraints (2.4.1) and (2.4.2) with the following constraints.

$$\sum_{(i,j)\in A} x_{ij}^k - \sum_{(j,i)\in A} x_{ji}^k = 0 \ \ \forall k \in K, \forall i \in N \tag{2.4.9}$$

The second revision is related to the time window constraints. By analyzing the open-access datasets in Sampaio, Savelsbergh, Veelenturf, & Van Woensel (2020b), we find that

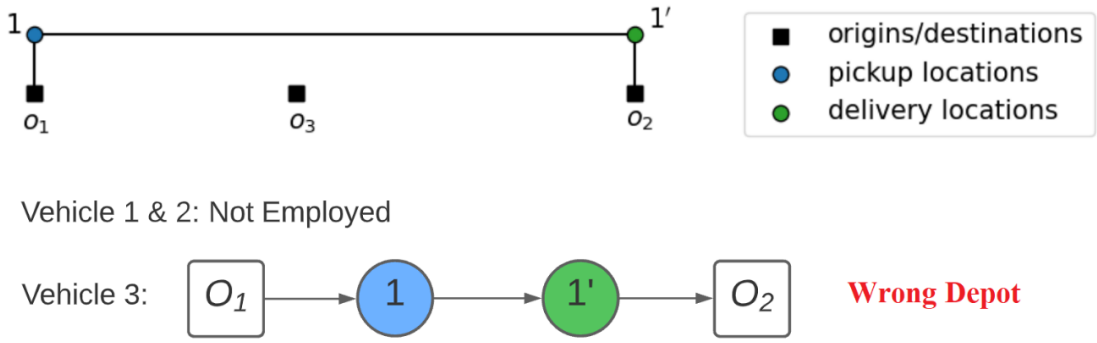

Figure 2.4.1: Illustration of the solution obtained by solving “example 4”

the current constraints are not sufficient to restrict the time windows. Constraints (2.4.4) only restrict one side for the time windows, i.e., the earliest departure time for the pickup locations and the latest arriving time for the delivery nodes. In the dataset, E_{i^+} is equal to zero for all nodes and L_{i^-} is equal to 180, 240, or 300 depending on the vehicle shift length. In contrast, E_{i^-} and L_{i^+} have different variables and thus are more important. However, E_{i^-} and L_{i^+} are not used in the model. It is thought necessary to restrict both sides of the time windows. Otherwise, the time of arrival and departure may not be restricted as expected. In addition, it is thought necessary to add new constraints to maintain time conservation. For each vehicle, the time of reaching a node must be earlier than the time of leaving that node. Without these constraints, the obtained optimal solution would be unreasonable.

An example is shown in Figure 2.4.2. In this example, the solution is obtained by solving "example5" using the revised model without revising the constraints. This solution is infeasible because the time window constraints fail to eliminate self-loops. To solve this issue, we replace constraints (2.4.4) with constraints (2.4.10), and add constraints (2.4.11).

$$E_i \leq b_i^k \leq L_i, E_i \leq a_i^k \leq L_i \; \forall k \in K, \forall i \in N \tag{2.4.10}$$

$$b_i^k \geq a_i^k \quad \forall i \in N, \forall k \in K, i \neq o(k) \tag{2.4.11}$$

The last revision is related to the subtour elimination. The subtours of the vehicle flow are eliminated by determining the order in which the nodes are visited. This method is similar to the subtour elimination constraints proposed by Miller, Zemlin, & Tucker (1960). However, the subtours of the request flow are not eliminated as expected. An illustrative example is shown in Figure 2.4.3.

The solution is obtained by solving "example6". The solid line represents the route of vehicle 1 while the dashed line represents the route of vehicle 2. This solution is infeasible because there are subtours for request 2. The instance includes four requests, two vehicles, and one transfer station. Note that we have made the first two revisions and add the new constraints. However, the solution is still infeasible because the coupling requirement is

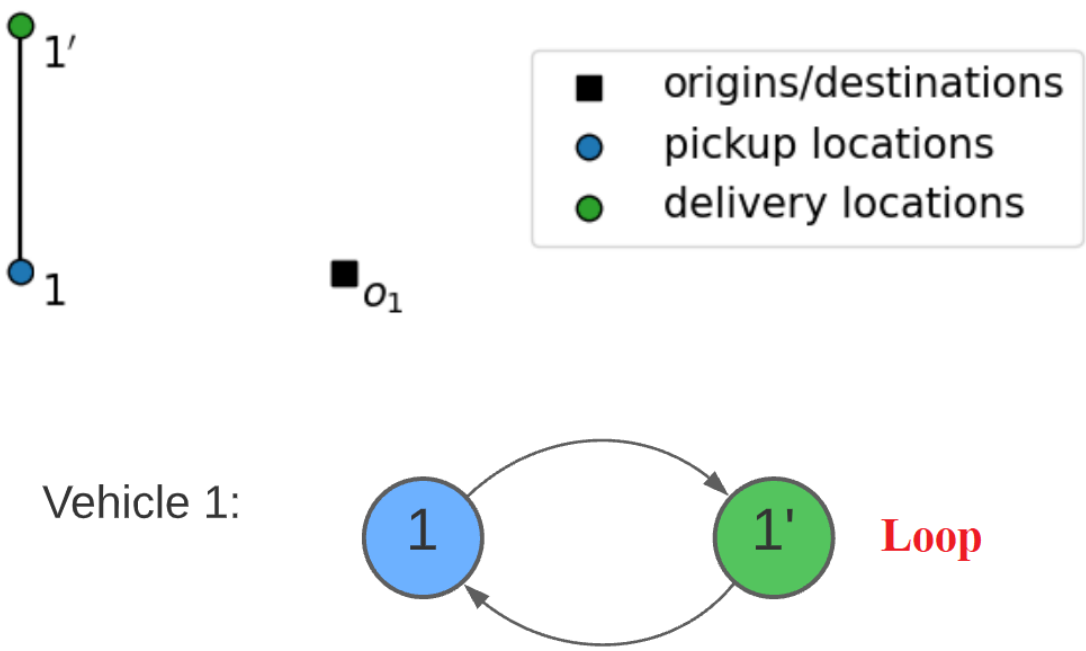

Figure 2.4.2: Illustration of the solution obtained by solving “example 5”

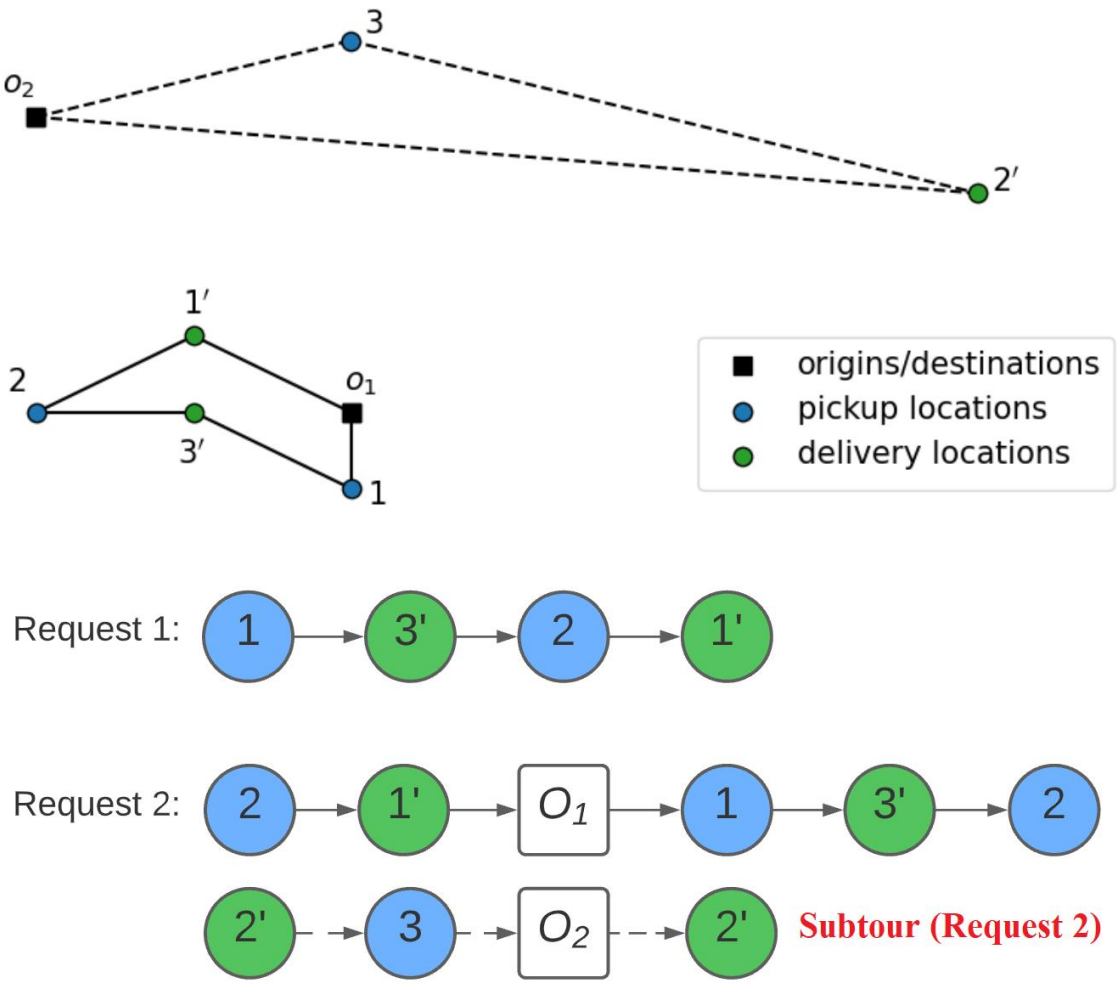

Figure 2.4.3: Illustration of the solution obtained by solving “example 6”

not respected. Request 2 is not picked up and delivered by the same vehicle. We find that there are subtours for request 2. The flow of request 2 is $2 \to 1' \to o_1 \to 1 \to 3' \to 2$ and $2' \to 3 \to o_2 \to 2'$. This request flow satisfies the model but contains subtours. There are several ways to solve this issue. For example, we can explicitly add subtour elimination constraints for the request flows. Here, we provide an alternative method. By analyzing the solutions, we find that all the subtours contain depots in the request flow. Therefore, we can eliminate the subtours by simply prohibiting depots in the request flow. The constraints to be added are (2.4.12), (2.4.13), and (2.4.14).

$$y_{ij}^{kr} = 0 \quad \forall k \in K, \forall r \in R, \forall (i,j) \in A, i \in O \tag{2.4.12}$$

$$y_{ij}^{kr} = 0 \quad \forall k \in K, \forall r \in R, \forall (i,j) \in A, j \in O \tag{2.4.13}$$

$$y_{ij}^{kr} = 0 \quad \forall k \in K, \forall r \in R, \forall (i,j) \in A, j = p(r) \tag{2.4.14}$$

The revised model for solving PDPTW-T is listed as follows,

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

s. t. (2.3.1), (2.3.4) - (2.3.6), (2.3.8), (2.3.13), (2.3.14), (2.3.16), (2.4.3), (2.4.5) - (2.4.14).

2.5 New MILP formulation

We propose a new MILP formulation to solve the PDP-T and PDPTW-T. We generate the model based on the one presented by Rais et al. (2014). The main improvement is that we add several redundant constraints for the model. These constraints are redundant in terms of searching for the optimal solution but can strengthen the LP relaxation. As a result, the efficiency of the model is significantly improved. These redundant constraints can also be designed as valid cuts for a branch-and-cut algorithm. In this paper, we directly add these

constraints to the model as the number of them is moderate. It may not be worth checking the solutions during the optimization process and adding the valid cuts as needed.

We have also tried some other methods to reduce the computing time of solving the model. For instance, one can reduce the problem scale by eliminating unnecessary arcs before feeding them into the model (see for example Cordeau, 2006). One can also reduce the number of variables by aggregating the time variables (see for example Cordeau, 2006). Similarly, the number of variables can be reduced by only keeping the arriving time for the nodes except the transfer stations. In addition, the constraints for time conservation can be lifted (see for example Ropke et al., 2007). However, based on our experiments, these techniques do not reduce the computing time significantly. Based on our preliminary experiments, an efficient way to reduce the computing time is to add appropriate redundant constraints to the model, as shown in the following MILP formulation.

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

s. t. (2.3.4) - (2.3.6), (2.3.8), (2.3.9), (2.3.13), (2.3.14), (2.3.16), (2.3.21), (2.3.24), (2.3.25), (2.4.2), (2.4.8)

$$\sum_{(j,i)\in A} x_{ji}^k = 0 \quad \forall k \in K, i = o(k) \tag{2.5.1}$$

$$\sum_{(i,j)\in A} x_{ij}^k = 0 \quad \forall k \in K, \forall i \in O \cup O', i \neq o(k) \tag{2.5.2}$$

$$\sum_{(j,i)\in A} x_{ji}^k = 1 \quad \forall k \in K, i = o'(k) \tag{2.5.3}$$

$$\sum_{(i,j)\in A} x_{ij}^k = 0 \quad \forall k \in K, i = o'(k) \tag{2.5.4}$$

$$\sum_{(i,j)\in A} x_{ij}^k \leq 1 \quad \forall k \in K, \forall i \in T \tag{2.5.5}$$

$$\sum_{(i,j)\in A}\sum_{k\in K} x_{ij}^k = 1 \; \forall i \in P \cup D \tag{2.5.6}$$

$$\sum_{(i,j)\in A}\sum_{k\in K} y_{ij}^{kr} = 0 \ \ \forall r \in R, j = p(r) \tag{2.5.7}$$

$$\sum_{(i,j)\in A} y_{ij}^{kr} = 0 \ \ \forall r \in R, \forall k \in K, \forall i \in O \cup O', i \neq o(k), i \neq o'(k) \tag{2.5.8}$$

$$a_t^{k_1} - b_t^{k_2} \leq M\left(1 - s_{tr}^{k_1 k_2}\right) \ \ \forall r \in R, t \in T, k_1, k_2 \in K, k_1 \neq k_2 \tag{2.5.9}$$

$$b_i^k + \tau_{ij}^k - a_j^k \leq M\left(1 - x_{ij}^k\right) \ \ \forall (i,j) \in A, \forall k \in K \tag{2.5.10}$$

$$a_i^k \geq E_i, b_i^k \leq L_i \ \ \forall i \in N, \forall k \in K \tag{2.5.11}$$

$$a_i^k \leq b_i^k \ \ \forall i \in N, \forall k \in K \tag{2.5.12}$$

The objective function minimizes the traveling costs of the vehicles. Constraints (2.5.1) restrict that the vehicles cannot go back to the origin depots. Departure from a position other than the origin depots is prohibited by constraints (2.5.2). Constraints (2.5.3) ensure that the vehicles end their routes at the destination depots. Constraints (2.5.4) make sure that the vehicles do not leave the destination depots. Constraints (2.5.5) restrict that each vehicle visits the same transfer station at most once. Constraints (2.5.6) limit that the pickup locations and delivery locations are visited only once. Constraints (2.5.7) restrict that the request flows should not contain arcs that head to the pickup locations. Constraints (2.5.8) make sure that the request flows should not include the origin depots or destination depots.Constraints (2.5.9) are used to maintain the synchronization requirements. If a request is transferred between two vehicles, the vehicle that drops off the request must arrive at the transfer node before the vehicle that comes to receive the request. We set $M \geq L_t - E_t$ to maintain the validity of constraints. Constraints (2.5.10) to (2.5.12) guarantee that the requests are picked up and delivered in the given time windows. The validity of constraints (2.5.10) is ensured by setting $M \geq \max\{0, L_i + \tau_{ij}^k - E_j\}$.

2.6 Computational Experiments

In this section, we evaluate the performance of the proposed formulation by comparing it with the two modified formulations shown in Section 2.3 and Section 2.4. The formulations

are coded in Python and tested using Gurobi 9.0.1 as the exact solver. The experiments are conducted on an Intel (R) Xeon (R) E-2274G CPU (4.00 GHz) machine with 32 GB of RAM, under Windows 10. We generate 340 instances to test the proposed model in solving PDP-T. To test the model in solving the PDPTW-T, we use the instances provided by Sampaio, Savelsbergh, Veelenturf, and Woensel (2020). The raw instances are available in https://data.mendeley.com/datasets/pywzcgyzrv/2. Since the scale of these instances is too large for the exact methods, we generate smaller instances by keeping only part of the requests. The coordinates of the vehicles and the transfer stations remain unchanged. The time limitation is set to 3600 seconds for the experiments. All instances tested in this paper and the computational results can be found at Mendeley Data by Lyu & Yu (2022) with the following link: https://data.mendeley.com/datasets/w925jygjct/4.

2.6.1 Instance generation

Rais et al. (2014) generate instances based on the datasets of Li & Lim (2001). As their instances are not open access, we generate new instances to evaluate the performance of the proposed model. We generate 24 groups of PDP-T instances with a different setting of requests, vehicles, and transfer stations. Each group contains 10 instances. The nodes (i.e., pickup nodes, delivery nodes, origin depots, destination depots, and transfer stations) are randomly located on a 100×100 Euclidean grid. The traveling cost between two nodes is the Euclidean distance. Each pair of pickup node and delivery node are associated with a positive load q and a negative load $-q$, respectively. The value of q is generated using a discrete uniform distribution within the interval of [1, 100]. The vehicles have different origin depots and destination depots but their capacity is homogenous, which is set to 100.

To test the proposed formulation in solving the PDPTW-T, we use the instances provided by Sampaio et al. (2020). These instances are originally solved by an adaptive large neighborhood search algorithm. Since the scale of these instances is too large for exact algorithms, we generate smaller instances by picking the first several requests (i.e., 3, 4, and 5). The depots and transfer stations are kept the same as the original instances. Unlike in Sampaio et al. (2020), we cannot set the number of vehicles to infinite because

the number of vehicles should be determined for the models. We set the number of vehicles to be 4 as we find it enough to serve the requests. In addition, Sampaio et al. (2020) do not consider the capacity constraints. We can set the vehicle capacity to be sufficiently large to release the capacity constraints. Thus, we set the vehicle capacity to 99 in our experiments. Both the transfer geometry MD-4T and MD-5T are used to generate instances with 3 requests. Since our focus is not to study the impact of different geometries, we only use the MD-4T setting to generate instances with 4 and 5 requests.

2.6.2 Results of solving the PDP-T instances

Table 2.6.1 presents a performance comparison between the proposed model and the pre-existing RAC model in tackling the generated PDP-T instances. The term RAC Model pertains to the model introduced by Rais et al. (2014) with the necessary revisions described in Section 2.3. To facilitate comprehension, the definitions of the columns in this table are outlined below.

The first column lists the names of the instance groups. The groups are named in the following manner: the value after R denotes the number of requests, the value after K denotes the number of vehicles, and the value after T denotes the number of transfer stations. Column #opt. reports the number of instances that is solved to optimality. Column #lim. reports the number of instances that feasible solutions are found. Column #no. reports the number of instances that no feasible solutions are found within the time limit. We omit column #no. for the proposed model because all the values under that column are zero. The average objective function values, average percentage gaps, and average computing times are listed under columns obj., column gap(%), and column t(s), respectively.

The gaps are directly obtained from Gurobi. The numbers are averaged according to the instances in the group. The gaps capture the percentage difference between the best solution found and the best lower bound. The time limitation of running the solver is 3600 seconds. Note that the instances that no solutions are found are excluded from calculating the average objective function value and the average gap. That is the reason why the average gap of the RAC model for group R12K3T3 is zero.

Table 2.6.1: Results of the models in solving the PDP-T instances

	RAC Model						Proposed Model				
Instance	#opt.	#lim.	#no.	obj.	gap (%)	*t*(s)	#opt.	#lim.	obj.	gap (%)	*t*(s)
R5K2T1	10	0	0	442.4	0.0	86.9	10	0	442.4	0.0	0.1
R5K2T2	10	0	0	415.5	0.0	91.0	10	0	415.5	0.0	0.2
R5K3T3	10	0	0	421.4	0.0	436.3	10	0	421.4	0.0	0.5
R7K2T1	6	4	0	529.3	6.1	1779.0	10	0	525.0	0.0	2.9
R7K2T2	7	3	0	574.2	2.9	1872.5	10	0	572.7	0.0	0.9
R7K3T3	2	7	1	575.7	10.2	2972.0	10	0	560.4	0.0	2.0
R10K2T1	0	4	6	673.4	11.1	3600.2	10	0	717.3	0.0	11.3
R10K2T2	0	4	6	705.4	16.8	3600.1	10	0	703.0	0.0	21.5
R10K3T3	0	1	9	665.4	5.7	3600.1	10	0	686.7	0.0	8.6
R12K2T1	0	1	9	703.3	18.1	3600.0	10	0	729.1	0.0	43.7
R12K2T2	0	0	10	-	-	3600.0	10	0	751.6	0.0	72.2
R12K3T3	1	0	9	778.3	0.0	3445.4	10	0	812.4	0.0	218.1
R15K2T1	0	0	10	-	-	3600.0	10	0	933.0	0.0	362.1
R15K2T2	0	0	10	-	-	3600.0	10	0	923.7	0.0	108.7
R15K3T3	0	0	10	-	-	3600.0	9	1	948.1	0.19	589.5
sum.	**46**	**24**	**80**				**149**	**1**			
avg.				**589.5**	**6.4**	**2632.2**			**676.2**	**0.01**	**96.1**

As is shown in the table above, the proposed model is superior to the RAC model in terms of solution quality and computing time. The RAC model obtains the optimal solutions and feasible solutions for 46 and 24 instances, respectively. There are 80 instances that the RAC model does not find feasible solutions. The average computing time is 2632 seconds and the average gap for the solvable instances (at least find one feasible solution) is 6.4%. By contrast, the proposed model solves 149 instances to optimality. Only 1 instance in the R15K3T3 group is not solved to optimality. A feasible solution with a gap of 1.9% is found for that instance, which makes the average gap of the group 0.19%. The proposed model reduces the computing time to 96 seconds, a reduction of 96.35%.

We will then compare the performance of the two models based on different instance scales. We roughly divide the instances into small-, medium-, large-scale, and extra-large-scale based on the capability that the RAC model can solve the instances. Instances with 5 requests are small-scale as the RAC model can easily solve them to optimality. The average computing time ranges from 101 seconds to 537 seconds. The proposed model reduces the average computing time to 0.5 seconds. Instances with 7 to 10 requests are considered medium-scale as the RAC model can solve them but may not be able to solve them to optimality. The average computing time ranges from 1844 seconds to 3600 seconds. The proposed model solves all medium-scale instances to optimality within 22 seconds on average. The instances with 12 to 15 requests are considered large-scale as the RAC model cannot even find feasible solutions for them. For the 60 large-scale instances, the RAC model only finds a feasible solution for one instance. In contrast, the proposed model solves 59 instances to optimality. One instance is not solved to optimality but a feasible solution with a gap of 1.9 % is obtained. The average computing time of solving the large-scale instances ranges from 44 seconds to 590 seconds.

Instances with more than 15 requests are not solvable for the RAC model. In order to evaluate the maximum scale of instances that the proposed model can solve, we further test the 90 extra-large-scale instances. Table 2.6.2 show that the proposed model can solve instances with up to 25 requests and 2 transfer stations. In the literature, the benchmarks of PDP-T are instances with 5 requests and 1 transfer station (see Rais et al. 2014).

Table 2.6.2: Results of the proposed model in solving extra-large-scale PDP-T

Instance	#opt.	#lim.	#no.	obj.	gap (%)	*t*(s)
R20K2T1	5	4	1	1163.7	2.8	2146.1
R20K2T2	4	2	4	1155.5	1.0	2815.4
R20K3T3	3	6	1	1186.0	8.1	2714.1
R25K2T1	1	1	8	1439.5	4.3	3358.5
R25K2T2	1	1	8	1395.4	8.5	3564.4
R25K3T3	0	3	7	1506.5	11.2	3600.0
R30K2T1	0	1	9	1574.8	10.3	3600.0
R30K2T2	0	0	10	-	-	3600.0
R30K3T3	0	0	10	-	-	3600.0
sum.	14	18	58			
avg.				**1345.9**	**6.6**	**3222.1**

The performances of the proposed model and the RAC model are compared when the number of vehicles increases from two to twenty. 100 instances are generated and tested where the number of requests is 5 and the number of transfer station is 1. The time limit for running the two models is set to one hour. The results are shown in Table 2.6.3. The columns share the same meaning as previously described. For the RAC model, it solves all 10 instances to optimality when the number of vehicles is 2. The average computational time is 274 seconds. For instances with more vehicles, the RAC model does not guarantee the optimality. When the number of vehicles increases to 20, the RAC model fails to obtain a feasible solution. In comparison, the proposed model scales well when the number of vehicles is increased. Specifically, all 100 instances are solved to optimality. For the first 40 instances, the computing time increase linearly with the number of vehicles. Although the linear relationship between computational time and the number of vehicles does not maintain for larger instances, the computing time increases moderately as the number of vehicles increases. The average computational time for solving the 100 instances is 2.3 seconds, which is significantly less than the average computational time for the RAC model on solving the instances.

2.6.3 Results of solving the PDPTW-T instances

Table 2.6.4 compares the performance of the proposed model and the existing model in solving the PDPTW-T instances with 3 requests and 4 vehicles under the MD-4T setting and MD-5T setting, respectively. The SSVW model refers to the model presented by Sampaio et al. (2020). It has been modified in Section 4. MD-4T indicates that the number of transfer stations is four. Similarly, MD-5T means that the number of transfer stations is five in the instances. The first column lists the name of the instances. There are 9 groups for each setting. Each group includes 10 different instances. The values in the group name represent the vehicle shift length. For example, 180 means that the shift lengths of the vehicles are 180 minutes. The letters in the group name indicate the type of the requests. Specifically, the group marked as long (L) only contains long-distance requests that the distance between a pickup and a delivery location is at least 60 units. The group marked as

Table 2.6.3: Performance of the models when the number of vehicles increases

	RAC Model						Proposed Model		
Instance	#opt.	#lim.	#no.	obj.	gap (%)	*t*(s)	#opt.	obj.	*t*(s)
R5K2T1	10	0	0	468.1	0.00	274.3	10	468.1	0.2
R5K4T1	5	5	0	470.0	2.06	1850.2	10	470.0	0.4
R5K6T1	4	5	1	592.0	5.67	2379.7	10	587.7	0.5
R5K8T1	3	6	1	669.9	6.86	2647.1	10	665.1	0.8
R5K10T1	4	4	2	729.4	4.81	2770.4	10	742.1	1.2
R5K12T1	2	1	7	789.5	8.39	3282.0	10	826.8	2.5
R5K14T1	1	3	6	863.8	2.55	3394.4	10	896.6	3.4
R5K16T1	0	3	7	1851.8	39.60	3601.0	10	1049.3	3.7
R5K18T1	0	1	9	3129.5	70.67	3600.8	10	1115.8	4.4
R5K20T1	0	0	10	-	-	3600.0	10	1180.0	6.1
sum.	**29**	**28**	**43**				**100**		
avg.				**1062.7**	**15.62**	**2740.0**		**800.2**	**2.3**

Table 2.6.4: Results of solving the PDPTW-T with MD-4T and MD-5T

	MD-4T setting				MD-5T setting			
Instance	*obj.*	$t_{ssvw}(s)$	$t_{pro}(s)$	$t_{reduce}(\%)$	*obj.*	$t_{ssvw}(s)$	$t_{pro}(s)$	$t_{reduce}(\%)$
180, L	411.75	39.38	7.89	79.96	400.83	84.53	32.82	61.18
180, M	330.91	28.82	8.56	70.31	328.53	99.76	25.95	73.99
180, S	261.32	3.88	3.73	4.02	261.00	8.17	6.02	26.33
240, L	376.67	9.54	4.84	49.26	372.36	27.28	14.24	47.81
240, M	299.72	7.74	4.27	44.84	298.66	34.57	9.12	73.62
240, S	251.64	2.16	2.46	-14.00	251.64	8.37	4.60	44.98
300, L	376.45	9.30	6.14	33.98	372.20	25.94	13.26	48.91
300, M	299.72	6.23	4.77	23.47	298.66	30.42	9.90	67.45
300, S	251.64	2.23	2.55	-14.48	251.64	3.98	4.00	-0.72
avg.	**317.76**	**12.14**	**5.02**	**30.82**	**315.06**	**35.89**	**13.32**	**49.28**

short (S) only includes short-distance requests that the distance between a pickup and a delivery location is less than 60 units but more than 30 units. The group marked as mix (M) is the third scenario, which has both long-distance requests and short-distance requests. Column *obj*. reports the average objective function values. Column t_{SSVW} reports the average computing time for the SSVW model while column t_{pro} reports the average computing time for the proposed model. Column t_{reduce} reports the percentage of time reduced by using the proposed model.

All instances in Table 2.6.4 are solved to optimality by using both the SSVW model and the proposed model. As is shown, the proposed model is more efficient in terms of computing time. In the MD-4T setting, the proposed model reduces the average computing time by 30.82%. In the MD-5T setting, the proposed model reduces the average computing time by 49.28%. The average time saving is 40.05%. However, there are three groups that the proposed model consumes more time than the SSVW model, i.e., groups (240, S) and (300, S) under the MD-4T setting, and group (300, S) under the MD-5T setting.

By comparing the two settings, we find that the instances under the MD-5T setting are hard to solve but have smaller average objective function values. This is to be expected as the additional one transfer station provides the possibility of transferring the requests more efficiently. Since our focus is not to study the impact of a different number of transfer stations, we only use the MD-4T setting for PDPTW-T instances with 4 requests and 5 requests.

Table 2.6.5 compares the performance of the SSVW model and the proposed model in solving the PDPTW-T instances with 4 requests and 4 vehicles under the MD-4T setting. The first column of the table shows the name of the groups. The meaning of the other columns is the same as that in Table 2.6.1. Note that we do not list column *#no* for the proposed model because all the values are zero. In other words, the solver Gurobi finds feasible solutions for all 90 instances by using the proposed model. As is shown in the last second row, the proposed model solves more instances to optimality in a shorter time compared with the SSVW model. Specifically, the SSVW model solves 84 instances to optimality. There are 6 instances that the SSVW model does not find optimal solutions.

Table 2.6.5: Results of the models in solving PDPTW-T with 4 requests

	SSVW Model						Proposed Model				
Instance	*#opt.*	*#lim.*	*#no.*	*obj.*	*gap(%)*	*t(s)*	*#opt.*	*#lim.*	*obj.*	*gap(%)*	*t(s)*
180, L	8	1	1	486.77	1.24	1027	10	0	491.32	0.00	168
180, M	7	2	1	418.85	3.78	1442	9	1	427.00	0.44	683
180, S	9	1	0	334.53	0.43	673	10	0	334.53	0.00	119
240, L	10	0	0	445.78	0.00	327	10	0	445.78	0.00	86
240, M	10	0	0	374.48	0.00	263	10	0	374.48	0.00	64
240, S	10	0	0	322.41	0.00	46	10	0	322.41	0.00	52
300, L	10	0	0	444.67	0.00	351	10	0	444.67	0.00	88
300, M	10	0	0	374.48	0.00	246	10	0	374.48	0.00	68
300, S	10	0	0	322.41	0.00	51	10	0	322.41	0.00	45
sum	**84**	**4**	**2**				**89**	**1**			
avg.				**391.60**	**0.61**	**492**			**393.01**	**0.05**	**152**

It fails to find feasible solutions for two of them. In contrast, our model solves 89 instances to optimality. There is only 1 instance that the proposed model does not solve to optimality but finds a feasible solution.

We report the average objective function values, average computing times, and average gaps in the last row in Table 2.6.4. It shows that the proposed model outperforms the SSVW model. The proposed model reduces the average gaps from 0.61% to 0.05% and reduces the average computing time from 492 seconds to 152 seconds. Note that it is not fair to directly compare columns #obj for the two models because instances that are not solvable have been excluded. For example, the average objective function value obtained by the SSVW model is 391.6 while the value is 393.01 for the proposed model. However, the proposed model finds better or equally good solutions for all instances. The SSVW model has a smaller average objective function value because two instances are excluded. Computational results for every single instance can be found at Lyu & Yu (2022).

Table 2.6.6 compares the performance of the SSVW model and the proposed model in solving the PDPTW-T instances with 5 requests and 4 vehicles under the MD-4T setting. This table is similar to Table 4 but deals with instances with one more request. As expected, the instances with five requests are harder to be solved. The SSVW model solves only 46 instances to optimality, accounting for about half of the instances. It finds feasible solutions for 24 instances and fails to solve the remaining 20 instances. The proposed model performs better than the SSVW model. It solves 72 instances to optimality. It finds feasible solutions for 9 instances and fails to solve the remaining 9 instances. In addition, it reduces the average computing time from 2149 seconds to 1334 seconds.

2.6.4 Discussion on valid inequalities

The reason that the proposed model outperforms the RAC model and SSVW model is that we add several valid inequalities (cuts). These cuts are redundant in terms of restricting the feasible region but can strengthen the LP relaxation of the model. As a result, the performance of the proposed model is increase significantly.

Table 2.6.6:Results of the models in solving PDPTW-T with 5 requests

	SSVW Model						Proposed Model					
Instance	*#opt.*	*#lim.*	*#no.*	*obj.*	*gap(%)*	*t(s)*	*#opt.*	*#lim.*	*#no.*	*obj.*	*gap(%)*	*t(s)*
180, L	1	1	8	589.7	12.67	3450	2	3	5	573.0	4.76	2958
180, M	1	1	8	464.5	8.39	3275	5	1	4	467.4	1.85	2550
180, S	7	2	1	409.5	2.32	1597	10	0	0	404.9	0.00	676
240, L	4	6	0	550.7	11.84	2880	9	1	0	528.3	1.29	1266
240, M	6	2	2	442.1	4.66	1762	8	2	0	452.0	1.71	1341
240, S	9	1	0	389.9	0.53	807	10	0	0	389.9	0.00	284
300, L	3	6	1	532.7	9.96	2926	9	1	0	527.9	1.62	1353
300, M	6	4	0	451.5	4.71	1852	9	1	0	449.9	1.02	1252
300, S	9	1	0	390.6	0.89	796	10	0	0	389.7	0.00	322
sum	**46**	**24**	**20**				**72**	**9**	**9**			
avg.				**469.0**	**6.22**	**2149**				**464.8**	**1.36**	**1334**

In this section, we explicitly show how the proposed cuts affect the RAC model by adding constraints (2.5.1) to constraints (2.5.8) into the RAC model, respectively. We also evaluate the performance of the RAC model when constraints (2.5.1) to constraints (2.5.8) are added in the same time. The instances used to perform the experiments are ten small PDP-T instances under the category R5K2T1. There are five requests, two vehicles, and one transfer station in each of the instance. Since the scale of these instances is small, all instances are solved to optimality by using any of the models evaluated. The metric used to compare performance is computing time. Table 2.7.1 shows the results of these experiments.

As has been mentioned before, the column RAC represents the model presented by Rais et al. (2014) with the necessary revisions. The column Prop. represents the proposed model. The columns C40 to C47 represent the RAC model with constraints (2.5.1) to constraints (2.5.8), respectively. The column ALL represents the RAC model with all cuts added together. From the table, we find that adding some of the cuts can significantly improve the RAC model. For instance, adding the first two constraints can reduce the average computing time of solving the instances from 86 seconds to less than 10 seconds. By contrast, some cuts do not affect the efficiency much, such as constraints (2.5.3) and constraint (2.5.5). We keep these constraints because they are not redundant for the proposed model. For example, constraints (2.5.3) are essential for the proposed model to make sure the vehicles end their routes at the destination depots. After adding all the cuts, the performance of the RAC model is very close to the proposed model. We do not perform similar experiments on the PDPTW-T instances because the SSVW model assumes that the origins and destinations of the vehicles are the same. This assumption prevents us from adding most of the constraints into the model without modification.

2.7 Conclusion

This paper reviews two existing models for the PDP-T and PDPTW-T. We point out the possible issues existing in the models, discuss the causes, and provide our revisions. In

Table 2.7.1: Computational time of solving PDP-T with different cuts

	RAC	Prop.	C1	C2	C3	C4	C5	C6	C7	C8	ALL
R5K2T1-0	20.29	0.08	2.64	0.55	22.24	4.53	17.06	1.92	4.84	17.09	0.16
R5K2T1-1	107.02	0.16	21.54	7.96	88.46	18.44	71.07	8.32	27.75	59.97	0.21
R5K2T1-2	57.10	0.11	10.78	13.48	30.07	16.08	63.83	1.45	1.54	27.38	0.17
R5K2T1-3	597.34	0.09	1.70	1.51	742.17	1.44	489.06	62.77	108.81	150.51	0.18
R5K2T1-4	2.67	0.08	1.06	0.53	2.08	1.14	1.41	0.68	1.74	1.67	0.18
R5K2T1-5	2.09	0.05	0.44	0.20	3.89	0.32	2.02	0.65	0.18	2.60	0.10
R5K2T1-6	31.67	0.08	4.10	9.10	31.70	5.14	20.22	3.62	0.73	19.97	0.17
R5K2T1-7	23.14	0.11	1.87	2.21	36.31	3.24	12.36	3.39	2.10	9.18	0.20
R5K2T1-8	17.89	0.08	0.29	0.30	25.58	0.33	14.17	38.36	5.22	17.01	0.19
R5K2T1-9	9.58	0.12	2.71	7.59	15.56	4.68	11.88	5.08	6.81	5.95	0.23
avg. time	**86.88**	**0.10**	**4.71**	**4.34**	**99.81**	**5.53**	**70.31**	**12.63**	**15.97**	**31.13**	**0.18**

addition, we present a new MILP model to solve the problems. Compared with the existing models, the proposed model has redundant constraints that can strengthen the LP relaxation. The performance of the proposed model is evaluated using 340 generated PDP-T instances and 360 open-access PDPTW-T instances. Based on our experiments, the proposed model is superior to the existing models in terms of solution quality and computing time. The proposed model solves 149 out of 150 PDP-T instances to optimality where the instance scales are less or equal to 15 requests, 3 vehicles and 3 transfer stations. The largest instance solved to optimality by the proposed model is the one with 25 requests, 2 vehicles and 2 transfer stations. The proposed model scales well when the number of requests increases. The average computational time increases from 0.2 seconds to 6.1 seconds as the number of vehicles increases from 2 to 20. The proposed model outperforms the existing two models mainly because of the cuts added into the model. These cuts are reductant in terms of restricting the feasible region, but they help tighten the relaxation of the MILP model, which helps solve the model faster. Based on our experiments, constraints (2.5.1), (2.5.2), and (2.5.4) are three efficient cuts.

To the best of our knowledge, this paper is the first work that solves PDP-T instances with 25 requests and 2 transfer stations to optimality within 1 hour. In the literature, Rais et al. (2014) report that they solve PDP-T instances with 5 requests to optimality. Wolfinger & Salazar-González (2021) state that the instance with 8 requests, at the time, is the largest that can be solved to optimality for the PDP-T and PDPSLT. For the PDPTW-T, the proposed model solves instances with 5 requests and 4 transfer locations to optimality within 1 hour.

There are several drawbacks for the proposed model that need to be discussed. First, the number of vehicles traveled in the network needs to be determined in advance. This limits the flexibility of using the model. Second, the order in which two nodes are visited cannot be identified properly if the nodes are in the same location. A tiny distance needs to be added between the coincident nodes. Third, the value $s_{tr}^{k_1 k_2}$ may be false positive although this would not affect the correctness of the vehicle flows and request flows. This issue can

be solved by adding $s_{tr}^{k_1 k_2}$ into the objective function, but this makes the meaning of the objective function ambiguous.

The future research may include fixing the drawbacks discussed above. In addition, we may extend the proposed model to handle larger problems by designing efficient exact algorithms, such as the branch and cut algorithm, branch and price algorithm, etc. Our preliminary results show that we may not benefit much from a branch and cut algorithm that simply adding the redundant constraints presented in this paper as cuts. The reason may be the number of proposed cuts is moderate. To achieve significantly better results, more efficient cuts need to be designed.

Chapter 3. A Metaheuristic for Solving the Consultant Assignment and Routing Problems

This chapter is based on a paper published by Zefeng Lyu and Andrew J. Yu:

> Lyu, Z., & Yu, A. J. (2021). Consultant assignment and routing problem with priority matching. *Computers & Industrial Engineering*, 151, 106921.

In this chapter, we solved a consultant assignment and routing problem that simultaneously assign consultant supplies to client demands and determine the best traveling routes for consultants. Constraints to be considered include skill requirement, capacity limitation, fixed demand, and a maximum number of travel legs. This paper further takes into consideration priority matching, which restricts that clients can only be assigned to consultants with appropriate priority levels. In order to solve this problem, we present a decomposition algorithm named RMIP and a MIP-based neighborhood search algorithm. In addition, we extend an existing MILP formulation and compare our algorithms with it. The effectiveness and efficiency of the proposed algorithms are evaluated on a hundred synthetic instances and twelve real-life instances. Computational results show that the modified MILP formulation is only suitable for solving small-scale instances and a part of the medium-scale instances. For large-scale and real-life instances, the proposed two algorithms are significantly superior to the MILP formulation in solution quality and computational time.

3.1 Introduction

As the energy costs are dramatically rising, more and more educational institutions are looking for ways to decrease their utility bills. Energy Education, Inc (EEI) is one of the consulting firms helping customers develop energy efficiency and conservation programs. This paper is motivated by an optimization problem faced by EEI. This problem is named as consultant assignment and routing problem with priority matching (CARPP), which aims to simultaneously assign consultants to clients and determine the traveling routes while minimizing the total cost. This work is traditionally performed manually by subject matter experts of the firm. However, developing such a working schedule is very complex and labor-intensive. It takes experienced experts sixteen hours of dedicated effort each

week, but the produced schedule is still often far from optimal. Therefore, it is crucial to design an efficient algorithm to support decision making for EEI.

The features of CARPP problem are listed as follows. The consulting activities must be performed face-to-face at the client site. Therefore, consultants travel to almost all client assignments to fulfill their jobs. The objective is to minimize the incurred airfares and the consultants' wages. Consultants are proficient in different skills and they can only fill demands requiring the relevant skill. The skills cannot be easily replicated or cross-trained. In addition, matching priority level is required before a consultant can be assigned to serve a client. The practical meaning of priority is basically a measure of importance for clients and a measure of seniority or experience for consultants. Clients with higher priority mean that they are relatively more important due to their projects' scope, stage, size, and other aspects. For example, some contracts may have a large amount of budget and coverage. Some clients may be new, and the company wants to have a successful start. Importance may also be related to technical perspective, meaning that the corresponding tasks are more challenge than the others. As a result, the company would like to include priority into the model to make sure that consultants with proper level of seniority or experience are assigned to their clients. EEI stipulates that the maximum number of flying trips each week should not exceed four to avoid dissatisfaction from the consultants. The CARPP problem does not need to consider time window because the company only specifies the overall work packages for their consultants. The specific working time within a day is determined after discussing with clients, as it is subject to the flight schedule, which is not under the consultants' control.

The CARPP problem is extended from the consultant assignment and routing problem presented by Yu & Hoff (2013). They developed a MILP formulation to solve their problem. However, solving the problem using MILP formulation becomes time-consuming after taking priority matching into consideration. Therefore, this paper develops the RMIP algorithm and the MIP-based neighborhood search algorithm (MNSA). The RMIP algorithm first decomposes the problem into serval subproblems via a reduced formulation. Then, these subproblems are solved by exhaustive method. The exhaustive method is

efficient to solve the subproblems because the maximum travel legs are limited by four. The MNSA algorithm is a metaheuristic that optimizes the RMIP solutions using neighborhood search. In addition, this paper modifies the existing MILP formulation and compares our algorithms with it.

The contributions of this paper are threefold. First, we introduce a new variant of the consultant assignment and routing problem, which takes priority matching into consideration. Second, we propose a decomposition method and a metaheuristic for the problem. Third, we generate several synthetic instances based on the real-world data. These synthetic instances have been published online together with the computational results, which can be used as a benchmark for future comparison.

The rest of this paper is organized as follows. Section 2.2 reviews the related literatures. Section 2.3 defines the CARPP problem and presents the modified MILP formulation. Section 2.4 introduces the decomposition algorithm. Section 2.5 presents the MNSA algorithm and relevant pseudo-codes. Section 2.6 introduces the real-life instances and the generation of synthetic instances. In addition, all computational results are shown in this section. Finally, conclusion and future research are discussed in Section 2.7.

3.2 Literature Review

The field of this paper has received a great amount of attention in recent years. There are many close-related problems such as the home health care problem, home care problem, technician scheduling for maintenance, manpower allocation, etc. Survey papers are given by Paraskevopoulos, Laporte, Repoussis, & Tarantilis (2017) and Castillo-Salazar, Landa-Silva, & Qu (2016). However, there is no agreed terminology to inductive these problems. For example, Maya, Sörensen, & Goos (2012) and Yalçındaˇg, Matta, ¸Sahin, & Shanthikumar (2014) considered their problem as an assignment and routing problem. Paraskevopoulos et al. (2017) classified this kind of problem as resources constrained routing and scheduling problem. Castillo-Salazar et al. (2016) referred these problems as workforce scheduling and routing problem. Some researchers also considered the problem

as a variant of the vehicle routing problem (VRP), such as in Rasmussen, Justesen, Dohn, & Larsen (2012) and Song & Ko (2016).

This paper does not classify our problem as a VRP because the main considerations of these two problems are different. For instance, most of the VRPs only focus on finding the optimal routes for vehicles or other objects. However, the CARPP problem treats the assignment of consultants and the routing procedure equally. Many constraints are designed for the assignment procedure such as skill matching, priority matching, and available shifts. If we do not consider the routing part, the remaining problem is still a workforce scheduling problem. In addition, the general applications of CARPP and VRP are quite different. VRP is usually associated with transportation problems. In contrast, assignment and routing problem is more common when workforces are involved.

The CARPP problem is extended from the consultant assignment and routing problem introduced by Yu & Hoff (2013). Features of their problem include skill matching, capacity limitation, fixed demand, and maximum traveling legs. Compared with their work, this paper takes a new constraint into consideration, i.e. priority matching. In terms of the methodology, they presented a two-stage approach, which first clusters the clients by a set-covering model and then solves the problem by a MILP formulation. Different from their approach, this paper proposes a decomposition method and a metaheuristic. In addition, we extend their MILP model to compare with our algorithms. Computational results show that the modified MILP formulation is only suitable for solving small-scale instances and a part of the medium-scale instances. For large-scale and real-life instances, the proposed two algorithms are superior to the MILP formulation.

Maya et al. (2012) introduced a teaching assistant assignment and routing problem (TARP), which aims to minimize the total traveling compensation received by teaching assistants. Different from the TARP, our problem considers both the traveling cost and wage of consultants. Kovacs, Parragh, Doerner, & Hartl (2012) presented a service technician routing and scheduling problem (STRSP) where technicians are assigned to complete service tasks. Similar problems are also studied by Xie, Potts, & Bektaş (2017) and Zamorano & Stolletz (2017). In contrast to the technician scheduling problem, team

building up and outsourcing are not allowed in the CARPP problem. However, our problem has more restrictions on the assignment procedure, such as priority matching, fixed demand, and maximum number of flight legs. Home care and home healthcare problems are also closely related and gain increasing interests recently. Review papers are given by Fikar & Hirsch (2017) and Cissé et al. (2017). The difference between home care and home healthcare is that the former refers to housework such as laundry and cleaning while the latter refers to healthcare activities typically performed by qualified nurses. Yuan, Liu, & Jiang (2015) presented a home health care problem where caregivers are classified into several levels. They restrict that high-level demands can only be assigned to highly qualified caregivers, but low-level demands can be assigned to any of the caregivers. Unlike their problem, our skill requirement is a one-to-one matching. In addition, our problem restricts the number of flight trips rather than the number of clients to serve. Eveborn et al. (2009) presented a home care problem that assigns customer requests to schedules and then assigns these schedules to staff members. Skill requirement is not explicitly considered in their problem. Other relevant applications include waste collection (De Bruecker, Beliën, De Boeck, De Jaeger, & Demeulemeester, 2018), airline catering (Ho & Leung, 2010), and inventory routing problem such as in Misra, Saxena, Kapadi, Gudi, & Srihari (2018), Maheshwari, Misra, Gudi, & Subbiah (2020), and Dong, Pinto, Sundaramoorthy, & Maravelias (2014). The inventory routing problem incorporates the production problem and vehicle routing problem. However, the considerations for controlling inventory are different from assigning consultants to clients as studied in this paper.

It was found that most of the relevant problems involve multi-period horizons because of the nature of scheduling. Examples can be seen in De Bruecker et al. (2018), An, Kim, Jeong, & Kim (2012), and Zamorano & Stolletz (2017). However, the CARPP problem is a single-period problem although the planning horizon is one week. This is because the roster only specifies the client to visit and the sequence. The specific consulting date and time are decided by consultants and clients.

Although close-related problems have been studied extensively, little attention has been paid to the consultant assignment and routing problem. To the best of our knowledge, none heuristic has been presented to solve the CARPP. Therefore, this paper develops the RMIP algorithm and the MNSA algorithm to fill this gap.

3.3 Problem Definition and MILP Formulation

The CARPP problem is defined on a connected graph $G = \{K, C, A\}$, where K is a set of consultants, C is a set of clusters, and A is a set of arcs representing available visiting routes. Consultants and clients live in different geographical locations. The set A includes not only the routes between consultants and clients but also routes between clients and clients so that consultants can travel from one client location to another. Each arc is associated with a specific traveling cost. Except for travel expenses, the wage of consultants is another part of cost considered for those consultants who are contract-based. The objective is to assign the consultant supply to client demand while minimizing the total cost. After solving the CARPP problem, we are supposed to provide a detailed working schedule for every consultant, as well as the corresponding visiting sequences. Considering that flight is the main mode of transportation, it is reasonable to cluster consultants and clients into the nearest airports. If there are too many airports involved, partial airports can be clustered to reduce computational complexity. However, whether the locations can be clustered, the specific way of clustering depends on the operational rules of the company. Although clustering can reduce the computational complexity, it may also lead to a loss of global optimality. Therefore, clustering is not the scope of this paper. In this paper, the nearest airports for the consultants and customers are predefined. We only focus on the methodology regarding assignment and routing.

The constraints in our problem include demand satisfaction, fixed demand, skill matching, priority matching, capacity restriction, and trip legs restriction. The explanations of these constraints are as follows. Demand satisfaction restricts that all the demands must be satisfied. Some of the demands are fixed demand, which must be served by designated

consultants. Each demand has one specific skill requirement from its visiting consultant, and each consultant has a unique set of skills to serve. The skill matching makes sure that a demand can be assigned to a consultant only if the consultant has the corresponding skill. Please note that the priority matching in the paper does not mean that demands with high priority should be completed earlier than the others. In fact, it refers to a level matching between the consultants and demands. For example, a demand with a priority level of four can be assigned to a consultant whose priority interval is between three and five. Capacity restriction ensures that the sum of shifts assigned to a consultant should not exceed his/her total available shifts. The maximum number of flight legs is limited to four per week to avoid the consultant's dissatisfaction. The unit "shift" is used to measure the workload. One shift means that the corresponding demand can be fulfilled in a half day by one consultant. Please note that travel time is not a part of working shifts because consultants have flexible schedules. The roster only specifies client demands and the corresponding visiting sequences. However, the specific working time is determined by the consultant and client. There is no need to worry about the travel time for our problem. In addition, it is worth mentioning that travel time is not billable to the clients and consultants are not paid for their travel time.

This paper assumes that the flight prices are symmetric and do not vary by time. Otherwise, the CARPP problem would become time-related and much more intractable. This paper ignores travel costs other than airfare because airline tickets are one of the EEI's biggest budget items. It is also worth mentioning that CARPP is a single-period problem although the planning horizon is one week. This is because the roster only specifies the clients and the visiting order. The specific working date and time shall be decided by the consultant and client. The notations for sets, parameters and variables are shown as follows.

Sets	
K	A set of consultants
C	A set of clusters
S	A set of skills

P	A set of priorities
Π	A set which combines set S and set P
N	$N = \{1, 2, \ldots, T+1\}$, where T is the maximum trip legs allowed
Parameters	
c_{ij}	Traveling cost from cluster i to cluster j
q_{kj}	Traveling cost from consultant k's home cluster to cluster j
w_k	Weekly wage of consultant k
a_k	Number of available shifts of consultant k
h_k	The home cluster of consultant k
d_{js}	Number of s-type demand requested from cluster j
f_{kjs}	Number of s-type demand requested from cluster j to be served by consultant k
u_{ks}	=1 if consultant k has s-type skill; 0 otherwise.
M	A sufficiently large positive number
Variables	
x_{kjs}	Number of s-type demand assigned to consultant k in cluster j
y_{knij}	=1 if consultant k travels from cluster i to cluster j on the nth trip; 0 otherwise. Note that cluster 0 represents the home cluster which varies for different consultants.
z_{kj}	=1 if consultant k has demand to serve in cluster j; 0 otherwise.

Since the priority requirement and skill requirement are similar in structure, this paper combines them together and forms a new constraint which restricts consultants to satisfy both skill and priority requirement simultaneously. To achieve it, this paper defines an aggregated set Π as follows.

$$\Pi = \{s \mid s = (i, j), i \in S, j \in P\}$$

For example, if the skill set $S = \{s_1, s_2, s_3\}$ and the priority set $P = \{p_1, p_2\}$, then the aggregated set is combined as

$$\Pi = \{s_1p_1, s_1p_2, s_2p_1, s_2p_2, s_3p_1, s_3p_2\}$$

The MILP formulation for the CARPP problem is listed as follows.

Minimize $$\sum_{k\in K}\sum_{j\in C}(q_{kj}+w_k)y_{k10j} + \sum_{k\in K}\sum_{n\in N\backslash\{1\}}\sum_{i\in C}\sum_{j\in C} c_{ij}y_{knij} + \sum_{k\in K}\sum_{n\in N\backslash\{1\}}\sum_{j\in C} q_{kj}y_{knj0}$$

(MILP)

Subject to

$$\sum_{k\in K} x_{kjs} = d_{js} \qquad \forall j\in C, \forall s\in \Pi \qquad (3.3.1)$$

$$x_{kjs} \geq f_{kjs} \qquad \forall k\in K, \forall j\in C, \forall s\in \Pi \qquad (3.3.2)$$

$$x_{kjs} \leq Mu_{ks} \qquad \forall k\in K, \forall j\in C, \forall s\in \Pi \qquad (3.3.3)$$

$$\sum_{j\in C}\sum_{s\in S} x_{kjs} \leq a_k \qquad \forall k\in K \qquad (3.3.4)$$

$$y_{k10i} \geq \sum_{j\in C\cup\{0\}} y_{k2ij} \qquad \forall k\in K, \forall i\in C \qquad (3.3.5)$$

$$\sum_{i\in C} y_{knij} \geq \sum_{h\in C\cup\{0\}} y_{k,n+1,j,h} \qquad \forall k\in K, \forall n\in N\backslash\{1, T+1\}, \forall j\in C \qquad (3.3.6)$$

$$\sum_{i\in C} y_{k10i} = \sum_{n\in N\backslash\{1\}}\sum_{j\in C} y_{knj0} \qquad \forall k\in K \qquad (3.3.7)$$

$$\sum_{n\in N}\sum_{j\in C\cup\{0\}} y_{knij} \leq 1 \qquad \forall k\in K, \forall i\in C\cup\{0\} \qquad (3.3.8)$$

$$\sum_{n\in N}\sum_{i\in C\cup\{0\}} y_{knij} \leq 1 \qquad \forall k\in K, \forall j\in C\cup\{0\} \qquad (3.3.9)$$

$$y_{kn0i} = 0 \qquad \forall k\in K, \forall n\in N\backslash\{1\}, \forall i\in C\cup\{0\} \qquad (3.3.10)$$

$$y_{k1ij} = 0 \qquad \forall k\in K, \forall i\in C, \forall j\in C\cup\{0\} \qquad (3.3.11)$$

$$y_{knii} = 0 \qquad \forall k\in K, \forall n\in N, \forall i\in C\cup\{0\} \qquad (3.3.12)$$

$$\sum_{s\in \Pi} x_{kjs} \leq M\sum_{n\in N}\sum_{i\in C\cup\{0\}} y_{knij} \qquad \forall k\in K, \forall j\in C \qquad (3.3.13)$$

$$\sum_{s\in \Pi} x_{kjs} \geq \sum_{n\in N}\sum_{i\in C\cup\{0\}} y_{knij} \qquad \forall k\in K, \forall j\in C \qquad (3.3.14)$$

$$\sum_{s \in S} x_{kjs} \leq M z_{kj} \qquad \forall k \in K, \forall j \in C \qquad (3.3.15)$$

$$\sum_{j \in C \setminus \{h_k\}} z_{kj} \leq T - 1 \qquad \forall k \in K \qquad (3.3.16)$$

$$x_{kjs} \in \mathbb{Z} \qquad \forall k \in K, \forall j \in C, \forall s \in \Pi \qquad (3.3.17)$$

$$y_{knij} \in \{0,1\} \qquad \forall k \in K, \forall n \in N, \forall i \in C \cup \{0\}, \forall j \in C \cup \{0\} \qquad (3.3.18)$$

$$z_{kj} \in \{0,1\} \qquad \forall k \in K, \forall j \in C \qquad (3.3.19)$$

The MILP formulation is extended from the formulation presented by Yu & Hoff (2013). We take priority matching into consideration by combining priority levels and skills, as explained in the definition of the aggregated set Π. Constraints (3.3.1) to (3.3.14) are borrowed from the existing model. We add variables z_{kj} and constraints (3.3.15) and (3.3.16) into our model because the original one is too tight under some cases. For example, if the first or last leg of a consultant's scheduled trip happens to be at his/her home depot, the consultant does not consume a flight leg for that trip. We allow n to be $T + 1$ and add the new variables and constrains to ensure that flight legs are appropriately restricted. Explanations for the MILP formulation are as follows.

The objective of the model is to minimize the total cost, which consists of the weekly wage of contract consultants and associated traveling expense. The first part of the objective function models the wage and the airfare of the consultants traveling from their home cluster to their first destinations. The second part represents the airfare between clusters. The last part denotes the airfare for the consultants to travel back home.

Constraints (3.3.1) to (3.3.4) model the demand satisfaction, fixed demand, skill matching, and capacity restriction, respectively. Constraint (3.3.1) ensures that all the demands are fully satisfied. Constraint (3.3.2) restricts that the fixed demands are served by the designated consultants. Constraint (3.3.3) indicates that demands can only be met by those consultants with the relevant skill and priority. Note that the M here is a sufficiently large positive number. It is reasonable to set M to a_k, i.e., the available shift of

consultant k. Constraint (3.3.4) means that the total number of shifts assigned to a consultant should not exceed his/her availability.

Constraints (3.3.5) to (3.3.12) model the traveling routes of consultants. Specifically, constraint (3.3.5) indicates that the consultants must start the travel from their home clusters. Constraint (3.3.6) restricts the relationship between current visit and the next visit. Constraint (3.3.7) ensures every consultant should return home after completing the tasks. Constraints (3.3.8) and (3.3.9) specify that a consultant travels to a cluster at most once. Constraints (3.3.10) make sure that, except for the first trip, the consultants are not allowed to leave their home clusters. In other words, the consultants cannot continue their trips once they return. Constraints (3.3.11) ensure that the consultants cannot start from clusters that are not their home clusters. The constraints also ensure that return at the first trip is prohibited. Constraint (3.3.12) prohibits the self-access.

Constraints (3.3.13) and (3.3.14) associate the assignment plans and the traveling routes. Constraint (3.3.13) means that a consultant can serve a demand only if the consultant visits the corresponding cluster. Constraint (3.3.14) ensures that consultants never go to clusters where they do not have any tasks assigned. Constraints (3.3.15) and (3.3.16) limit the total number of trip legs for each consultant. Given that the maximum number of trip legs in a period is T, each consultant is allowed to visit at most $T-1$ clusters outside the home cluster. Just like in the constraints (3.3.3), the value of M in constraints (3.3.13) and (3.3.15) can also be reasonably set to a_k. Constraints (3.3.17)to (3.3.19) restrict the decision variable x_{kjs} to be integers, and y_{knij} and z_{kj} to be binary variables.

3.4 RMIP algorithm

The CARPP problem requires us to solve both the assignment problem and the routing problem simultaneously. It can also be expressed as a problem shown below.

$$\arg\min_{x} F(x), \quad \forall x \in X \qquad (P1)$$

Notation X is a set of feasible allocation plans and x is one specific plan in X. Notation F is a function which constructs the optimal routes for an allocation plan and calculates the total cost. Solving problem $P1$ is time-consuming when the problem scale is large. The huge number of feasible allocations is one of the reasons. More importantly, the function F has to figure out the optimal routes for each consultant, which is equivalent to solving m traveling salesman problems (m-TSP) where m is the number of consultants.

Since problem $P1$ is too difficult to solve, we present an alternative problem $P2$ as follows,

$$\arg\min_{x} F'(x), \quad \forall x \in X \qquad (P2)$$

The function F' has the similar function as function F but it does not need to find out the optimal routes. It minimizes the summation of the roundtrip airfares for each cluster from a visiting consultant's home cluster. Without the need of determining the traveling sequences, the problem $P2$ is much easier to be solved than the original problem. Figure 2 and Figure 3 demonstrate how the idea works.

Figure 3.4.1 shows the optimal solution obtained by solving problem $P1$. The rectangular A and B denote two consultants, and the circles 1 to 6 represent six clusters. As was shown, consultant A visits cluster 1, 2, and 3 in sequence, and consultant B visits cluster 5, 6, and 7 in sequence. Figure 3.4.2 shows the optimal routes obtained by solving problem $P2$. Instead of flying from one cluster to another, consultants A and B return their home clusters before going to the next cluster.

Although the solutions are different, the allocation plans may be the same just as shown in the given example. Cluster 1, 2, 3 are assigned to consultant A, and cluster 4, 5, 6 are assigned to consultant B in both of the two solutions. It's worth noting that the example in Figure 2 and Figure 3 just shows a special case. The best allocation plan for problem $P2$ is by no means always to be optimal for problem $P1$. However, we found that the generated allocation plan is generally quite good. The underlying idea is that simply assigning consultants to closer clusters, to some extent, can minimize the actual travel distance.

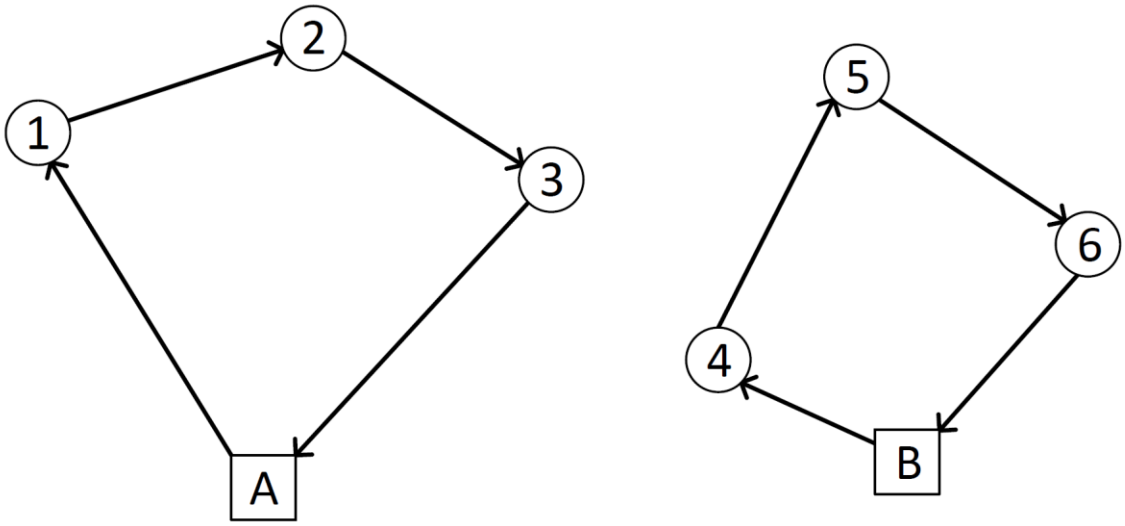

Figure 3.4.1: Optimal Solution of P1

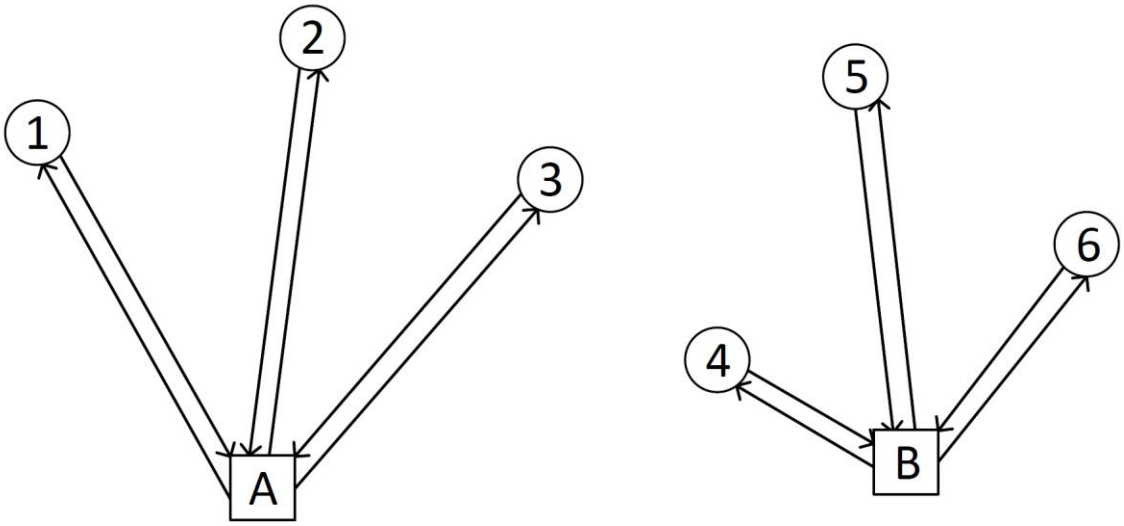

Figure 3.4.2: Optimal Solution of P2

Thus, we can just solve problem $P2$ and then solve m sub-problems to get the same solutions where m is the number of consultants. Based on the idea, we present a reduced mixed-integer linear programming model.

Minimize $$\sum_{k\in K}\sum_{j\in C} 2q_{kj}y_{kj} + \sum_{k\in K} w_k z_k \qquad \text{(Reduced-MILP)}$$

Subject to

$$\sum_{k\in K} x_{kjs} = d_{js} \qquad \forall j \in C, \forall s \in \Pi \qquad (3.4.1)$$

$$x_{kjs} \geq f_{kjs} \qquad \forall k \in K, \forall j \in C, \forall s \in \Pi \qquad (3.4.2)$$

$$x_{kjs} \leq M u_{ks} \qquad \forall k \in K, \forall j \in C, \forall s \in \Pi \qquad (3.4.3)$$

$$\sum_{j\in C}\sum_{s\in \Pi} x_{kjs} \leq a_k \qquad \forall k \in K \qquad (3.4.4)$$

$$\sum_{s\in \Pi} x_{kjs} \leq M y_{kj} \qquad \forall k \in K, \forall j \in C \qquad (3.4.5)$$

$$\sum_{j\in C\setminus\{0\}} y_{kj} \leq T - 1 \qquad \forall\, k \in K \qquad (3.4.6)$$

$$\sum_{j\in C} y_{kj} \leq M z_k \qquad \forall k \in K \qquad (3.4.7)$$

$$x_{kjs} \in \mathbb{Z} \qquad \forall k \in K, \forall j \in C, \forall s \in \Pi \qquad (3.4.8)$$

$$y_{kj} \in \{0,1\} \qquad \forall k \in K, \forall j \in C \qquad (3.4.9)$$

$$z_k \in \{0,1\} \qquad \forall k \in K \qquad (3.4.10)$$

The formulation RMIP aims to minimize a reduced traveling cost and the wage of consultants. The first part models the airfare and the second part models the wage. There are three types of variables in the model. Specifically, x_{kjs} is an integer variable which denotes the number of s-type demand assigned to consultant k from cluster j. Notation y_{kj} is a binary variable. If consultant k has tasks in cluster j, the value of y_{kj} is equal to one. Notation z_k is also a binary variable which records whether consultant k has any work assigned in the week.

Constraints (3.4.1) to (3.4.4) are side constraints regarding the allocation plan. They are the same as what in the formulation MILP. These constraints ensure that the feasible region X of problem $P2$ is the same as that of the problem $P1$. Constraint (3.4.5) indicates that consultant k is not allowed to visit cluster j if he/she has no task there. Constraint (3.4.6) limits the maximum number of trip legs. The consultants visit no more than $T-1$ clusters out of their home cluster. Constraint (3.4.7) restricts that if any demand is assigned to consultant k, the corresponding auxiliary decision variables z_k should be equal to one. The variables z_k are used to calculate the weekly wage. Constraint (3.4.8) to (3.4.10) restrict x_{kjs} , y_{kj} , z_k to integer variables, binary variables, and binary variables, respectively.

The purpose of solving the reduced formulation is to obtain an allocation plan. Then, exhaustive method is applied to obtain best travel routes under that allocation plan. Since the maximum number of flight trip has been limited by four, it is efficient to use exhaustive method to solve the sub-problems.

3.5 MNSA algorithm

We use a neighborhood search algorithm to improve the initial solution obtained by the RMIP formulation. The algorithm is based on the basic framework presented by Hansen, Mladenović, & Moreno Pérez (2010). The input of the algorithm is the initial solution x, and the output is a near-optimal solution x_{best} . t_{max} , n_{max} , and σ_{shake} are three parameters used to tune the algorithm. t_{max} is the maximum computational time. Its value is equal to the time limitation, which is set to be one hour in our experiments, minus the time consumed at RMIP algorithm. n_{max} is the maximum number of iterations without improvement. The MNSA algorithm terminates when the running time or the number of iterations without improvement reaches the limitation. Shaking operator is used to drop out from the local optimal. Notation σ_{shake} indicates the shaking strength, which is used to make a trade-off between the searching efficiency and the searching depth. The pseudo-code of the neighborhood search algorithm is shown in Algorithm 1.

Algorithm 1 *Neighborhood search algorithm*

Input: $x, t_{max}, n_{max}, \ \sigma_{shake}$

Output: x_{best}, t_{run}

$t_{start} \leftarrow currentTime$

$t_{end} \leftarrow t_{start} + t_{max}$

$x_{cur} \leftarrow x$

$x_{best} \leftarrow x$

$n \leftarrow 1$

while $currentTime < t_{end}$ **and** $n < n_{max}$ **do**

    $x_{cur} \leftarrow bestImprovement\ (x_{cur})$

    if $F(x_{cur}) < F(x_{best})$ **then**

        $x_{best} \leftarrow x_{cur}$

        $n \leftarrow 1$

    else

        $x_{cur} \leftarrow shakingOperator\ (x_{best}, \sigma_{shake})$

        $n \leftarrow n + 1$

    end if

end while

$t_{run} \leftarrow t_{start} + currrentTime$

return x_{best}

Function F is called to evaluate the allocation plans. If $F(x_{cur}) < F(x_{best})$, the current solution x_{cur} would be updated as the best solution. Otherwise, we discard the current solution and move to next iteration. The nature of the function F is to find optimal routes for each consultant given the allocation plan. It is equivalent to solve a m-TSP problem, where m is the number of consultants. In our CARPP problem, the consultants are restricted to visit at most four clusters. Thus, an enumerative search is efficient enough to find the optimal routes.

3.5.1 Neighborhood Structure

The neighbors of a solution x is defined to be solutions which can be obtained by one swapping operation. There are two cases to consider. If the number of shifts to be swapped are equal, we swap them completely. If one demand requires more shifts, the demands would be swapped partially because the consultant to receive it may not have enough idle shifts. In this case, the swapping shifts would be equal to the smaller one. Feasibility needs to be checked after each swapping operation or inserting operation. Any move that violates the feasibility must be abandoned. In addition, the fixed demand should be excluded from the operations in case they are changed.

3.5.2 Improving Rules

There are two common improving rules, the First Improvement (FI) and the Best Improvement (BI). Rule FI means that the current iteration stops as soon as it finds a better solution. Rule BI enforces the algorithm to iterate all the neighborhoods of the current solution and then return the best one. If there is no better neighborhood, the current solution would be returned. BI rule can achieve the greatest descent in each iteration, but it may be time-consuming. Hansen et al. (2010) suggests that if the initial solution is chosen at random, using FI rule should be appropriate. However, if some constructive heuristic is used, using BI rule might be better. Our computational results are in consistent with the above statements. Therefore, BI rule is used in the MNSA algorithm. The pseudo-code of the best improvement rule is shown in Algorithm 2.

Algorithm 2 $bestImprovement$

```
Input: x
Output: x'
x' ← x
while True do
    X ← Neighbors(x')
    x'' ← best neighbor among X
    if F(x'') < F(x') then
        x' ← x''
    else
        return x'
    end if
end while
```

3.5.3 Shaking Operator

The shaking operator is a perturbation, which helps the MNSA algorithm escape from a local optimal. Shaking strength σ_{shake} is used to control the intensification and diversification of the MNSA algorithm (Lourenço, Martin, & Stützle, 2003). If σ_{shake} is set too small, the global searching ability is limited, which makes the MNSA algorithm hard to escape from the local optimal. However, if σ_{shake} is set too large, the feature of a good solution would be discarded, which also makes it difficult to obtain good solutions. The pseudo-code of the shaking operator is shown in algorithm 3.

3.6 Numerical Experiments

The formulation MILP and RMIP are coded in Python 3 and solved by the commercial solver Gurobi (version 8.1.0.) running on an x64-based PC. This PC runs Microsoft Windows 10 Pro with Intel Core i7-3770 CPU (3.40GHz) and 8 GB of RAM. The MNSA algorithm is also coded in Python and run in PyCharm on the same machine. The computational results and the synthetic instances are published at Mendeley Data: http://dx.doi.org/10.17632/p4h8w2hmwm.1.

3.6.1 Generation of Synthetic Instances

We generated 16 datasets of which 4 are small-sized, 8 are medium-sized, and another 4 are large-sized. We randomly generated 10 instances for each of the small datasets and 5 instances for each of the medium and large datasets. $4 \times 10 + 12 \times 5 = 100$ synthetic instances are generated in total. The clients and consultants are randomly distributed on a 100×100 Euclidean grid using a continuous uniform distribution. The traveling cost between two clusters is set to be the Euclidean Distance.

Table 3.6.1 shows the parameters and their corresponding values for the synthetic instances. The scale of an instance depends on the values of $|C|, |K|, |S|,$ and $|D|$. After extensive experimentations, we found that the number of clusters $|C|$ is the most critical parameter related to the complexity of an instance. The computational time increased

Algorithm 3 $shakingOperator$

Input: x, σ_{shake}

Output: x'

```
σ ← 1
x' ← x
while σ ≤ σ_shake do
    x'' ← randomNeighbor(x')
    if x'' is feasible then
        x' ← x''
        σ ← σ + 1
    end if
end while
return x'
```

Table 3.6.1: Parameters and the corresponding values for the synthetic datasets

Notation	Meaning	Value
$\|C\|$	Number of clusters	{10, 30, 50}
$\|K\|$	Number of consultants	{20, 40, 60, 100}
$\|S\|$	Number of skills	{10, 30, 50}
$\|D\|$	Number of demands	{150, 300, 500, 800}
R_c	Radio of independent contractor	{0.4, 0.6}
R_f	Radio of fixed demand	0.1
R_s	Radio of learned skill of consultant	{0.5, 0.8}
N_d	Shifts needed for each demand	U(1, 6)
N_c	Available shifts for each consultant	10
W_r	Weekly wage for the regular consultants	0
W_c	Weekly wage for the contract consultant	U(900, 1100)

significantly while the number of clusters was increased. Thus, we divided the datasets into small-, medium-, and large-scales based on $|C|$. The possible values of C are also shown in this table. The parameters $|K|, |S|, |D|$ also affect the complexity of the datasets. Different values are assigned to them to keep the diversity of the instances. The symbol U represents the uniform distribution.

In order to be consistent with the real-life instances, we generated two kinds of consultants, i.e., the regular consultants and contract consultants. The ratio of the contract consultant (R_c) is set to be 0.4 or 0.6. The ratio of fixed demand (R_f) is set to be 0.1 for all the datasets. R_s denotes the ratio of learned skills for the consultants. For example, if the skill pool consists of ten different skills and R_s is equal to 0.5, each consultant would be randomly assigned 5 kinds of skills.

It was found that the value of R_s affects the solution space. If R_s was set too small, there could be no feasible solution because of a lack of specific skills. In our experiment, its value was set to be either 0.5 or 0.8.

Table 3.6.2 shows the specific configuration of parameters for each dataset. Notation *S*, *M*, *L* denotes the small-, medium-, and large-scale dataset, respectively. Take the dataset S1 as an example. Each instance in S1 contains 10 clusters, 20 consultants, 10 skills, and 150 shifts of demands. The ratio of contract consultants is 0.4, which means that there are eight consultants. Likewise, it can be computed that there are 15 shifts of fixed demand, and each consultant is proficient in 5 kinds of skills.

3.6.2 Result of Synthetic Instances

The performances of the MILP algorithm, RMIP algorithm, and MNSA algorithm are evaluated using the synthetic instances. MILP and RMIP are solved by the commercial solver Gurobi. MNSA algorithm takes the results of the RMIP as its initial solutions. Then, these initial solutions are optimized by a neighborhood search algorithm. The parameters n_{max} and σ_{shake} are set to be 30 and 10, respectively. The maximum computational time is limited by 3600 seconds for all the three algorithms.

Table 3.6.2: Configuration of parameters for the synthetic datasets

Dataset	$\lvert C \rvert$	$\lvert K \rvert$	$\lvert S \rvert$	$\lvert D \rvert$	R_c	R_f	R_s
S1	10	20	10	150	0.4	0.1	0.5
S2	10	20	10	150	0.4	0.1	0.8
S3	10	20	10	150	0.6	0.1	0.5
S4	10	20	10	150	0.6	0.1	0.8
M1	30	40	30	300	0.4	0.1	0.5
M2	30	40	30	300	0.4	0.1	0.8
M3	30	40	30	300	0.6	0.1	0.5
M4	30	40	30	300	0.6	0.1	0.8
M5	30	60	30	500	0.4	0.1	0.5
M6	30	60	30	500	0.4	0.1	0.8
M7	30	60	30	500	0.6	0.1	0.5
M8	30	60	30	500	0.6	0.1	0.8
L1	50	100	50	800	0.4	0.1	0.5
L2	50	100	50	800	0.4	0.1	0.8
L3	50	100	50	800	0.6	0.1	0.5
L4	50	100	50	800	0.6	0.1	0.8

Figure 3.6.1 shows the gaps and computational time solved by the three algorithms for small-scale (left), medium-scale (middle), and large-scale (right) instances. The x-axis in the six subfigures represents the index of the instances. The upper three subpictures are associated with the computational gaps. For small-scale instances, the MILP algorithm performs best. It finds optimal solutions for all 40 instances. For medium-scale instances, the solution qualities of the three algorithms are similar. However, there is a tendency that MNSA algorithm gradually outperforms MILP. In addition, MILP algorithm is significantly slower than the other two algorithms. For large-scale instances, the MNSA and RMIP perform better than MILP. The differences of solution quality between MNSA and RMIP are very small. The lower three subpictures are associated with computational time. As is shown, RMIP has the highest efficiency, while MILP is the least efficient algorithm. MILP algorithm reaches the time limit of one hour and terminates in advance for both medium- and large-scale instances. However, the time is sufficient for RMIP algorithm and MNSA algorithm.

It is worth mentioning that the gap, except for the small-scale instances, refers to the percentage difference between the solution obtained and the best lower bound found by the solver. This is because finding the global optimums for medium- and large-scale instances are intractable. Figure 3.6.2 provides an example of solving a medium instance for twenty-four hours.

As demonstrated, the gap undergoes significant reduction within the initial few hours. Subsequently, the gap gradually approaches zero at a slower pace. However, despite the solver being terminated due to overtime, the optimal solution remains elusive. This observation suggests that the task of identifying global optima becomes challenging for medium-scale instances and even intractable for large-scale instances. Therefore, with the exception of small-scale instances, the gap used in this paper refers to a percentage difference between the objective function value and the best lower bound found by the solver.

Table 3.6.3 shows the gaps and computational time for the forty small-scale instances. As is shown, MILP algorithm is able to find the optimal solutions within an average time

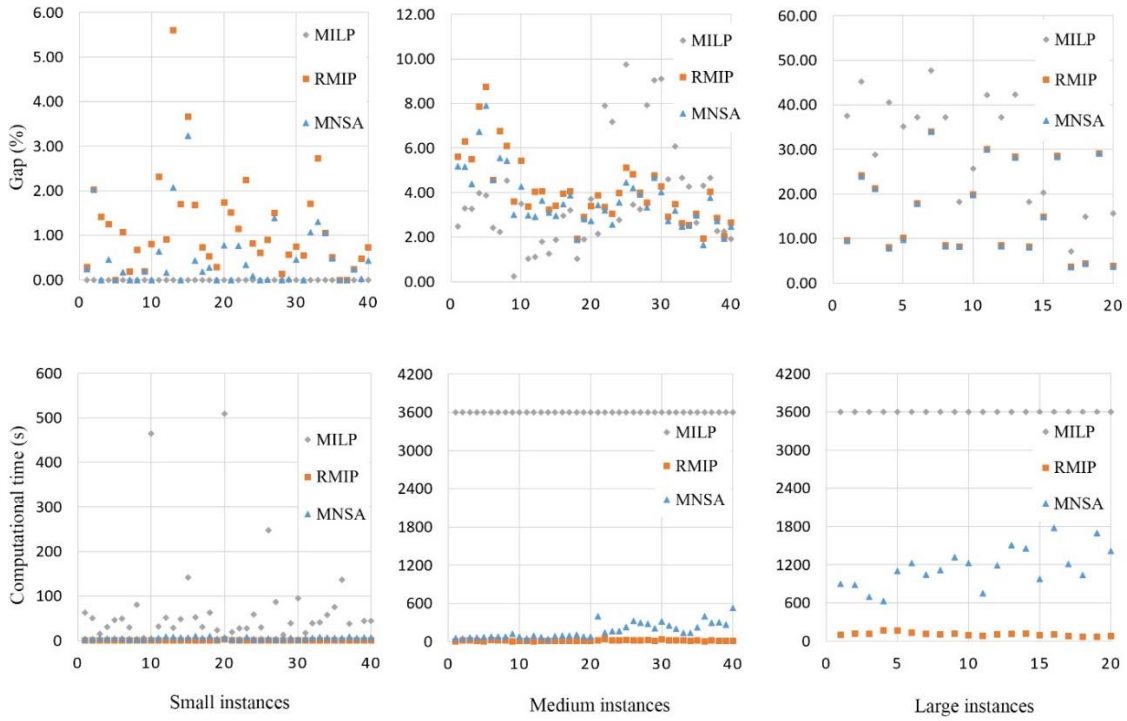

Figure 3.6.1: The performance of the algorithms MILP, RMIP, and MNSA.

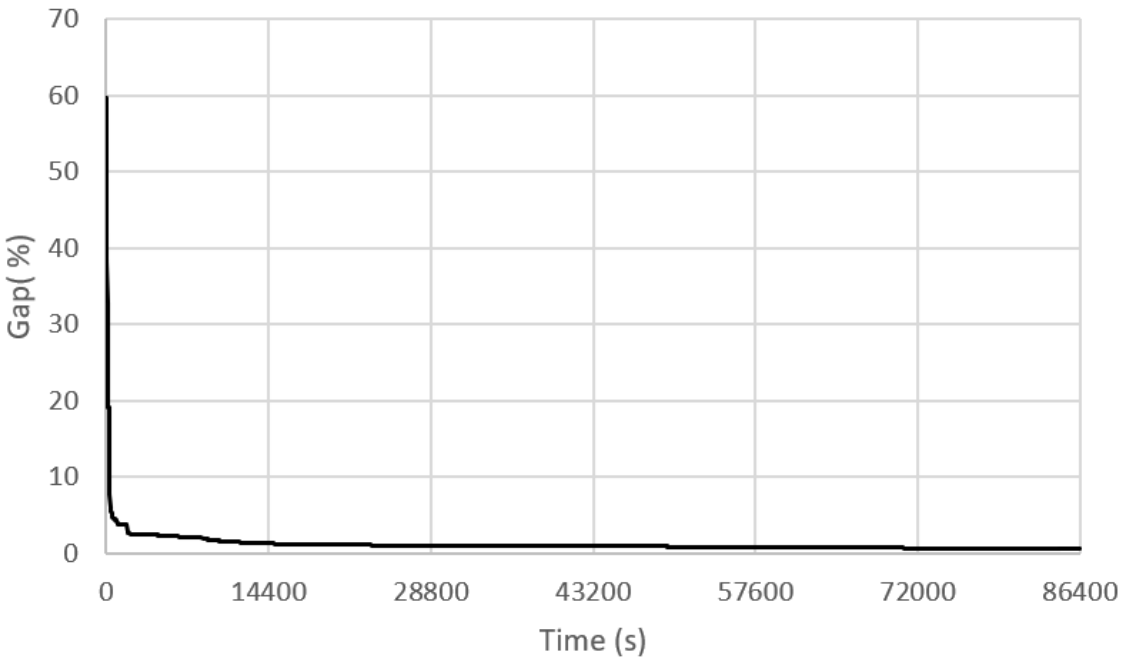

Figure 3.6.2: The gaps of running MILP formulation for twenty-four hours

Table 3.6.3: Gaps and computational time for the small-scale instances

	# of opt.	Max. gap (%)	Ave. gap (%)	Min. Gap (%)	Ave. Time (s)
MILP	40	0.00	0.00	0.00	85.88
RMIP	3	5.60	1.13	0.00	0.59
MNSA	13	3.24	0.46	0.00	5.54

of a minute and a half. By contrast, RMIP algorithm performs worse on solution quality but has a great advantage on computational efficiency. It takes only one second to obtain near-optimal solutions. The maximum gap and average gap are 5.60% and 1.13%, respectively. MNSA algorithm is a compromise between the MILP and RMIP. Its computational time is shorter than that of MILP, and its solution quality is better than that of RMIP. As is shown in the last row, the MNSA solved thirteen instances to optimal in six seconds. Its maximum gap and average gap are 3.24% and 0.46%, respectively.

Table 3.6.4 shows more details with regard to the computational results. The smallest objective function value, the shortest computational time, and the smallest gap are bolded. Note that each of the rows from S1 to S4 represents an average result of ten instances, rather than one single instance. For example, the value 4409.0 in the row of S1 represents the average objective function value for instances through the first instance of S1 to the last instance of S1. The MILP algorithm achieves the best performance on solution quality, and the RMIP algorithm consumes the shortest time to obtain near-optimal solutions. MNSA algorithm falls between the above two algorithms.

Table 3.6.5 compares the three algorithms in terms of medium-scale synthetic datasets. The MILP algorithm terminates when the time limitation of 3600 seconds is reached, and the solver provides us the best known lower bound for comparison. The results show that the MILP formulation performs best on solving the first four datasets. However, with the increase of the problem scale, the performance of RMIP algorithm and MNSA algorithm gradually exceeds that of MILP. For dataset M5 to M8, RMIP algorithm obtained better solutions. This is because the MILP formulation terminates with a relatively big gap due to the time limitation, while the computational time is sufficient for RMIP algorithm. The MNSA algorithm further improves the RMIP solutions within a reasonable time. As is shown, the average gap is reduced from 4.09% to 3.66% in three minutes.

Table 3.6.6 shows the computational results with regard to large-scale synthetic instances. The MILP algorithm terminates when the time limitation of 3600 seconds is reached. As is shown, the MNSA algorithm dominates MILP algorithms on all the instances.

Table 3.6.4: Comparison of the three algorithms in terms of small-scale datasets

Data	MILP				RMIP			MNSA		
	Global Optimum	Obj.	Time (s)	Gap† (%)	Obj.	Time (s)	Gap† (%)	Obj.	Time (s)	Gap† (%)
S1	4409.0	4409.0	83.50	0.00	4444.0	0.49	0.79	4422.5	4.13	0.31
S2	4125.0	4125.0	98.26	0.00	4207.3	0.82	1.91	4158.2	7.27	0.78
S3	8220.4	8220.4	64.54	0.00	8305.4	0.49	1.01	8245.6	4.10	0.31
S4	7584.7	7584.7	97.21	0.00	7646.3	0.54	0.80	7620.0	6.65	0.46
Ave.	6084.8	6084.8	85.88	0.00	6150.7	0.59	1.13	6111.6	5.54	0.46

† Gap to global optimum

Table 3.6.5: Comparison of the three algorithms in terms of medium-scale datasets

Data	MILP			RMIP			MNSA		
	Lower Bound	Obj.	Gap‡ (%)	Obj.	Time (s)	Gap‡ (%)	Obj.	Time (s)	Gap‡ (%)
M1	8631.5	8932.6	3.38	9261.9	13	6.80	9170.3	57	5.87
M2	7645.2	7844.7	2.58	8070.6	15	5.29	8008.7	85	4.56
M3	16261.0	16494.5	1.41	16872.5	8	3.63	16785.3	66	3.12
M4	16036.6	16458.7	2.56	16575.5	9	3.25	16526.1	87	2.96
M5	16510.8	17562.0	5.95	17175.8	24	3.87	17099.2	220	3.44
M6	16275.7	17421.8	6.55	17003.2	22	4.27	16958.4	285	4.02
M7	28259.5	29580.3	4.46	29110.9	19	2.92	29069.0	189	2.78
M8	27890.9	28783.4	3.09	28668.1	12	2.70	28612.5	357	2.52
Ave.	17188.9	17884.8	3.75	17842.3	15	4.09	17778.7	168	3.66

‡ Gap to best known lower bound found by solver

Table 3.6.6: Comparison of the three algorithms in terms of large-scale datasets

	MILP			RMIP			MNSA		
Data	Lower Bound	Obj.	Gap‡ (%)	Obj.	Time (s)	Gap‡ (%)	Obj.	Time (s)	Gap‡ (%)
L1	21158.3	34063.8	37.46	24792.6	134	14.64	24731.4	842	14.42
L2	19762.1	29924.7	33.21	24023.9	112	17.69	23992.9	1184	17.58
L3	22569.6	34361.0	32.06	28151.1	103	17.97	28104.6	1179	17.84
L4	37890.1	50502.3	24.19	44006.0	80	13.93	43934.9	1425	13.79
Ave.	25345.0	37212.9	31.73	30243.4	107	16.06	30191.0	1157	15.91

‡ Gap to best known lower bound found by solver

The average gap is 31.73% for the MILP algorithm while that is only 15.91% for the MNSA algorithm. Just like the results in the previous two tables, the RMIP algorithm is of slightly worse solution quality than the MNSA algorithm, but it is the fastest algorithm. Its average computational time is less than two minutes.

In summary, the MILP algorithm is only suitable for solving small-scale instances. It is able to obtain optimal solutions in a reasonable time. However, with the increase of instance scale, the MILP algorithm becomes more and more inefficient. The RMIP algorithm and MNSA algorithm gradually outperform MILP algorithm. This is because MILP is terminated in a relatively early stage due to the time limitation, but the computational time is sufficient for the other two algorithms. Since the MNSA algorithm is based on the RMIP algorithm, it consumes more time than the RMIP algorithm, but its solution quality is slightly better.

3.6.3 Real-life Instances

The performances of MILP, RMIP, and MNSA algorithms are also verified in twelve real-life instances. The amount of demand fluctuates from week to week. The consultants consist of two types: regular consultants (RC) and contract consultants (CC). The RCs are salary-based employees. They get paid whether they are assigned demand or not. Thus, there is no need to include the RCs' wages in the objective function. The CCs only get paid when they work in the week. Generally speaking, RC is always the first-tier to be assigned and CC is only assigned demand if needed. The maximum number of trip legs allowed is four per week, which includes the last trip home.

As is shown in Table 3.6.7 , there are 63 consultants in total, which consists of 29 RCs and 34 CCs. The wages for the RCs are set to zero because they are fixed and thus not counted into the objective function. The average wage of CCs is $2,162 dollars per week. All the consultants work two shifts a day, five days a week. The skillset includes 18 types of skills in total. On average, RC is proficient in 13 different skills, and CC has about 12. There are two kinds of demands: the regular demand and fixed demand. The fixed demand can only be served by designated consultants.

Table 3.6.7: Characteristics of regular consultant and contract consultant

	Regular Consultant	Contract Consultant
Number of consultants	29	34
Average wage ($)	0	2162
Number of available shifts	10	10
Average number of skills	13	12

Table 3.7.1 shows the number of fixed demand and regular demand among the twelve scheduling weeks. On average, the total weekly demands are 464, which includes 433 regular demands, and 32 fixed demands.

Table 3.7.2 shows the computational results with regard to the twelve real-life instances. The MILP algorithm terminates at 3600 seconds. The RMIP algorithm and MNSA algorithm are superior to the MILP algorithm in eleven instances. Especially for week 4, the solution gap for the MILP algorithm is as large as 35.31% while that is only 3.68% for RMIP and 3.63% for MNSA. The only exception is week 6, where the gap of MILP is 2.47% while that is 3.14% for RMIP and 3.13% for MNSA. On average, the performances of the RMIP algorithm and MNSA algorithm are also significantly better than that of the MILP algorithm. However, the differences in solution quality between the MNSA algorithm and RMIP algorithm is quite small. MNSA algorithm reduced the average gap of RMIP solutions from 4.31% to 4.24% in approximately five minutes.

3.7 Conclusion

The paper studies a variant of the consultant assignment and routing problem, which takes priority matching into consideration. The new constraint does not change the problem much, but it increases the computational complexity. The existing MILP formulation was found inefficient to solve the problem. Therefore, we present a decomposition algorithm named RMIP and a metaheuristic named MNSA in this paper. The RMIP algorithm first obtains feasible an allocation plan for the consultants in the first stage. Then, the best traveling routes under that allocation plan are determined by the exhaustive method. The MNSA algorithm applies a neighborhood search algorithm to further optimize the solutions. The performances of the presented algorithms are evaluated by comparing them with the modified MILP formulation.

Computational results show that the MILP formulation performs well in solving small-scale instances. It finds optimal solutions for all synthetic instances. RMIP and MNSA algorithms obtain near-optimal solutions with a gap of 1.1% and 0.5% respectively. With

Table 3.7.1: Number of flexible demand and fixed demand

Week	1	2	3	4	5	6	7	8	9	10	11	12	Ave.
# of Total demand	453	483	446	470	438	490	477	449	446	473	450	497	464
# of Regular demand	419	457	414	435	411	459	443	413	414	443	423	463	433
# of Fixed demand	34	26	32	35	27	31	34	36	32	30	27	34	32

Table 3.7.2: Comparison of the three algorithms in terms of real-life instances

Data	MILP			RMIP			MNSA		
	Lower Bound	Obj.	Gap‡ (%)	Obj.	Time (s)	Gap‡ (%)	Obj.	Time (s)	Gap‡ (%)
wk.1	41127.3	43607.9	5.69	42977.2	166	4.30	42977.2	399	4.30
wk.2	46406.8	49930.8	7.06	49353.2	331	5.97	49353.2	602	5.97
wk.3	39205.7	43810.4	10.51	40959.5	91	4.28	40958.2	322	4.28
wk.4	42289.9	65376.3	35.31	43906.9	512	3.68	43884.9	806	3.63
wk.5	36654.0	38478.6	4.74	38345.8	287	4.41	38283.0	568	4.26
wk.6	49125.3	50371.5	2.47	50719.0	102	3.14	50711.7	443	3.13
wk.7	36771.1	38605.2	4.75	38130.2	133	3.56	38116.9	401	3.53
wk.8	29453.9	32205.0	8.54	31046.6	561	5.13	30951.4	1182	4.84
wk.9	31750.5	34330.0	7.51	32880.3	72	3.44	32879.0	281	3.43
wk.10	34102.9	36879.2	7.53	36390.9	634	6.29	36390.9	942	6.29
wk.11	30319.8	32281.0	6.08	31622.9	333	4.12	31571.9	601	3.97
wk.12	50891.4	53716.9	5.26	52695.9	471	3.42	52612.0	742	3.27
Ave.	39008.2	43299.4	8.79	40752.4	308	4.31	40724.2	607	4.24

‡ Gap to best known lower bound found by solver

the increase of the problem size, solving MILP formulation becomes more and more time-consuming and the performance of the other two algorithms gradually exceeds that of MILP. For large-scale instances, the RMIP algorithm significantly outperforms MILP. However, the improvement achieved by the MNSA algorithm is not significant. The testing of real-life instances produces similar results. The presented two algorithms outperform MILP formulation in eleven of the twelve instances.

Future research may include but are not limited to the following aspects. First, future studies may take more realistic factors into consideration. For example, the price of a flight ticket may change during a week. In addition, the price may also be asymmetric for departing and back. Considering the price change in the planning horizon would make the model more accurate. Furthermore, the improvement of solution quality brought from the MNSA algorithm is not significant when the problem scale increased. A more efficient heuristic may be designed to improve the solution.

Chapter 4. A Supervised Learning Approach for Solving the Traveling Salesman Problems

This chapter studies the Traveling Salesman Problem (TSP), which is the most basic version of vehicle routing problems. Recently, many learning-based approaches are proposed to solve TSP. However, the scalability and generalization of these approaches remain significant challenges. This paper addresses this gap by introducing a supervised learning approach that leverages local information to make predictions. In particular, we introduce the concept of "anchors," which represents nodes that should be connected to their nearest neighbors in the optimal solution. Our approach differs from the previous supervised learning approaches in that, instead of inputting the global distance information, it solely relies on the surrounding nodes to make predictions, which enables it to handle large-scale instances without sacrificing prediction accuracy. Experimental results demonstrate that our model successfully identifies 87% of the anchors with a precision of over 95% for both generated and TSPLIB instances. By integrating the predicted anchors into established methods such as the Miller-Tucker-Zemlin (MTZ) model and insertion algorithms, we achieve significant improvements in solution quality, reducing the average gap. This work showcases the scalability and adaptability of our proposed learning approach for solving TSPs.

4.1 Introduction

The Traveling Salesman Problem (TSP) is a well-known combinatorial optimization problem that seeks to find the shortest route visiting a set of locations and returning to the starting point. Various approaches have been developed to solve TSP, including exact methods, heuristics, and metaheuristics, each with their own strengths and limitations. Solver based on exact methods, such as the Concorde, can find optimal solutions but are computationally intensive, particularly for large-scale instances (Applegate et al., 2002). Heuristic algorithms, such as the nearest neighbor algorithm, insertion algorithms, and 2-opt algorithm are faster but may not guarantee optimality. State-of-the-art heuristics like Lin-Kernighan-Helsgaun (LKH) algorithm offer a balance between efficiency and solution quality, making them popular choices for TSP (Helsgaun, 2000; Lin & Kernighan, 1973).

Recent research has explored learning-based algorithms for combinational optimization (Bengio et al., 2021). In many scenarios, it is common to repeatedly solve TSPs with different data while maintaining the same problem structure. This repetitive nature presents an opportunity to develop machine learning (ML) algorithms that can leverage the underlying problem pattern. By learning from a large amount of historical optimal routes, ML can effectively exploit the pattern behind the optimal solutions which can be used to improve the existing algorithms. However, generalization to larger-scale instances remains a challenge. Scaling up training data can be costly, limiting the scalability and adaptivity of ML methods.

In this paper, we propose a novel supervised learning approach that overcomes these challenges. Our approach trains on small-scale generated instances and exhibits remarkable scalability to large-scale instances as well as adaptivity to heterogeneous instances, all while maintaining high prediction accuracy. The proposed approach differs from the traditional ML methods by the fact that we utilize local information instead of global information for making predictions, which allows our approach to a better generalization ability. To simplify the explanation, we introduce a new term called "anchors," which represents the nodes that should connect to their nearest neighbors in an optimal solution. By using ML, we train a neural network to predict the anchors based on spatial information.

The contribution of this paper is twofold. Firstly, we propose a supervised learning framework that trains on small-scale instances but generalizes well to large-scale instances. Empirical results demonstrate that the trained model successfully predicts 87% of the anchors with a precision of more than 95% for both generated instances and TSPLIB instances. Notably, the trained model achieves comparable performance in predicting TSPLIB instances, despite these instances not being included in the training data. Secondly, we provide practical ways for incorporating the anchor concept into existing algorithms. With the predicted anchors, we can significantly reduce the computational time required for solving the Miller-Tucker-Zemlin (MTZ) formulation using Gurobi. The proposed approach also achieves a good integration with heuristics. Moreover, the paper presents a

comprehensive analysis of the trained model, discusses various evaluation metrics, and explores a trade-off between precision and recall scores.

The remaining sections of the paper are organized as follows: Section 4.2 provides a comprehensive review of existing learning-based approaches for the Traveling Salesman Problem (TSP), with a particular focus on supervised learning methods. Section 4.3 introduces the concept of anchors and explores their potential to enhance traditional heuristics for TSP. Section 4.4 outlines our proposed approach in detail, including feature selection, data preprocessing, hyperparameter tuning, and the integration of anchors with second-stage optimization algorithms. Section 4.5 presents a comprehensive performance evaluation of the trained model. Section 4.6 demonstrates the effectiveness of utilizing anchors with exact methods and heuristics on the generated instances and TSPLIB instances. Section 4.7 discusses the findings and practical insights gained from our study. Finally, we summarize the paper and suggest directions for future research in Section 4.8.

4.2 Literature Review

The TSP has attracted significant research attention as a well-known optimization problem. Various methodologies, including constructive heuristics, exact methods, and metaheuristics, have been developed to address this problem. State-of-the-art solvers, such as the Concorde solver, have demonstrated efficient solutions for the TSP. The Lin-Kernighan-Helsgaun (LKH) algorithm has also been successful in finding near-optimal solutions for large-scale instances. Traditional methodologies on solving TSP can be found in the literature review by Laporte (1992). Recent research has explored the use of machine learning (ML) and reinforcement learning (RL) approaches for combinatorial optimization. Comprehensive reviews are available in Karimi-Mamaghan et al. (2022) and Mazyavkina et al. (2021).

In the early stages of applying ML to solve the TSP, one commonly used approach was the pointer network, which is a sequence-to-sequence neural network introduced by Vinyals et al. (2015). This technique utilizes attention mechanisms to select elements from

the input sequence as outputs. Kool et al. (2018) extended this approach by training the model using the REINFORCE algorithm. Through their work, they were able to obtain near-optimal solutions for TSP instances with up to 100 nodes. Building upon the pointer network framework, Stohy et al. (2021) proposed a hybrid graph pointer network that combines a graph embedding layer with the encoder of a transformer model. Expanding the application of attention-based pointer networks, Kong et al. (2022) proposed an attention-based pointer network to solve drone logistic delivery problems. further considered multiple objectives for TSP. Furthermore, Perera et al. (2023) extended the problem by considering multiple objectives.

RL has gained significant popularity in the field due to its ability to generate decision-making policies for complex optimization problems. Several studies have successfully applied RL to address TSP. For instance, Dai et al. (2017) and Kwon et al. (2020) proposed to solve TSP using RL. Nazari et al. (2018) introduced an end-to-end framework that utilized RL for solving vehicle routing problems. Wu et al. (2022) employed deep RL to learn a heuristic that improves initial solutions for TSP. Furthermore, Zhang et al. (2023) and (Ling et al., 2023) utilized deep RL to solve TSP. In recent years, there has been a growing interest in drone-based TSP (Macrina et al., 2020). Ha et al. (2018) and Tiniç et al. (2023) specifically addressed TSP with multiple drones. Salama & Srinivas (2020) explored the use of customer location clustering to improve last-mile delivery in the context of TSP with drones. The application of RL has shown promising results in solving drone-aided routing problems, as demonstrated by the work of Bogyrbayeva et al. (2023). Liu et al. (2022) addressed the challenge of stochastic travel times in TSP with drones, highlighting the potential of RL in this field.

ML techniques have also been used to enhance existing TSP algorithms. Researchers have explored the integration of ML with the LKH algorithm, a state-of-the-art TSP heuristic, to improve its performance. Zheng et al. (2021, 2023) introduced the Variable Strategy Reinforced LKH (VSR-LKH) algorithm, which combines reinforcement learning with the LKH algorithm. Similarly, Xin et al. (2021) proposed the Neuro-LKH algorithm, which incorporates deep learning with the LKH algorithm. These approaches leverage ML

techniques to enhance the effectiveness and efficiency of the LKH algorithm in solving the TSP.

In this paper, our primary focus is on supervised learning (SL) approaches, as our proposed method falls within this category. SL approaches have received less attention compared to other learning-based approaches, primarily due to the challenges they face in generalizing to larger problem instances. Li et al. (2018) proposed a learning-based approach that combines deep learning with tree search. They employed a graph convolutional network to estimate the likelihood of each vertex being part of the optimal solution. ML can reduce the problem size of TSP by predicting which arcs should be included in the optimal solution. This approach acts as a preprocessing technique by fixing a subset of decision variables for a solver to solve the mathematical formulation. However, solving the simplified model can still be computationally expensive. Sun et al. (2021) introduced a problem-reduction technique that utilizes a support vector machine to predict optimal edges. Their study found that when the training and test instances belong to the same TSP variant, the model exhibits a small error. However, as the model is tested on TSP variants that differ significantly from the training data, the generalization error naturally increases. Mele et al. (2021) employed ML to predict edges likely to be part of the optimal solution in the TSP. They combined this partial solution with a constructive heuristic to generate the feasible solutions. The computational complexity for solution construction was shown to be $O(n^2 \log n^2)$ where n is the number of cities.

One critical limitation of SL-based approaches is the requirement to have a fixed input size, which limits their adaptability and scalability. Joshi et al. (2022) specifically addressed this issue by noting that while state-of-the-art learning-driven approaches for TSP demonstrate strong performance when trained on small instances, they struggle to generalize their learned policies to larger instances at practical scales. As a result, they emphasized the necessity of reevaluating various aspects to achieve effective generalization beyond the training data. To overcome this generalization limitation, Fu et al., (2021) proposed a solution by training a small-scale model in a supervised manner. This trained model can be repeatedly applied to generate heat maps for TSP instances of

various sizes. The approach incorporates techniques such as graph sampling, graph converting, and heat map merging to ensure scalability and achieve the desired level of generalization.

As mentioned above, various attempts have been made to improve the scalability and adaptivity of ML methods for solving TSP. However, to the best of our knowledge, none of the existing supervised learning techniques have demonstrated the ability to generalize and maintain accuracy when solving large-scale instances. This paper fills this gap by introducing a new supervised learning approach. The novelty of our approach lies primarily in our unique training methodology. Unlike previous approaches that focus on predicting individual arcs in the optimal solution, we predict "anchors" which represent the nodes that connect to their nearest neighbors in the optimal solution. This shift allows the model to make predictions based on local information, i.e., the surrounding neighbors of a node, greatly enhancing the generalization ability of the neural network. Furthermore, empirical experiments illustrate that the trained model can effectively solve TSP instances with significantly different distributions, further highlighting its robustness and versatility.

4.3 Definition of anchors

To facilitate the subsequent discussion, we introduce the term "anchor" to refer to nodes that should be connected to their nearest neighbors in the optimal solution. In this section, we present an illustrative example to demonstrate the necessity of employing anchors and their role in facilitating the generation of near-optimal solutions. Additionally, we conduct experiments to showcase that, in general, a significant proportion, approximately 80% to 90%, of the nodes in TSP problems can serve as anchors. Although we cannot provide mathematical proofs, we believe that the ratio of anchors does not decrease with the increase of problem scales. This assumption implies that as the size of TSP problems increases, the relative number of nodes acting as anchors remains relatively constant.

Figure 4.3.1 shows two solutions for a TSP instance comprising six nodes. As presented on the left side, one solution is generated by using the nearest neighbor algorithm, resulting

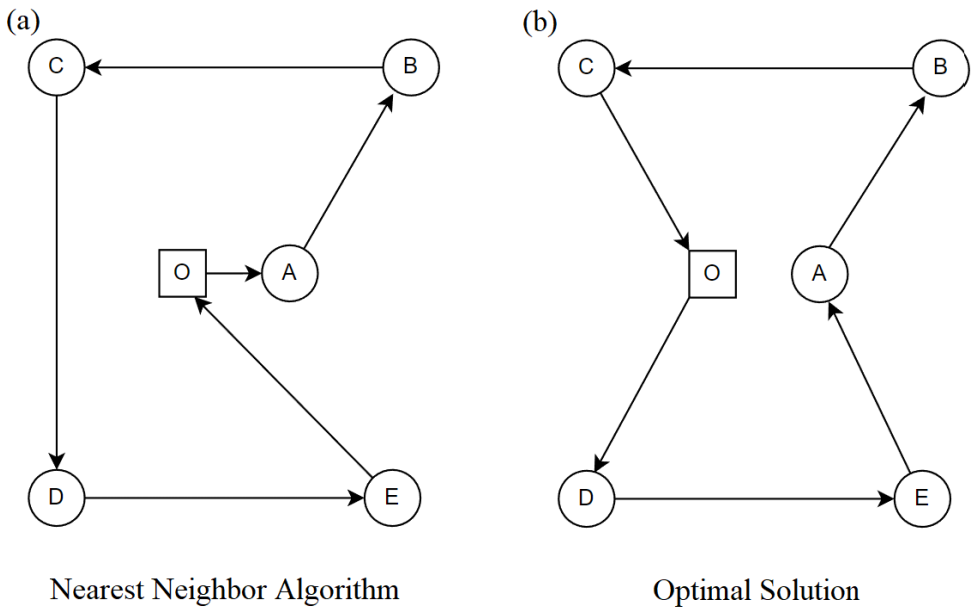

Figure 4.3.1: A Demonstration of Solutions

in a total route distance of 16.06. As displayed on the right side, the optimal solution is obtained by Gurobi. In contrast, the optimal total route distance is 14.94. It is widely recognized that heuristics, such as the nearest neighbor algorithm, offer the advantage of quickly solving TSP problems. However, the solution quality is typically far away from optimality. By comparing the two solutions above, it becomes apparent that the starting city, denoted as O in the example, should not be connected to its nearest neighbor, denoted as node A. This observation demonstrates the limitations of the nearest neighbor algorithm.

In order to overcome the limitations of traditional constructive heuristics in solving the TSP, we propose to develop a predictive model that can determine which nodes should connect to their nearest neighbors in advance. These nodes are denoted as anchors in the subsequent discussions. By incorporating the predicted anchor nodes, we can enhance traditional approaches, preserving their advantages in terms of computational efficiency while mitigating their shortcomings in solution quality. While we have used the nearest neighbor algorithm as an example, other constructive algorithms such as insertion algorithms and 2-opt algorithm can also be implemented alongside anchors to improve their performance.

Figure 4.3.2 demonstrates the utilization of anchors in solving a TSP instance. The process entails three steps: anchor identification, connecting anchors to their nearest neighbors, and connecting the remaining nodes using a constructive algorithm. In the provided example, our proposed supervised learning model predicts four anchor nodes, highlighted in red. As depicted in subfigure (b), these anchors are then connected to their nearest neighbors. Finally, the remaining nodes are connected to their respective nearest neighbors, resulting in the complete solution shown in subfigure (c). It is found that the solution generated from these three steps is the same as the optimal solution obtained by Gurobi. This highlights the potential of enhancing traditional approaches through the prediction of anchors.

Figure 4.3.3 presents a box chart illustrating the percentage of anchors for various scales of the Traveling Salesperson Problem (TSP). The data consists of 500 instances, and each

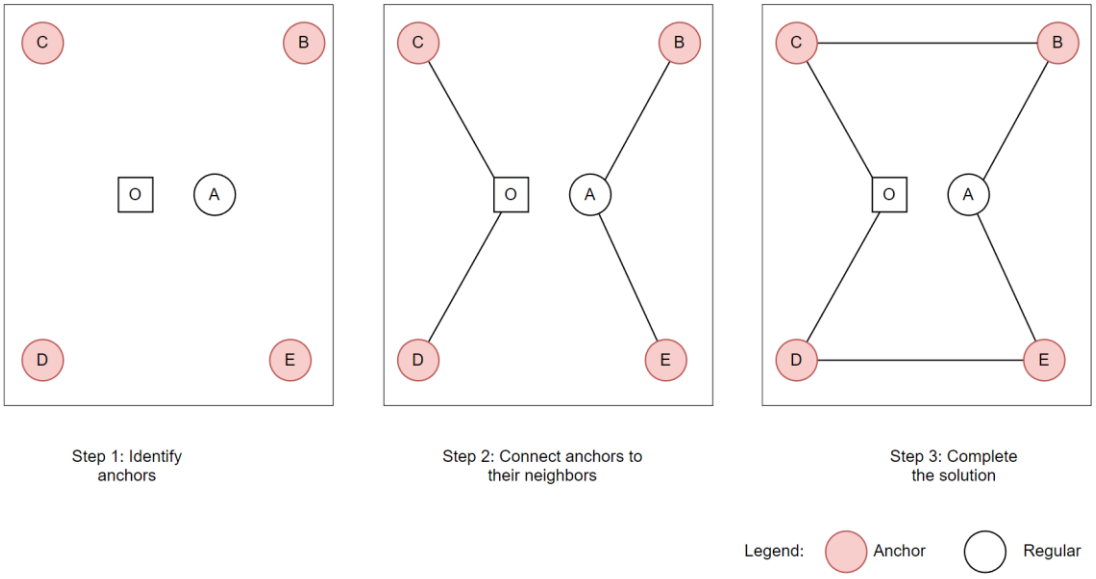

Figure 4.3.2: Framework of an Anchor-based Constructive Heuristic

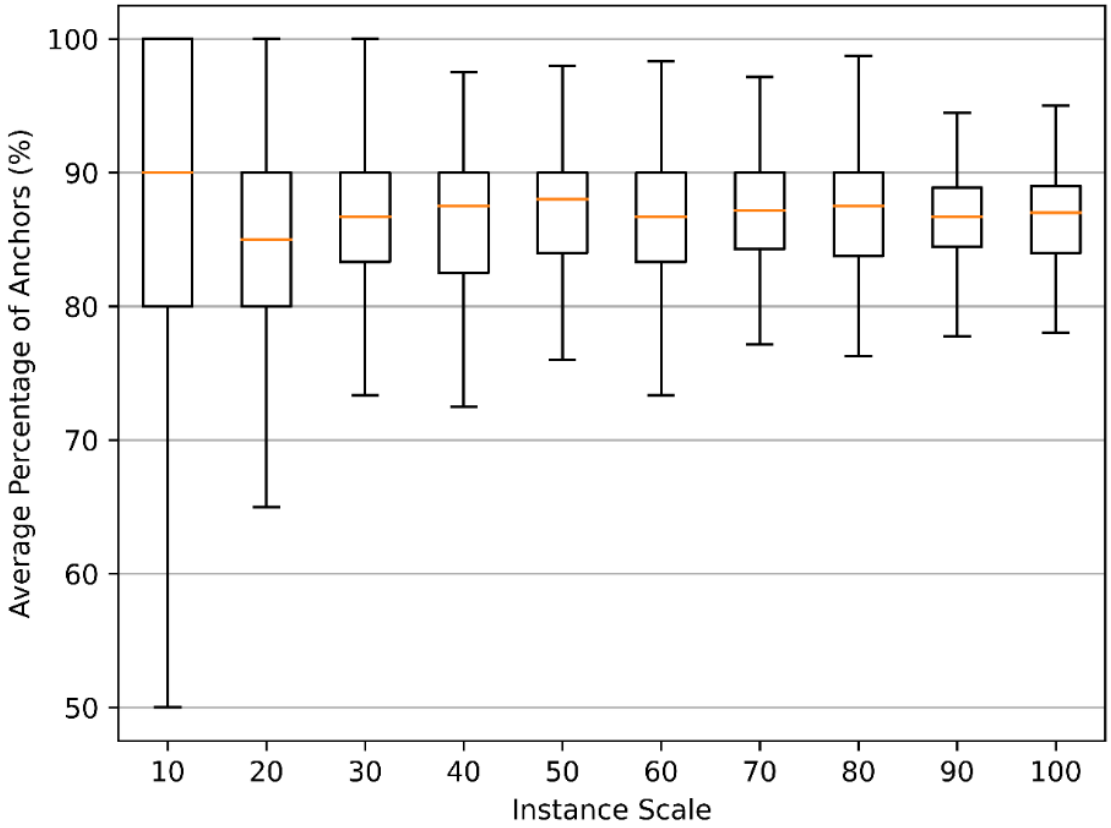

Figure 4.3.3. Average Percentage of Anchors for general instances of TSP

box in the chart represents the average percentage of anchors calculated from these instances. The TSP instances were solved using the Concorde solver.

4.4 Methodology

The methodology section consists of five parts: feature selection, data preprocessing, hyperparameter tuning, anchor insertion algorithm, and anchor-MTZ formulation. The first three parts introduce the training process of the supervised learning model. The last two parts present the pseudo code of the proposed algorithm and the model that integrates the predicted anchors with the MTZ model.

4.4.1 Feature Selection

Our study proposes a novel approach for training a supervised learning model to solve TSP. We define the feature as an $m \times m$ matrix, where m represents the dimension of neighbors. Specifically, we consider m neighbors of a node and the m neighbors of each of those nodes as inputs to construct the feature matrix. We propose to use this feature definition based on our observation of a hidden pattern that can be learned to determine whether a node should be classified as an anchor or not, using only its local information.

One intuitive insight is that if a node has similar distances to its neighbors, it is less likely to be an anchor. Instead of connecting to the nearest neighbor, the node can connect to the second-nearest one or even other nodes without a significant loss. By contrast, if the distance to the nearest neighbor and the second-nearest neighbor differs significantly, the node is more likely to be an anchor. It is difficult to describe the underlying patterns by mathematical formulations. Therefore, we employ a supervised learning model to approximate the relationship and predict whether a node should be classified as an anchor.

To approximate the relationships between features and labels, we propose training a multiple layer perceptron (MLP). An essential hyperparameter that needs to be determined is the dimension of the feature matrix. Generally, a larger dimension allows for the utilization of more information in the prediction process. However, larger networks can be more difficult to train and may require a larger dataset for effective training. To determine

the optimal dimension, we conducted several preliminary experiments and tested different dimensions ranging from 3 to 10. The results are summarized in Table 4.4.1.

The trend indicates that as the dimension of the feature matrix increases, the average accuracy and precision of the model decrease. Consequently, selecting a large value for the dimension is not ideal. In order to achieve a balance between the precision and specification, we have chosen a dimension of 5 for the feature matrix.

4.4.2 Data Preprocessing

Data preprocessing plays a vital role in enhancing the generalization ability of the model. Among the critical preprocessing techniques, normalization is of utmost importance, especially when instances exhibit varying scales. To address this, we employ min-max normalization, which ensures that each feature is within the same range, specifically between zero and one. The formulation for min-max normalization is as follows,

$$x_{normal} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

where x represents the feature matrix, x_{min} and x_{max} denote the minimum and maximum values within the matrix, and x_{norm} represents the normalized feature. By applying this approach, we have achieved improved performance and more robust predictions. This normalization technique ensures that the data is comparable across different instances, enabling better generalization ability of the model.

4.4.3 Hyperparameter Tuning

In this study, we utilize a random search policy to fine-tune the hyperparameters. The ranges for tuning are presented in Table 4.4.2. It should be noted that the number of epochs is not considered as a hyperparameter. Instead, we set a maximum of 200 training epochs and implement an early stopping mechanism if the accuracy does not improve for five consecutive epochs. This approach enables us to efficiently optimize our model while mitigating the risk of overfitting.

Table 4.4.1: The relationship between input size and prediction performance

	Accuracy	Precision	Recall	Specific	F1 Score	Time (s)
3	0.877	0.886	0.975	0.347	0.928	72
4	0.872	0.881	0.975	0.327	0.925	116
5	0.860	0.880	0.962	0.377	0.918	107
6	0.852	0.869	0.968	0.316	0.914	83
7	0.857	0.869	0.973	0.307	0.916	133
8	0.855	0.872	0.967	0.331	0.915	127
9	0.852	0.873	0.962	0.342	0.913	76
10	0.851	0.860	0.980	0.242	0.914	115

Table 4.4.2: The range of hyperparameters for random search

Hyperparameters	**Range**
Learning rate	[0.001, 0.01, 0.1]
Number of hidden layers	[1, 2, 3, 4]
Number of neurons in each layer	[64, 128, 256, 512]
Activation function	[ReLU, Sigmoid]
Optimizer	[Adam, RMSProp]
Dropout rate	[0, 0.1, 0.2]
Batch size	[128, 256, 512, 1024]

To address the challenge of excessively long training time, we generated a smaller dataset specifically for hyperparameter tuning purposes. The dataset consists of 405,198 feature and label pairs generated from various TSP instances, including TSP25, TSP50, TSP100, as well as instances from TSPLIB and National Traveling Salesman Problems. The results are summarized in Table 4.4.3.

The experiments revealed that a learning rate of 0.001, in combination with the Adam optimizer, consistently yielded good performance across most cases. It was found that the number of layers and the number of neurons did not have a significant impact on the performance. Given that the actual training dataset is much larger than the dataset used for parameter tuning, we were able to use a relatively large number of layers and neurons without compromising performance. For the activation function, we selected ReLU, which has shown to be effective in various tasks. To mitigate the risk of overfitting, we set the dropout rate to 0.1. To expedite the training process and enhance efficiency, we set the batch size to 1024. The selected hyperparameters, as summarized in Table 4.4.4, allowed us to achieve satisfactory performance while reducing the training time, resulting in a more efficient process.

4.4.4 Anchor Insertion Algorithm

In this section, we present the proposed anchor insertion algorithm, which is inspired by the nearest insertion algorithm. The pseudo code for this algorithm is provided in Algorithm 1.

The anchor insertion algorithm operates based on a straightforward concept. Initially, a supervised learning technique is employed to predict a set of anchors, denoted as A. Subsequently, the algorithm proceeds by randomly selecting a node a from the set A. This extracted node a is then inserted into the current solution, x, at a location determined by the $BestInsertIndex$ function. The function calculates the index i that represents the optimal position for inserting node a into the incompletely solved solution. Following this step, the algorithm checks whether the nearest neighbor of the anchors, denoted as node b, has already been visited. If node b has not been visited, it is inserted into the solution at the

Table 4.4.3: Top 10 trained model during the hyperparameters tuning

Learn Rate	# of Layer	# of Neuron	Activate	Opt.	Drop Rate	Batch	Acc.	Epoch	Time (s)
0.001	2	256	ReLU	Adam	0.2	1024	0.898	47	157
0.001	1	128	ReLU	Adam	0.1	512	0.897	71	135
0.01	4	256	Sigmoid	Adam	0	512	0.896	37	337
0.01	3	128	Sigmoid	Adam	0.1	128	0.896	38	340
0.001	1	64	ReLU	Adam	0	256	0.896	48	67
0.001	4	512	ReLU	Adam	0	128	0.895	21	563
0.001	2	512	Sigmoid	Adam	0.2	512	0.895	148	1040
0.001	1	512	ReLU	Adam	0.1	1024	0.895	48	111
0.01	3	512	Sigmoid	Adam	0	512	0.895	19	183
0.001	1	64	ReLU	Adam	0.1	512	0.895	56	54

Table 4.4.4: Hyper-parameters selected after fine tuning

learning rate	layers	neurons	activation	optimizer	dropout	batch
0.001	3	256	ReLU	Adam	0.1	1024

Algorithm 1 $Anchor\ Insertion$

Inputs: distance matrix d, threshold σ

Output: route x

```
get the set of anchors A ← SupervisedLearning(d, σ)
create a set of unvisited nodes U
create an empty route x
while A is not empty do
    a ← RandomAnchor(A)
    i ← BestInsertionIndex(a)
    Insert node a into route x at position i
    remove node a from set A and set U
    b ← NearestNeighbor(a)
    if b ∉ U do
        j ← BestInsertionIndex(b)
        insert node b into route x at position j
        remove node b from set A and set U
while U is not empty do
    c ← RandomNode(U)
    k ← BestInsertionIndex(c)
    insert node c into route x at position k
    remove node c from U
return route x
```

most suitable location. Again, the optimal location is determined by the $BestInsertIndex$ function. Finally, the algorithm handles the remaining nodes using similar approaches. It examines the remaining nodes, determining their optimal insertion locations, and inserts them into the solution accordingly. The output of the algorithm is a fully completed route.

In summary, the anchor insertion algorithm efficiently constructs a solution by employing a prioritized approach to insert nodes into an incomplete solution, utilizing supervised learning. The algorithm begins by predicting the anchors, and subsequently inserts both the anchors and their nearest neighbors into the solution. The insertion locations are determined based on minimizing the increment in route length. Then, the algorithm deals with the remaining unvisited nodes by employing similar insertion procedures. This approach emphasizes the critical role of anchors in constructing an effective solution for TSP.

4.4.5 Anchor-MTZ Algorithm

The predicted anchors can be employed to enhance the existing mathematical models for solving TSP. In this paper, we propose an integration of the predicted anchors with the MTZ formulation. Although we also tested the Dantzig-Fulkerson-Johnson (DFJ) model, preliminary results show that the solvable scale is too small without considering additional enhancements. The formulation for anchor-MTZ is as follows.

$$minimize \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij}$$

$$\sum_{j \in N, j \neq i} x_{ij} = 1, \forall i \in N \tag{4.4.1}$$

$$\sum_{i \in N, i \neq j}^{n} x_{ij} = 1, \forall j \in N \tag{4.4.2}$$

$$u_i - u_j + n x_{ij} \leq n - 1, \forall\, i \in N, j \in N_0, i \neq j \tag{4.4.3}$$

$$x_{ij} + x_{ji} = 1, \forall i, j \in G \tag{4.4.4}$$

$$x_{ij} \in \{0,1\}, \forall i \in N, j \in N, i \neq j \tag{4.4.5}$$

$$u_i \geq 0, \forall i \in N \tag{4.4.6}$$

The notations used in this context have the following meanings. The symbol n represents the total number of nodes (cities), and N represents the set of these nodes. N_0 is a set of nodes that excludes the starting node 0. The notation c_{ij} represents the distance between two nodes. x_{ij} is the decision variable that takes a value of 1 when the two nodes are connected. u_i is an auxiliary variable used to determine the visiting sequence. G represents a graph where several nodes are already connected. These connections are predicted by the neural network as previously described.

The objective is to minimize the total traveling distance. Constraints (4.4.1) ensure that each node has an outgoing connection to another node. Constraints (4.4.2) enforce that each node has an incoming connection to another node. Constraints (4.4.3) are known as subtour elimination constraints. They determine the visiting sequence of each node. Constraints (4.4.4) are newly introduced to complement the classical MTZ. These constraints serve the purpose of ensuring that the connections between the predicted anchors and their nearest neighbors are maintained. These connections are established in the graph G, and constraints (4.4.4) guarantee that these connections are enforced within the model. Constraints (4.4.5) and (4.4.6) specify the restrictions on the variables.

4.5 Training and Evaluations

This section introduces the training process of the supervised learning model. In Section 4.5.1, we discuss the impact of under-sampling. Section 4.5.2 explores the trade-off between precision and recall by adjusting the prediction thresholds. In Section 4.5.3, we evaluate the generalization ability. Section 4.5.4 presents the performance metrics, including accuracy, precision, recall, specificity, F1 score, and training time.

To ensure a diverse training dataset, we generated a substantial number of instances. Specifically, we created 1 million TSP25 instances, 0.5 million TSP50 instances, and 0.25

million TSP100 instances. This resulted in a total of 75 million features and labels. We did not include TSPLIB instances for two reasons. Firstly, while we have the optimal objective function values, the optimal routes are unavailable for most TSPLIB instances, making it impossible to generate ground truth for these instances. Secondly, it is thought better to reserve the TSPLIB instances to evaluate the generalization ability of the model.

4.5.1 Under Sampling

In this section, we present ROC curves to evaluate the necessity of performing under-sampling as a preprocessing step. We trained two models, one with under-sampling and the other without, for 20 epochs using the entire dataset. Then, we plotted ROC curves using four validation datasets that had not been seen during the training process. The results are presented in Figure 4.5.1.

The figure depicts the ROC curves of different instances, comparing the performance of two models with and without under-sampling. Each subfigure represents a specific instance, with dashed lines representing the ROC curves with under-sampling and solid lines representing the curves without under-sampling. The diagonal line represents a benchmark model that makes random predictions.

For the generated instances, both models present similar performance, with an area under the curve (AUC) of approximately 0.9 across all instances. The high AUC values indicate strong prediction performance. When evaluating the TSPLIB instances, the model trained without under-sampling slightly outperforms the model with under-sampling by a margin of 0.01 in terms of AUC. Based on this performance difference, the decision was made to not use under-sampling for subsequent experiments. In summary, the ROC curves provide evidence of the efficacy of the supervised learning model in predicting anchors.

4.5.2 Trade-off Between Precision and Recall

Precision and recall are commonly in conflict with each other, meaning that improving one metric often results in a decrease in the other metric. In our case, precision holds more significance than recall because we aim to minimize the wrong information. Increasing the

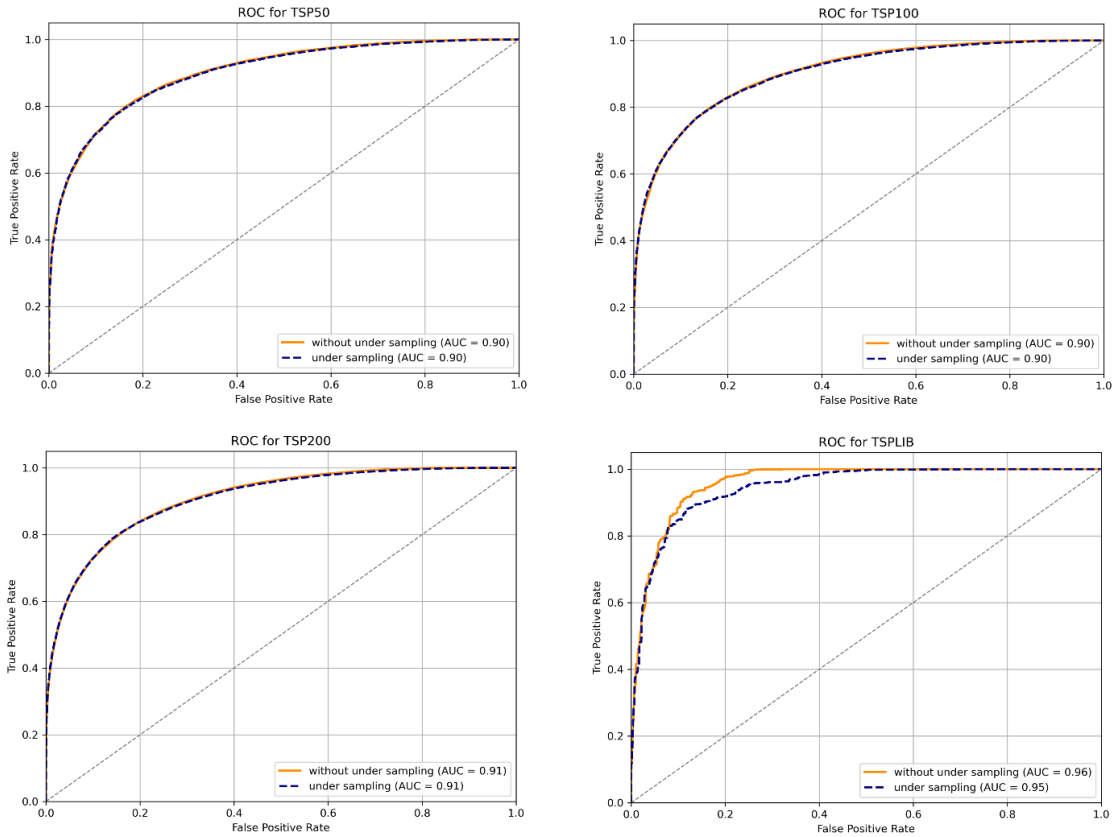

Figure 4.5.1: Comparisons on ROC with and without under sampling

prediction threshold can enhance precision, but it comes at the cost of reduced recall. Therefore, it is vital to achieve a good trade-off between the two metrics when determining the threshold for our trained model. Figure 4.5.2 illustrates the precision-recall curves obtained from experiments on validation datasets, providing valuable insights into the prediction performance of the trained model across different thresholds.

The figure demonstrates that as we decrease the prediction threshold to increase the recall score, the precision score generally decreases. However, the precision does not diminish to zero even when the recall rate reaches one. This is due to the fact that the positive data comprises approximately 80% of the dataset. Consequently, even if the model classifies all nodes as anchors, there will still be around 80% correct predictions. To ensure a high precision for the model, a lower bound of 0.95 for precision is established. Subsequently, the corresponding thresholds for TSP25, TSP50, TSP100, TSP200, and TSPLIB are determined. The calculated thresholds are 0.781, 0.764, 0.759, 0.743, and 0.789, respectively. The strictest threshold of 0.789 is selected to guarantee a precision score of at least 95% for all instances. By utilizing this threshold, it is found that the recall ranges between 0.869 and 0.908 for all instances. This approach effectively strikes a balance between precision and recall while maintaining a high level of precision for the model.

4.5.3 Generalization Capacity on Solving Large-scale Instances

We conducted tests to evaluate the generalization ability of the trained model. While the model is trained with instances containing fewer than 100 nodes, it is evaluated on diverse datasets ranging from 100 nodes to 1,000 nodes. The evaluation metrics employed include accuracy, precision, recall, specificity, and F1 score. The results of these evaluations are depicted in the figure below.

As depicted in Figure 4.5.3, the left subfigure illustrates the metrics obtained using the default threshold of 0.5, while the right subfigure displays the metrics using the new threshold of 0.789. The use of the default threshold resulted in a low specificity, indicating that numerous non-anchor nodes were incorrectly classified as anchors, potentially leading

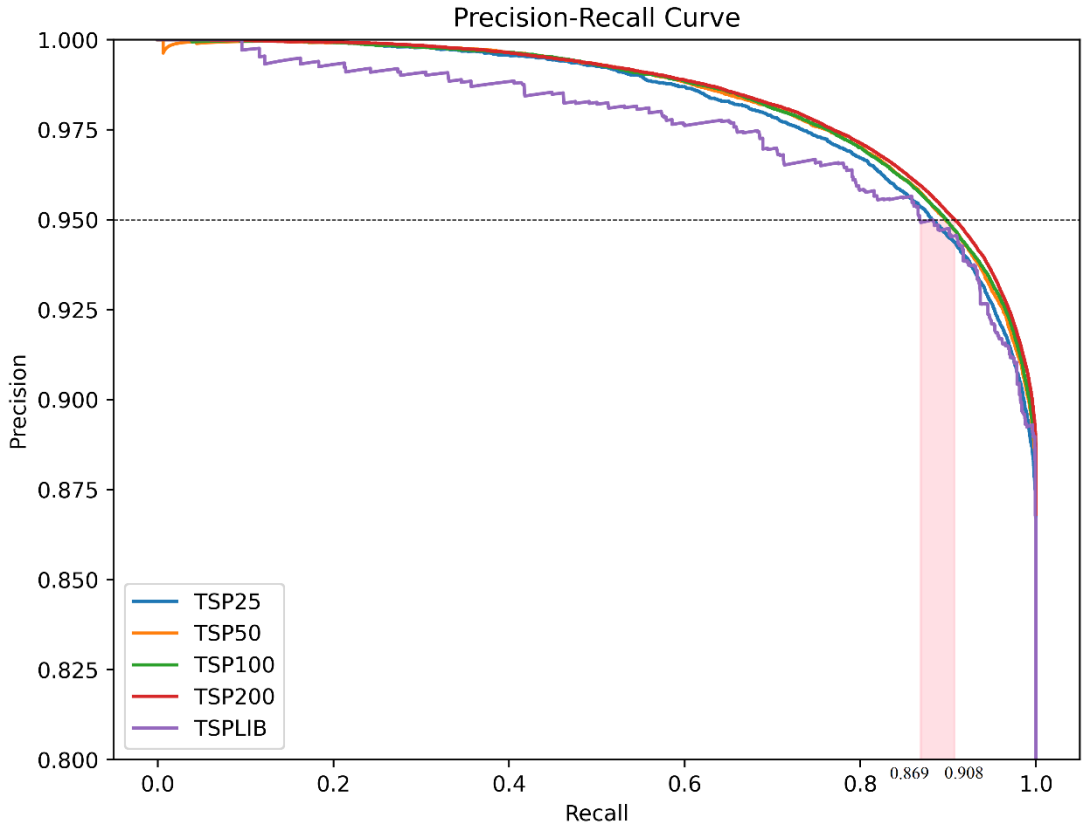

Figure 4.5.2: Precision-Recall curve for different instances

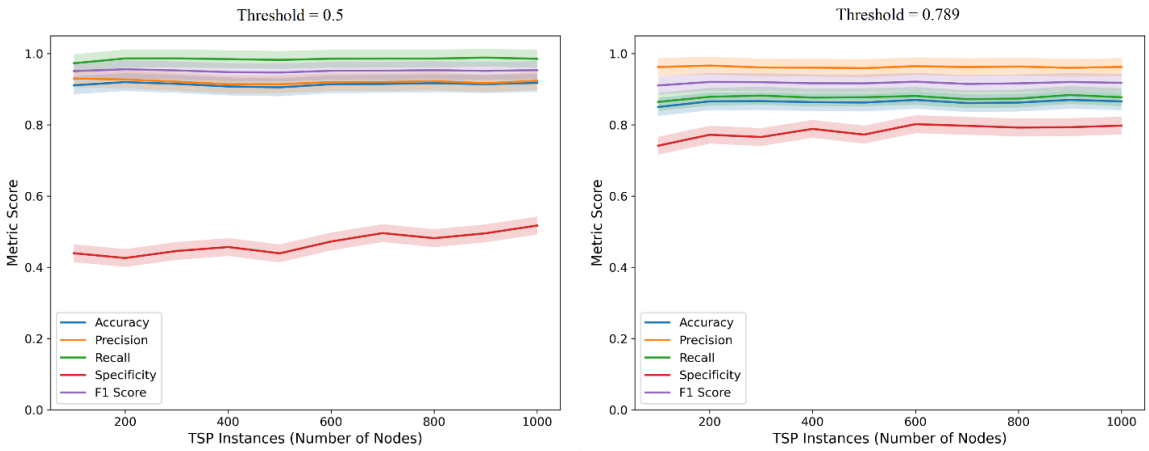

Figure 4.5.3: The generalization capacity of the model with improved thresholds.

to erroneous connections between nodes. By increasing the threshold to 0.789, the specificity improved significantly from an average of 0.5 to an average of 0.9, while maintaining a recall rate above 0.8. This tradeoff resulted in an enhanced overall performance of the model.

Furthermore, it was observed that the generalization ability of the model remained stable as the instance size increased, despite being trained solely on generated TSP instances with fewer than 100 nodes. This suggests that the trained model can effectively predict larger instances without experiencing a substantial reduction in prediction accuracy.

4.5.4 Prediction Performances

After conducting preliminary experiments, we proceeded to retrain the machine learning model using the determined hyperparameters and settings. The model was trained for 100 epochs, but due to the early stopping policy, the training process terminated after 20 epochs. The total training time was 4.8 hours. The trained model was then evaluated on various datasets, including the training and testing datasets, as well as generated TSP instances with 25, 50, 100, and 200 cities, and instances from the TSPLIB. The threshold for predicting positive nodes was set to 0.789 as previously mentioned.

Table 4.5.1 presents the average results obtained from the evaluation. Specifically, there are 1,000 instances in each of the generated instances (TSP25, TSP50, TSP100, and TSP200). The TSPLIB instances included in the evaluation satisfy the Euclidean distance requirement and have known optimal tours available. The evaluation time indicated in the table refers to the total prediction time.

The results demonstrate that the trained model achieved high accuracy, ranging from 0.855 to 0.880, indicating its effectiveness in distinguishing anchors from non-anchor nodes. Furthermore, the model exhibited excellent precision, ranging from 0.950 to 0.958, meaning that approximately 95% of the predicted anchors were true anchors. The recall rates were consistently high, ranging from 0.875 to 0.878, indicating that over 87% of the anchor nodes were successfully identified. Additionally, the model displayed a high level of specificity, with values ranging from 0.715 to 0.902. The F1 score, which represents a

Table 4.5.1: Validation results of the trained model

Dataset	Accuracy	Precision	Recall	Specification	F1	Time (s)
TSP25	0.855	0.951	0.877	0.715	0.913	0.803
TSP50	0.858	0.955	0.878	0.724	0.915	1.646
TSP100	0.856	0.955	0.876	0.729	0.914	3.396
TSP200	0.858	0.958	0.875	0.748	0.915	5.842
TSPLIB	0.880	0.949	0.869	0.902	0.907	0.199

balanced performance between precision and recall, ranged from 0.913 to 0.915.

The prediction time increased with the size of the input, ranging from 0.803 seconds for predicting 1,000 TSPLIB25 instances to 5.842 seconds for predicting 1,000 TSP200 instances. These results highlight the efficiency of the trained model in predicting anchors across various TSP instances.

It is worth noting that the trained model was solely exposed to TSP instances with up to 100 nodes during the training process and was not specifically trained on TSP200 or TSPLIB instances. Moreover, the instances in TSPLIB have distinct patterns from the generated instances used for training. However, the model demonstrated a robust ability to generalize and effectively predict anchors in TSP200 and TSPLIB instances, indicating its strong generalization ability.

4.6 Computational Results

In this section, we compare the proposed approach with several baseline approaches in solving TSP instances. The evaluation involves two different settings: 1) the integration with exact methods and 2) the integration with heuristic algorithms. In the first setting, we propose the anchor-based Miller-Tucker-Zemlin method (denoted as Anchor-MTZ) and compare it with the traditional MTZ method. We solve the Anchor-MTZ and MTZ using the commercial solver Gurobi. In the second setting, we propose the anchor insertion algorithms and compare it with baseline algorithms such as Concorde, LKH3, five constructive heuristics, and a state-of-the-art learning-based approach called GPN.

The MTZ model is solved using the Gurobi commercial solver (version 8.1.0.) on a workstation equipped with an Intel Core i7-3770 CPU (3.40 GHz) and 8 GB of RAM. The anchor insertion algorithm, as well as the baseline algorithms, are implemented in Python and executed on the same machine. It's important to note that no GPU was utilized during the training of the proposed model. However, for training the GPN algorithm used for comparison, a different machine equipped with an RTX 3080 Ti GPU was employed due to the extensive computational requirements.

4.6.1 Integration with Exact Methods

The predicted anchors can be leveraged to enhance the efficiency of exact algorithms. Specifically, we explore the use of anchors as a means to obtain partial nodes to be connected in advance, which can then be inputted into commercial solvers, thereby reducing computational time.

In this section, we evaluate the performance of the classical MTZ model with and without anchors. The MTZ model is a mathematical formulation that uses sequences to eliminate subtours. Each node in the TSP is associated with a sequence variable that represents its position in the tour. By imposing appropriate constraints on these sequence variables, the MTZ model ensures that subtours are eliminated.

Although we also considered the DFJ model, the results are not presented in this section due to the limited solvable scale of the model without considering additional enhancements such as lazy constraints.

Table 4.6.1 presents the results of solving the generated instances using the MTZ and Anchor-MTZ approaches. The “Opt.” column denotes the optimal solutions obtained by Concorde. The "Avg. Obj." column denotes the average objective function value obtained from solving the instances. The "Gurobi Time" column represents the total computational time required to solve the instances, while the “Predict Time” refers to the time for predicting the anchors using the trained model. Lastly, the "Gap" column provides the percentage differences between the obtained solutions and the optimal solutions, indicating the extent of deviation from optimality.

The MTZ approach successfully obtains optimal solutions for TSP50 instances. However, it encounters computational limitations when tackling larger instances, such as those with more than 100 nodes. In contrast, the Anchor-MTZ approach demonstrates superior performance in solving large instances. For example, when solving TSP250, the Anchor-MTZ approach achieves a significantly lower gap of 16.85% compared to the gap of 47.4% observed with the MTZ approach. The total time for predicting the anchors remains consistently low, which is less than 2 seconds for each dataset.

Table 4.6.1: Performance Comparison: MTZ vs. Anchor-MTZ

		MTZ			Anchor-MTZ			
Dataset	Opt.	Avg. Obj.	Gurobi T. (s)	Gap (%)	Avg. Obj.	Predict T. (s)	Gurobi T. (s)	Gap (%)
TSP50	5730	5730	332	0.00	5762	1.6	48	0.56
TSP100	7729	7818	3461	1.15	7792	1.7	2152	0.82
TSP150	9414	9872	3600	4.87	9507	1.6	3419	0.99
TSP200	10799	13117	3600	21.46	11510	1.7	3600	6.58
TSP250	11895	17534	3600	47.40	13900	1.7	3600	16.85

4.6.2 Integration with Heuristics

The predicted anchors can also be leveraged to enhance heuristics. To illustrate this, we introduce the anchor insertion algorithm, as introduced in Section 4.4.4. In this section, we compare the performance of the anchor insertion algorithm with several baseline approaches, including LKH3, Concorde, and GPN, among others. The detailed results of these comparisons are presented in Table 4.6.2. The results are obtained from 10 instances within each dataset. The reported objective function value represents the average performance across these instances, while the time denotes the total computational time.

In the field of solving TSP instances, Concorde and LKH3 are widely recognized traditional approaches. Concorde stands out for consistently producing high-quality solutions but requires longer computational time, especially for larger datasets. In contrast, LKH3 achieves similar results to Concorde with shorter solving time, making it a more efficient alternative in terms of computational resources. Constructive heuristics provide fast solutions but sacrifice solution quality. The choice of method depends on the specific requirements, considering factors such as dataset size, available computational resources, and the desired balance between solution quality and computation time.

Our primary objective is to compare our proposed approach with an existing learning-based method known as GPN. Through this comparison, we have discovered that the anchor insertion method shows great promise. In fact, it outperforms the GPN method in terms of both solution quality and computational time. This finding is significant as it demonstrates the superiority of the anchor insertion approach in addressing the TSP. It is worth noting that the proposed anchor-based approaches are still in the process of development and refinement. A dedicated algorithm is needed to fully utilize the anchors. However, there is a growing belief that integrating anchor methods with heuristics can lead to further improvements in performance. By surpassing the performance of existing learning-based methods like GPN, the anchor insertion approach opens new possibilities for advancing the state-of-the-art in solving the TSP.

Table 4.6.2: Performance Comparison: Heuristics

	TSP 250		TSP 500		TSP 750		TSP 1000	
Method	Obj	Time (s)	Obj	Time (s)	Obj	Time (s)	Obj	Time (s)
LKH3	11983	1.3	16508	3.7	20217.0	8.1	23102.7	14.2
Concorde	11953	29.4	16489	150.6	20187.7	666.1	23077.7	1082.9
Nearest Neighbor	15119	0.1	20618	0.3	25252.4	0.7	28895.2	1.2
Shortest First	14182	0.3	19327	1.5	23837.1	2.9	26935.6	5.3
Nearest Insertion	14747	5.2	20457	40.9	25079.6	138.1	28949.1	327.2
Farthest Insertion	14173	5.2	20161	40.7	25050.9	138.9	28890.9	332.1
Cheapest Insertion	14125	18.8	19774	142.7	24333.8	480.3	27739.4	1101.6
GPN	15253	8.3	23037	12.3	29852.9	19.0	36709.3	25.1
Anchor Insertion (ours)	13308	2.2	18712	3.1	22991.6	5.8	26377.5	10.2

4.6.3 Generalization ability on TSPLIB instances

Table 4.7.1 provides a comprehensive comparison between the GPN and Anchor Insertion methods on multiple instances from the TSPLIB dataset. The comparison includes objective values, computation times in seconds, and gap percentages for both approaches. The results indicate that the anchor insertion algorithm consistently outperforms the GPN method in terms of objective values and solve times, particularly for larger instances. For instance, in the ca4663 instance, the gap percentage for the anchor insertion algorithm is 29.61%, whereas for the GPN method it is 120.43%. Similarly, in the ja9847 instance, the gap percentage for the anchor insertion algorithm is 26.55%, compared to 158.29% for the GPN method. These significant improvements demonstrate the effectiveness of the anchor insertion algorithm in achieving better solutions with reduced gaps compared to the GPN method.

4.7 Insights and Discussions

During our training and testing process, we made several interesting observations. Firstly, we found that the quantity of data had a more significant impact on performance than the dimensions, i.e., the number of features of the data. Surprisingly, even with a dimension as small as three, we were able to train models that yielded satisfactory results. This contradicted our initial assumption that higher input dimensions would provide more accurate predictions by offering more information. There are several possible explanations for this phenomenon. Firstly, the task of determining whether a node should be an anchor may not require extensive local information, as confirmed by our experiments. Secondly, incorporating irrelevant information into the features can introduce noise, particularly when training data is limited. Lastly, higher-dimensional inputs can pose challenges in effectively training the model.

Secondly, we discovered that under-sampling and normalization may not always be necessary. Initially, we believed that under-sampling was necessary to address the data imbalance issue. As shown in Figure 4.3.3, positive samples accounted for 80% - 90% of

Table 4.7.1: Performance Comparison: TSPLIB Instances

Instance	Opt.	GPN Obj.	GPN Time (s)	GPN Gap (%)	Anchor Obj.	Anchor Time (s)	Anchor Gap (%)
a280	2,579	3,606	2.48	39.82	3,202	0.26	24.16
berlin52	7,542	8,962	0.13	18.83	8,007	0.15	6.17
ca4663	1,290,319	2,844,220	10.60	120.43	1,672,320	50.20	29.61
ch130	6,110	6,921	0.28	13.27	6,972	0.15	14.11
ch150	6,528	7,923	0.35	21.37	7,118	0.15	9.04
eil101	629	720	0.22	14.47	697	0.14	10.81
eil51	426	457	0.12	7.28	472	0.13	10.80
eil76	538	623	0.17	15.80	609	0.14	13.20
ja9847	491,924	1,270,604	21.84	158.29	622,535	425.84	26.55
kroA100	21,282	27,911	0.23	31.15	23,670	0.16	11.22
kroC100	20,749	27,164	0.23	30.92	22,212	0.15	7.05
kroD100	21,294	26,057	0.23	22.37	23,588	0.16	10.77
lin105	14,379	20,484	0.25	42.46	16,926	0.16	17.71
pcb442	50,778	71,144	1.00	40.11	61,747	0.27	21.60
pr1002	259,045	424,108	2.30	63.72	318,840	0.93	23.08
pr2392	378,032	727,079	5.63	92.33	478,371	7.94	26.54
pr76	108,159	122,867	0.17	13.60	125,078	0.14	15.64
rd100	7,910	8,662	0.23	9.51	9,206	0.14	16.38
tsp225	3,916	5,177	0.54	32.20	4,638	0.16	18.44

the entire dataset. Consequently, we expected that under-sampling would help reduce biases caused by sample imbalance. However, our experimental results led us to different conclusions. While under-sampling improved precision, it significantly decreased recall. After careful comparison, we found that it is better to prioritize achieving a higher recall initially and then increase precision by appropriately adjusting the threshold for predicting positive samples. Moreover, we observed an interesting phenomenon regarding normalization. When the training data reached a certain quantity, the neural network automatically developed internal structures that performed a similar function to normalization. This is an advantage of our proposed model over the traditional GPN approach. Unlike the proposed approach, the GPN performed significantly worse without normalization.

Thirdly, the best way to utilize the predicted anchors remains unclear. In our research, we attempted to integrate the predicted anchors with exact methods and heuristic algorithms. While the integration with exact methods significantly reduced computational time, it also resulted in a loss of otherwise guaranteed optimality. Additionally, we explored combining the model with various constructive heuristics. However, most of these heuristics demonstrated insensitivity to the initial solution, making the anchors less useful. Based on our experimental findings, we believe that designing a dedicated algorithm based on the characteristics of anchors may be necessary to fully exploit their potential. Furthermore, we are considering the integration of anchor points with the LKH algorithm.

Fourthly, there are alternative ways to define the anchors. In this paper, we defined anchors as nodes that should connect to their nearest neighbors. However, anchors could also be defined as nodes that connect to their two nearest neighbors, as each node needs to connect to two other nodes. This approach could provide additional post-prediction information. Nevertheless, this made the model more challenging to train, and the post-prediction results were more prone to conflicts.

Admittedly, there are still limitations to our proposed algorithms. As mentioned before, the percentage of anchors in TSP instances is approximately 80%-90%. Even if the model could perfectly predict every anchor, it may not completely solve the TSP. Therefore, a

second-stage algorithm is required to complete the solution. Considering that some cities serve as anchors for each other, the number of optimal arcs that can be determined is around 60%. Additionally, for certain anchors, there may be more than one nearest point, meaning that knowing the anchors alone is not sufficient to connect them to other nodes.

4.8 Conclusion

This paper introduces an approach for solving the TSP by integrating machine learning with optimization. Through a supervised learning process, we train a model using small-scale instances and demonstrate its ability to generalize well to larger-scale instances. We explore various preprocessing techniques, hyperparameter tuning, and feature selection to enhance the performance of model.

The proposed approach exhibits a strong generalization ability. The trained model can accurately identify 87% of the anchors with a precision exceeding 95%. Remarkably, the model is trained on instances with less than 100 nodes, but it successfully predicts anchors for instances with more than 1,000 nodes without any decline in prediction accuracy. Furthermore, we integrate the predicted anchors with exact methods, leading to reduced computational time when solving the MTZ formulation. When combined with a heuristic, the proposed anchor insertion algorithm outperforms the state-of-the-art learning-based approach GPN in terms of both solution quality and computational time.

Future research can be directed towards designing dedicated algorithms tailored to the specific characteristics of anchors. Additionally, it is promising to integrate the predicted anchors with the LKH algorithm. Despite some limitations, this paper provides a novel new way in leveraging machine learning to address the TSP, opening up new possibilities for efficient and effective learning-based algorithms.

Chapter 5. Summary and Conclusions

This dissertation delves into both the traditional and innovative solutions for Vehicle Routing Problems (VRPs) and their variants. VRPs are classic problems in the field of combinatorial optimization, and they also hold significant importance in logistics and transportation. These problems require the allocation of resources in a way that optimizes various objectives, such as minimizing transportation costs, maximizing customer satisfaction, and ensuring timely deliveries, all while considering common constraints like maximum capacity and time windows. Efficiently solving VRPs is crucial for improving operational efficiency, reducing costs, and ensuring the timely and effective delivery of goods and services. Consequently, substantial research efforts have been dedicated to developing innovative methods, algorithms, and technologies for addressing VRPs and their variants. This dissertation provides a comprehensive exploration of exact methods, heuristics, and innovative supervised learning methods, aiming to offer a well-rounded perspective on addressing the critical challenges in this domain.

In Chapter 2, we focused on the pickup and delivery problem with transshipments (PDP-T) and its extension, the pickup and delivery problem with time windows and transshipments (PDPTW-T). Through a thorough examination of existing models, we identified limitations and proposed refined formulations. Furthermore, we introduced a new mixed-integer linear programming (MILP) formulation tailored to tackle PDP-T and PDPTW-T. Our computational results showcased the superior performance of this new model.

In Chapter 3, we addressed a variant of the vehicle routing problem aimed at efficiently assigning consultants to serve clients with minimized costs and maximized customer satisfaction. Our objective was to simultaneously assign consultants to clients while optimizing the traveling routes for the consultants. Our approach took into account skill requirements, capacity limitations, fixed demand, and a maximum number of travel legs. Additionally, we introduced a priority matching mechanism to ensure that consultants were assigned to clients with suitable priority levels. To address the computational complexity of this problem, we introduced two algorithms: the RMIP decomposition algorithm and a MIP-based neighborhood search algorithm. We conducted a comparative analysis against

an existing MILP formulation. Our computational evaluation, encompassing a diverse set of synthetic and real-life instances, underscored the superior solution quality and reduced computational time achieved by our proposed algorithms, particularly in large-scale and real-world scenarios.

In Chapter 4, we delved into the Traveling Salesman Problem (TSP), which serves as the foundational version of vehicle routing problems. To address this classic problem, we introduced a novel concept named as "anchors," representing nodes that should connect to their nearest neighbors to form the optimal routing. Our approach used supervised learning to predict these anchors based solely on local information, setting it apart from previous models that relied on global data. Experimental results demonstrate that our model successfully identifies 87% of the anchors with a precision of over 95%. By integrating these predicted anchors into well-established methods such as the Miller-Tucker-Zemlin (MTZ) model and insertion algorithms, we achieved an improvement in both solution quality and computational efficiency. This work also underscores the scalability and adaptability of our learning-based approach.

In summary, this dissertation has explored the vehicle routing problems and their variants from three aspects: modeling, heuristics, and learning-based approaches. This research has yielded enhancements in solution quality, reduced computational time, and improved scalability for specific problems and methodologies. As a future research direction, we look forward to further investigating the application of "anchors," expanding their utility, and finding more effective ways to leverage this concept.

References

An, Y. J., Kim, Y. D., Jeong, B. J., & Kim, S. D. (2012). Scheduling healthcare services in a home healthcare system. Journal of the Operational Research Society, 63(11), 1589–1599. https://doi.org/10.1057/jors.2011.153

Applegate, D., Cook, W., Dash, S., & Rohe, A. (2002). Solution of a Min-Max Vehicle Routing Problem. INFORMS Journal on Computing, 14(2), 132–143. https://doi.org/10.1287/ijoc.14.2.132.118

Arslan, A. M., Agatz, N., Kroon, L., & Zuidwijk, R. (2019). Crowdsourced delivery—a dynamic pickup and delivery problem with ad hoc drivers. Transportation Science, 53(1), 222–235. https://doi.org/10.1287/trsc.2017.0803

Bengio, Y., Lodi, A., & Prouvost, A. (2021). Machine learning for combinatorial optimization: A methodological tour d'horizon. European Journal of Operational Research, 290(2), 405–421. https://doi.org/10.1016/j.ejor.2020.07.063

Berbeglia, G., Cordeau, J. F., & Laporte, G. (2010). Dynamic pickup and delivery problems. European Journal of Operational Research, 202(1), 8–15. https://doi.org/10.1016/j.ejor.2009.04.024

Bogyrbayeva, A., Yoon, T., Ko, H., Lim, S., Yun, H., & Kwon, C. (2023). A deep reinforcement learning approach for solving the Traveling Salesman Problem with Drone. Transportation Research Part C: Emerging Technologies, 148, 103981. https://doi.org/10.1016/j.trc.2022.103981

Castillo-Salazar, J. A., Landa-Silva, D., & Qu, R. (2016). Workforce scheduling and routing problems: literature survey and computational study. Annals of Operations Research, 239(1), 39–67. https://doi.org/10.1007/s10479-014-1687-2

Cissé, M., Yalçındağ, S., Kergosien, Y., Şahin, E., Lenté, C., & Matta, A. (2017). OR problems related to Home Health Care: A review of relevant routing and scheduling problems. Operations Research for Health Care, 13–14, 1–22. https://doi.org/10.1016/j.orhc.2017.06.001

Cordeau, J. F. (2006). A branch-and-cut algorithm for the dial-a-ride problem. Operations

Research, 54(3), 573–586. https://doi.org/10.1287/opre.1060.0283

Cortés, C. E., Matamala, M., & Contardo, C. (2010). The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method. European Journal of Operational Research, 200(3), 711–724. https://doi.org/10.1016/j.ejor.2009.01.022

Cortes, J. D., & Suzuki, Y. (2020). Vehicle Routing with Shipment Consolidation. International Journal of Production Economics, 227, 107622. https://doi.org/10.1016/j.ijpe.2020.107622

Costa, L., Contardo, C., & Desaulniers, G. (2019). Exact branch-price-and-cut algorithms for vehicle routing. In Transportation Science (Vol. 53). https://doi.org/10.1287/trsc.2018.0878

Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B., & Song, L. (2017). Learning Combinatorial Optimization Algorithms over Graphs. Advances in Neural Information Processing Systems, 30. http://arxiv.org/abs/1704.01665

Danloup, N., Allaoui, H., & Goncalves, G. (2018). A comparison of two meta-heuristics for the pickup and delivery problem with transshipment. Computers and Operations Research, 100, 155–171. https://doi.org/10.1016/j.cor.2018.07.013

De Bruecker, P., Beliën, J., De Boeck, L., De Jaeger, S., & Demeulemeester, E. (2018). A model enhancement approach for optimizing the integrated shift scheduling and vehicle routing problem in waste collection. European Journal of Operational Research, 266(1), 278–290. https://doi.org/10.1016/j.ejor.2017.08.059

Dong, Y., Pinto, J. M., Sundaramoorthy, A., & Maravelias, C. T. (2014). MIP model for inventory routing in industrial gases supply chain. Industrial and Engineering Chemistry Research, 53(44), 17214–17225. https://doi.org/10.1021/ie500460c

Eveborn, P., Rönnqvist, M., Einarsdólttir, H., Eklund, M., Lidén, K., & Almroth, M. (2009). Operations research improves quality and efficiency in home care. Interfaces, 39(1), 18–34. https://doi.org/10.1287/inte.1080.0411

Fikar, C., & Hirsch, P. (2017). Home health care routing and scheduling: A review. Computers and Operations Research, 77, 86–95. https://doi.org/10.1016/j.cor.2016.07.019

Fu, Z.-H., Qiu, K.-B., & Zha, H. (2021). Generalize a Small Pre-trained Model to Arbitrarily Large TSP Instances. Proceedings of the AAAI Conference on Artificial Intelligence, 35(8), 7474–7482. https://doi.org/10.1609/aaai.v35i8.16916

Ha, Q. M., Deville, Y., Pham, Q. D., & Hà, M. H. (2018). On the min-cost Traveling Salesman Problem with Drone. Transportation Research Part C: Emerging Technologies, 86, 597–621. https://doi.org/10.1016/j.trc.2017.11.015

Hansen, P., Mladenović, N., & Moreno Pérez, J. A. (2010). Variable neighbourhood search: Methods and applications. Annals of Operations Research, 175(1), 367–407. https://doi.org/10.1007/s10479-009-0657-6

Helsgaun, K. (2000). An effective implementation of the Lin–Kernighan traveling salesman heuristic. European Journal of Operational Research, 126(1), 106–130. https://doi.org/10.1016/S0377-2217(99)00284-2

Ho, S. C., & Leung, J. M. Y. (2010). Solving a manpower scheduling problem for airline catering using metaheuristics. European Journal of Operational Research, 202(3), 903–921. https://doi.org/10.1016/j.ejor.2009.06.030

Joshi, C. K., Cappart, Q., Rousseau, L. M., & Laurent, T. (2022). Learning the travelling salesperson problem requires rethinking generalization. Constraints, 27(1–2), 70–98. https://doi.org/10.1007/s10601-022-09327-y

Karimi-Mamaghan, M., Mohammadi, M., Meyer, P., Karimi-Mamaghan, A. M., & Talbi, E.-G. (2022). Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. European Journal of Operational Research, 296(2), 393–422. https://doi.org/10.1016/j.ejor.2021.04.032

Koç, Ç., Laporte, G., & Tükenmez, İ. (2020). A review of vehicle routing with simultaneous pickup and delivery. Computers and Operations Research, 122, 104987. https://doi.org/10.1016/j.cor.2020.104987

Kong, F., Li, J., Jiang, B., Wang, H., & Song, H. (2022). Trajectory Optimization for Drone Logistics Delivery via Attention-Based Pointer Network. IEEE Transactions on Intelligent Transportation Systems. https://doi.org/10.1109/TITS.2022.3168987

Kool, W., van Hoof, H., & Welling, M. (2018). Attention, Learn to Solve Routing

Problems! International Conference on Learning Representations. http://arxiv.org/abs/1803.08475

Kovacs, A. A., Parragh, S. N., Doerner, K. F., & Hartl, R. F. (2012). Adaptive large neighborhood search for service technician routing and scheduling problems. Journal of Scheduling, 15(5), 579–600. https://doi.org/10.1007/s10951-011-0246-9

Kwon, Y.-D., Choo, J., Kim, B., Yoon, I., Gwon, Y., & Min, S. (2020). POMO: Policy Optimization with Multiple Optima for Reinforcement Learning. Advances in Neural Information Processing Systems, 33, 21188–21198. http://arxiv.org/abs/2010.16011

Laporte, G. (1992). The Traveling Salesman Problem: An overview of exact and approximate algorithms. European Journal of Operational Research, 59(2), 231–247. https://doi.org/10.1016/0377-2217(92)90138-Y

Li, H., & Lim, A. (2001). A metaheuristic for the pickup and delivery problem with time windows. Proceedings of the International Conference on Tools with Artificial Intelligence, 12(02), 173–186. https://doi.org/10.1142/s0218213003001186

Li, Z., Chen, Q., & Koltun, V. (2018). Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search. Advances in Neural Information Processing Systems, 31. http://arxiv.org/abs/1810.10659

Lin, S., & Kernighan, B. W. (1973). An Effective Heuristic Algorithm for the Traveling-Salesman Problem. Operations Research, 21(2), 498–516. https://doi.org/10.1287/opre.21.2.498

Ling, Z., Zhang, Y., & Chen, X. (2023). A Deep Reinforcement Learning Based Real-Time Solution Policy for the Traveling Salesman Problem. IEEE Transactions on Intelligent Transportation Systems. https://doi.org/10.1109/TITS.2023.3256563

Liu, Z., Li, X., & Khojandi, A. (2022). The flying sidekick traveling salesman problem with stochastic travel time: A reinforcement learning approach. Transportation Research Part E: Logistics and Transportation Review, 164(January), 102816. https://doi.org/10.1016/j.tre.2022.102816

Lourenço, H. R., Martin, O. C., & Stützle, T. (2003). Iterated Local Search. In Handbook of metaheuristics (pp. 320–353). Springer, Boston, MA. https://doi.org/10.1007/978-

3-319-07124-4_8

Macrina, G., Di Puglia Pugliese, L., Guerriero, F., & Laporte, G. (2020). Drone-aided routing: A literature review. Transportation Research Part C: Emerging Technologies, 120. https://doi.org/10.1016/j.trc.2020.102762

Maheshwari, A., Misra, S., Gudi, R. D., & Subbiah, S. (2020). A Short-Term Planning Framework for the Operation of Tanker-Based Water Distribution Systems in Urban Areas. Industrial and Engineering Chemistry Research, 59(20), 9575–9592. https://doi.org/10.1021/acs.iecr.0c00303

Masson, R., Lehuédé, F., & Péton, O. (2013). An Adaptive Large Neighborhood Search for the Pickup and Delivery Problem with Transfers. Transportation Science, 47(3), 344–355. https://doi.org/10.1016/j.cor.2017.06.012

Maya, P., Sörensen, K., & Goos, P. (2012). A metaheuristic for a teaching assistant assignment-routing problem. Computers and Operations Research, 39(2), 249–258. https://doi.org/10.1016/j.cor.2011.04.001

Mazyavkina, N., Sviridov, S., Ivanov, S., & Burnaev, E. (2021). Reinforcement learning for combinatorial optimization: A survey. Computers & Operations Research, 134, 105400. https://doi.org/10.1016/j.cor.2021.105400

Mele, U. J., Gambardella, L. M., & Montemanni, R. (2021). A New Constructive Heuristic Driven by Machine Learning for the Traveling Salesman Problem. Algorithms, 14(9), 267. https://doi.org/10.3390/a14090267

Miller, C. E., Zemlin, R. A., & Tucker, A. W. (1960). Integer Programming Formulation of Traveling Salesman Problems. Journal of the ACM (JACM), 7(4), 326–329. https://doi.org/10.1145/321043.321046

Misra, S., Saxena, D., Kapadi, M., Gudi, R. D., & Srihari, R. (2018). Short-Term Planning Framework for Enterprise-wide Production and Distribution Network of a Cryogenic Air Separation Industry. Industrial and Engineering Chemistry Research, 57(49), 16841–16861. https://doi.org/10.1021/acs.iecr.8b05138

Mitrović-Minić, S., & Laporte, G. (2006). The pickup and delivery problem with time windows and transshipment. INFOR: Information Systems and Operational Research,

44(3), 217–227. https://doi.org/10.1080/03155986.2006.11732749

Nazari, M., Oroojlooy, A., Snyder, L. V., & Takáč, M. (2018). Reinforcement Learning for Solving the Vehicle Routing Problem. Advances in Neural Information Processing Systems, 31. http://arxiv.org/abs/1802.04240

Paraskevopoulos, D. C., Laporte, G., Repoussis, P. P., & Tarantilis, C. D. (2017). Resource constrained routing and scheduling: Review and research prospects. European Journal of Operational Research, 263(3), 737–754. https://doi.org/10.1016/j.ejor.2017.05.035

Perera, J., Liu, S. H., Mernik, M., Črepinšek, M., & Ravber, M. (2023). A Graph Pointer Network-Based Multi-Objective Deep Reinforcement Learning Algorithm for Solving the Traveling Salesman Problem. Mathematics, 11(2), 1–21. https://doi.org/10.3390/math11020437

Qu, Y., & Bard, J. F. (2012). A GRASP with adaptive large neighborhood search for pickup and delivery problems with transshipment. Computers and Operations Research, 39(10), 2439–2456. https://doi.org/10.1016/j.cor.2011.11.016

Rais, A., Alvelos, F., & Carvalho, M. S. (2014). New mixed integer-programming model for the pickup-and-delivery problem with transshipment. European Journal of Operational Research, 235(3), 530–539. https://doi.org/10.1016/j.ejor.2013.10.038

Rasmussen, M. S., Justesen, T., Dohn, A., & Larsen, J. (2012). The Home Care Crew Scheduling Problem: Preference-based visit clustering and temporal dependencies. European Journal of Operational Research, 219(3), 598–610. https://doi.org/10.1016/j.ejor.2011.10.048

Ropke, S., & Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science, 40(4), 455–472. https://doi.org/10.1287/trsc.1050.0135

Ropke, S., Cordeau, J. F., & Laporte, G. (2007). Models and branch-and-cut algorithms for pickup and delivery problems with time windows. Networks, 49(4), 258–272. https://doi.org/10.1002/net.20177

Salama, M., & Srinivas, S. (2020). Joint optimization of customer location clustering and drone-based routing for last-mile deliveries. Transportation Research Part C:

Emerging Technologies, 114, 620–642. https://doi.org/10.1016/j.trc.2020.01.019

Sampaio, A., Savelsbergh, M., Veelenturf, L. P., & Van Woensel, T. (2020a). Delivery systems with crowd-sourced drivers: A pickup and delivery problem with transfers. Networks, 76(2), 232–255. https://doi.org/10.1002/net.21963

Song, B. D., & Ko, Y. D. (2016). A vehicle routing problem of both refrigerated- and general-type vehicles for perishable food products delivery. Journal of Food Engineering, 169, 61–71. https://doi.org/10.1016/j.jfoodeng.2015.08.027

Stohy, A., Abdelhakam, H.-T., Ali, S., Elhenawy, M., Hassan, A. A., Masoud, M., Glaser, S., & Rakotonirainy, A. (2021). Hybrid pointer networks for traveling salesman problems optimization. PLOS ONE, 16(12), e0260995. https://doi.org/10.1371/journal.pone.0260995

Sun, Y., Ernst, A., Li, X., & Weiner, J. (2021). Generalization of machine learning for problem reduction: a case study on travelling salesman problems. OR Spectrum, 43(3), 607–633. https://doi.org/10.1007/s00291-020-00604-x

Tiniç, G. O., Karasan, O. E., Kara, B. Y., Campbell, J. F., & Ozel, A. (2023). Exact solution approaches for the minimum total cost traveling salesman problem with multiple drones. Transportation Research Part B: Methodological, 168, 81–123. https://doi.org/10.1016/j.trb.2022.12.007

Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer Networks. Advances in Neural Information Processing Systems, 28. http://arxiv.org/abs/1506.03134

Voigt, S., & Kuhn, H. (2021). Crowdsourced logistics: The pickup and delivery problem with transshipments and occasional drivers. Networks, (April), 1–24. https://doi.org/10.1002/net.22045

Wolfinger, D. (2021). A Large Neighborhood Search for the Pickup and Delivery Problem with Time Windows, Split Loads and Transshipments. Computers and Operations Research, 126, 105110. https://doi.org/10.1016/j.cor.2020.105110

Wolfinger, D., & Salazar-González, J. J. (2021). The Pickup and Delivery Problem with Split Loads and Transshipments: A Branch-and-Cut Solution Approach. European Journal of Operational Research, 289(2), 470–484.

https://doi.org/10.1016/j.ejor.2020.07.032

Wu, Y., Song, W., Cao, Z., Zhang, J., & Lim, A. (2022). Learning Improvement Heuristics for Solving Routing Problems. IEEE Transactions on Neural Networks and Learning Systems, 33(9), 5057–5069. https://doi.org/10.1109/TNNLS.2021.3068828

Xie, F., Potts, C. N., & Bektaş, T. (2017). Iterated local search for workforce scheduling and routing problems. Journal of Heuristics, 23(6), 471–500. https://doi.org/10.1007/s10732-017-9347-8

Xin, L., Song, W., Cao, Z., & Zhang, J. (2021). NeuroLKH: Combining Deep Learning Model with Lin-Kernighan-Helsgaun Heuristic for Solving the Traveling Salesman Problem. Advances in Neural Information Processing Systems, 34, 7472–7483. http://arxiv.org/abs/2110.07983

Yalçındaˇg, S., Matta, A., ¸Sahin, E., & Shanthikumar, J. G. (2014). A two-stage approach for solving assignment and routing problems in home health care services. In Proceedings of the international conference on health care systems engineering (pp. 47–59). Springer, Cham. https://doi.org/10.1007/978-3-319-01848-5

Yu, J., & Hoff, R. (2013). Optimal routing and assignment of consultants for Energy Education, Inc. Interfaces, 43(2), 142–151. https://doi.org/10.1287/inte.1120.0656

Yuan, B., Liu, R., & Jiang, Z. (2015). A branch-and-price algorithm for the home health care scheduling and routing problem with stochastic service times and skill requirements. International Journal of Production Research, 53(24), 7450–7464. https://doi.org/10.1080/00207543.2015.1082041

Zamorano, E., & Stolletz, R. (2017). Branch-and-price approaches for the Multiperiod Technician Routing and Scheduling Problem. European Journal of Operational Research, 257(1), 55–68. https://doi.org/10.1016/j.ejor.2016.06.058

Zhang, R., Zhang, C., Cao, Z., Song, W., Tan, P. S., Zhang, J., Wen, B., & Dauwels, J. (2023). Learning to Solve Multiple-TSP With Time Window and Rejections via Deep Reinforcement Learning. IEEE Transactions on Intelligent Transportation Systems, 24(1), 1325–1336. https://doi.org/10.1109/TITS.2022.3207011

Zheng, J., He, K., Zhou, J., Jin, Y., & Li, C.-M. (2021). Combining Reinforcement

Learning with Lin-Kernighan-Helsgaun Algorithm for the Traveling Salesman Problem. Proceedings of the AAAI Conference on Artificial Intelligence, 35(14), 12445–12452. https://doi.org/10.1609/aaai.v35i14.17476

Zheng, J., He, K., Zhou, J., Jin, Y., & Li, C.-M. (2023). Reinforced Lin–Kernighan–Helsgaun algorithms for the traveling salesman problems. Knowledge-Based Systems, 260, 110144. https://doi.org/10.1016/j.knosys.2022.110144

Vita

Zefeng Lyu was born on November 21, 1994, in Zhejiang, China. He graduated from Zhejiang University of Technology with a Bachelor of Science degree in Industrial Engineering. In 2018, he moved to the United States and began his Ph.D. program at the University of Tennessee, Knoxville (UTK), majoring in Industrial Engineering with an Interdisciplinary Graduate Minor in Computational Science. During his studies and work at UTK, his primary focus was on innovative optimization methods, machine learning, and reinforcement learning, with applications in logistics, transportation, scheduling and planning, and infrastructure maintenance. His research findings have been published in journals such as the European Journal of Operations Research and Computers & Industrial Engineering.