



**SIMULTANEOUS OPERATION OF TRUCKS AND
UNMANNED AERIAL VEHICLES AT PACKAGE
DELIVERY**

BAYBARS İBROŞKA

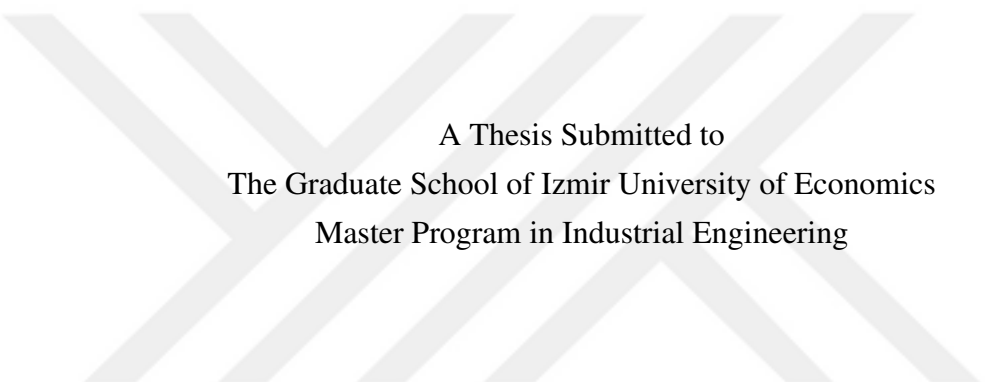
Master's Thesis

Graduate School
Izmir University of Economics

Izmir
2021

SIMULTANEOUS OPERATION OF TRUCKS AND UNMANNED AERIAL VEHICLES AT PACKAGE DELIVERY

BAYBARS İBROŞKA



A Thesis Submitted to
The Graduate School of Izmir University of Economics
Master Program in Industrial Engineering

İzmir
2021

ABSTRACT

SIMULTANEOUS OPERATION OF TRUCKS AND UNMANNED AERIAL VEHICLES AT PACKAGE DELIVERY

İbroşka, Baybars

Industrial Engineering Master Program

Advisor: Assoc. Prof. Dr. Selin Özpeynirci

August, 2021

With the developing technologies, the use of unmanned aerial vehicles has become more popular. Although it was previously used only for military purposes, with the acceleration of its development, its usage area has expanded and it has started to appear in sectors such as health, agriculture, transportation, mapping and communication. While shopping on the Internet previously appealed to a certain audience, the number of e-commerce users has increased significantly with the pandemic. As a result, with the increase in the number of people and points that cargo trucks will reach, various difficulties have emerged in terms of planning in transportation. One way to overcome all these difficulties is to make the use of unmanned aerial vehicles more widespread in this sector with proper planning and control, thus shortening the long distance that trucks will travel as much as possible. In this study, we consider a simultaneous operation of trucks and unmanned aerial vehicles at package delivery. We develop a General Variable Neighborhood Search algorithm and compare the results with the existing studies in the literature. Computational experiments show that our approach can find very good solutions in reasonable times and outperforms the existing methods in most instances.

Keywords: Transportation, Routing, Drone delivery, Unmanned aerial vehicles, Variable Neighborhood Search.

ÖZET

PAKET TESLİMATINDA KAMYON VE İNSANSIZ HAVA ARACININ EŞ ZAMANLI ÇALIŞMASI

İbroşka, Baybars

Endüstri Mühendisliği Yüksek Lisans Programı

Tez Danışmanı: Doç. Dr. Selin Özpeynirci

Ağustos, 2021

Gelişen teknolojiler ile beraber insansız hava aracı kullanımı daha da popülerleşmiştir. Önceden sadece askeri amaçlarla kullanılsa da gelişiminin hızlanmasıyla beraber kullanım alanı genişleyerek sağlık, tarım, taşımacılık, haritalama ve iletişim gibi sektörlerde de görünmeye başlanmıştır. İnternet üzerinden alışveriş, önceden belirli bir kitleye hitap ederken pandemi ile birlikte e-ticaret kullanıcı sayısı ciddi şekilde artmıştır. Bunun sonucunda kargo kamyonlarının ulaşacağı kişi ve nokta sayısının artmasıyla beraber taşımacılıkta planlama bakımından çeşitli zorluklar ortaya çıkmıştır. Tüm bu zorlukları aşmanın bir yolu da insansız hava aracı kullanımını, bu sektörde uygun planlama ve kontrol ile daha yaygınlaştırarak kamyonların gideceği uzun mesafeyi mümkün olduğunca kısaltmaktır. Bu çalışmada, paket teslimatında kamyon ve insansız hava araçlarının eşzamanlı çalışmasını ele alıyoruz. Geliştirdiğimiz Genel Değişken Komşuluk Arama (General Variable Neighborhood Search) algoritması literatürdeki mevcut çalışmalarla karşılaştırılmıştır. Yapmış olduğumuz deneyler, yaklaşımımızın makul zamanlarda çok iyi çözümler bulabildiğini ve çoğu durumda mevcut yöntemlerden daha iyi performans gösterdiğini gösteriyor.

Anahtar Kelimeler: Ulaşım, Rotalama, Drone teslimatı, İnsansız hava araçları, Değişken Komşuluk Araması.

To my family...



ACKNOWLEDGEMENT

First and foremost I am extremely grateful to my supervisors, Assoc. Prof. Dr. Selin Özpeynirci and Assoc. Prof. Dr. Özgür Özpeynirci for their advice, support, and patience during my study. I hope we get the chance to work together again in the future.

I would like to express my gratitude to Izmir University of Economics and the HAVI family for their support during my studies.

I would like to express my deepest gratitude to my girlfriend and best friend Leyli Buse Öztürk, I always felt her love and support during these difficult times. Without her, this process would have been much more difficult for me.

Finally, I would like to thank my parents Nuran İbroşka and Sezai İbroşka, they have been my biggest supporters every day throughout my life, and my twin İdil İbroşka, she is the best friend I could have on this journey. I am so lucky to have you all.

TABLE OF CONTENTS

ABSTRACT	ii
ÖZET	iii
ACKNOWLEDGEMENT	v
TABLE OF CONTENTS	vi
LIST OF TABLES	viii
LIST OF FIGURES	ix
CHAPTER 1: INTRODUCTION	1
1.1 <i>UAV Technology in Practice</i>	1
1.2 <i>Problem Statement</i>	2
1.3 <i>Purpose of the Study</i>	3
1.4 <i>Structure of the Thesis</i>	4
CHAPTER 2: LITERATURE REVIEW	5
2.1 <i>Travelling Salesperson Problem and Vehicle Routing Problem</i>	5
2.2 <i>Drone Extension of TSP and VRP</i>	6
CHAPTER 3: METHODOLOGY	10
3.1 <i>Mathematical Model for Single Truck & Single Drone Tandem</i>	10
3.2 <i>Multiple Travelling Salesperson Problem with Drones (mTSPD)</i>	12
3.2.1 <i>Mathematical Model</i>	13
3.2.2 <i>Solution Approach</i>	15
3.2.3 <i>Proposed General Variable Neighborhood Search Approach</i>	19
3.3 <i>Lower Bound</i>	35
3.3.1 <i>Case-1</i>	35
3.3.2 <i>Case-2.A</i>	35
3.3.3 <i>Case-2.B</i>	36
3.3.4 <i>LB Calculation</i>	36
CHAPTER 4: EXPERIMENTAL RESULTS	38
4.1 <i>Datasets</i>	38
4.2 <i>Results</i>	40
4.2.1 <i>Examination of the Results Based on the Number of Trucks</i>	40
4.2.2 <i>Examination of the Results Based on the Instance Types</i>	41
4.2.3 <i>Examination of the Results Based on the Number of Nodes</i>	42
4.2.4 <i>Graphical Comparison of Results</i>	43
4.2.5 <i>Small Instances</i>	46
4.3 <i>Lower Bounds of Problem Sets</i>	46
4.4 <i>Dashboard</i>	48

CHAPTER 5: CONCLUSION	53
REFERENCES	56



LIST OF TABLES

Table 1.	Classification of Truck and Drone Delivery Problems	9
Table 2.	Order of Neighborhoods Used in GVNS	38
Table 3.	Number of Best Solutions According to the Number of Trucks . . .	40
Table 4.	Average Gaps According to the Number of Trucks	41
Table 5.	Number of Best Solutions According to Instance Type	41
Table 6.	Average Gaps According to Instance Type	42
Table 7.	Number of Best Solutions According to the Number of Nodes . . .	42
Table 8.	Average Gaps According to the Number of Nodes	43
Table 9.	Comparison of Small Instances	46
Table 10.	Lower Bounds of Small Sized Problems	47
Table 11.	Lower Bounds of Medium and Large Sized Problems	48
Table 12.	The Results of The Algorithms in Instances with 25 Nodes	50
Table 13.	The Results of The Algorithms in Instances with 50 Nodes	51
Table 14.	Deviations of Algorithms from the Best Results in Instances with 25 Nodes	52
Table 15.	Deviations of Algorithms from the Best Results in Instances with 50 Nodes	52

LIST OF FIGURES

Figure 1.	Sample Solution Representations	3
Figure 2.	Basic Variable Neighborhood Search Algorithm	17
Figure 3.	Reduced Variable Neighborhood Search Algorithm	17
Figure 4.	Variable Neighborhood Descent Algorithm	19
Figure 5.	General Variable Neighborhood Search Algorithm	19
Figure 6.	Proposed General Variable Neighborhood Search Algorithm	20
Figure 7.	Truck Vector	21
Figure 8.	Drone Vector	21
Figure 9.	Nearest Neighbor Algorithm	22
Figure 10.	Calculate Arrival Time	23
Figure 11.	Cost Evaluation	24
Figure 12.	Shake Phase	25
Figure 13.	Union Variable Neighborhood Descent	26
Figure 14.	U-VND Neighborhoods	27
Figure 15.	swapMove Neighborhood	28
Figure 16.	visitorSwap Neighborhood	29
Figure 17.	swap Neighborhood	30
Figure 18.	oneMove Neighborhood	31
Figure 19.	departure Neighborhood	32
Figure 20.	landing Neighborhood	33
Figure 21.	droneAdd Neighborhood	34
Figure 22.	Visualization of Cases Used in Lower Bound Calculation	35
Figure 23.	Flowchart of the Proposed General Variable Neighborhood Search Algorithm	37
Figure 24.	Dataset Visualisation	39
Figure 25.	Comparison of Results for Instances with 25 Nodes	44
Figure 26.	Comparison of Results for Instances with 50 Nodes	45
Figure 27.	Dashboard Overview	49
Figure 28.	Dashboard Details	49

CHAPTER 1: INTRODUCTION

In this chapter, firstly we explain the usage areas of unmanned aerial vehicles (that will be referred to as UAV or drone interchangeably) in practice, then in the next section we define the problem that this thesis focuses on, and in the final section we present the general structure of the thesis.

1.1 UAV Technology in Practice

In the world and in our country, one of the first factors to consider when changing and developing technologies are examined is the use of unmanned aerial vehicles. The impact of this growing and spreading technology is clearly noticeable. It has been observed that unmanned aerial vehicles are designed in different ways, primarily in military areas for use in various operations. Afterwards, this rapidly developing technology has been used in communication, health, mapping, agriculture and transportation fields.

Considering today's conditions, when we consider the transportation sector, it is seen that cargo transportation has a big place in our lives with the effect of e-commerce. Nowadays, with the advancing technology and due to pandemic, people make their purchases over the internet. This situation leads to an increase in the number of destination points that the cargo transporters are trying to reach and also increase in distances, decreasing the package sizes and increasing the delivery frequency. As a result, planning of transportation operations has become more difficult. One way to solve these problems is to make unmanned aerial vehicles more widespread in this sector and to ensure that trucks travel less by appropriate planning.

There are several companies, including Amazon, which predict this problem and test the use of unmanned aerial vehicles in transportation. As an example, Amazon Prime Air delivers directly to the destinations that are particularly difficult to reach. In addition to this, UPS aims to make delivery with the use of cargo truck together with the unmanned aerial vehicle set up on the top of the truck. In this way, the unmanned aerial vehicle delivers to a point, while the cargo truck is delivering a separate point, aiming to speed up the delivery.

1.2 Problem Statement

Package delivery with the UAV's, which we will begin to see in package transportation with the developing technology, results in many new routing problems. UAV included problems appear as an extension of Vehicle Routing Problem (VRP) with capacity constraints, or as Travelling Salesperson Problem (TSP) extensions that ignore the capacity of trucks and assume that UAV's can carry one package per delivery. TSP extensions have more examples than VRP extensions in the literature. The reason for this is because of the decrease of package sizes due to the effect of e-commerce, capacities are often neglected. We can categorize these problems based on the drone's ability to move, or in other words, its capability of departure and landing on nodes during the transport operation. If we consider the first category as scenarios where deliveries are made directly from the warehouse, we can give Amazon Prime Air as an example. Another category can be defined as the situation in which the truck and drone deliver simultaneously, which is being tested by UPS. On the other hand, the number of trucks and drones used in the package transport operation also causes differentiation of problems and increases diversity.

In this thesis, we focus on the UAV delivery problem which is an extension of the TSP. In the Classic TSP, the salesperson is expected to deliver to n points at the lowest cost. The cost here refers to the total distance traveled. In our case, assuming the salesperson is the cargo truck, we can define all n points as customers waiting for package delivery. In addition, with the involvement of a drone into the delivery operation, it is expected that either truck or drone will deliver to each point at the lowest cost. The cost here can be considered as the total distance traveled or total time required for delivering all packages.

We can take this problem to a different dimension by increasing the number of trucks and drones used in the transport operation. We may encounter different types of problems where there are one truck and more than one drones, multiple trucks and one drone, or multiple trucks and multiple drones in the system. In addition to the number of trucks and drones used in delivery, the mobility of the drone is another important factor affecting the problem environment. Following cases may be considered regarding the mobility of the drones: the drones are only allowed to depart and land on the warehouse, the drones are only allowed to depart and land on the truck, the drones can depart from a truck and land on different truck when there are multiple trucks. Finally, problems with multiple trucks and drones where the drones can land

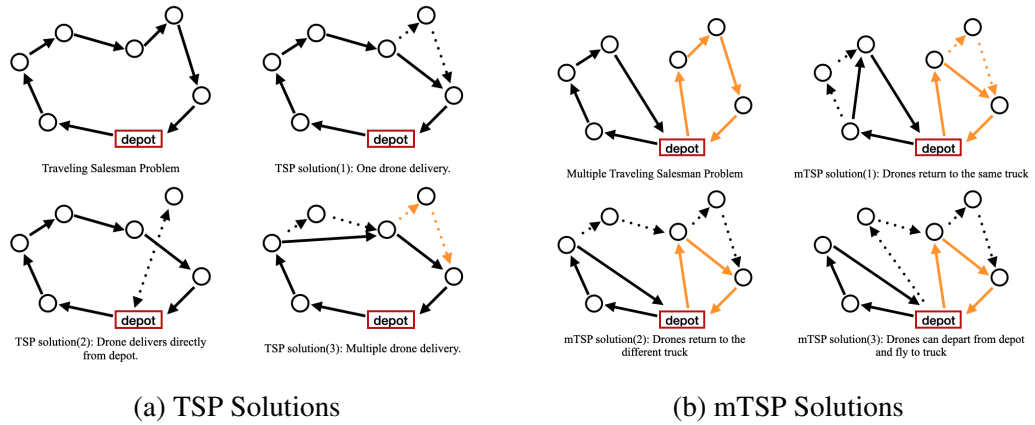


Figure 1: Sample Solution Representations

on or departure from any vehicle or warehouse. We can observe the visualization of these problem types in Figure 1.

In this thesis, we focus on Multiple Traveling Salesperson with Drones (mTSPD), which has a few examples in the literature and is a complex problem, including multiple trucks and multiple drones, and the drones are allowed to depart from a truck or the warehouse, and land on the same truck, a different truck or the warehouse.

1.3 Purpose of the Study

The aim of this thesis is to offer solutions for the problems found in the literature about the simultaneous operation of truck and UAV for package delivery. In line with the objectives, new solution proposals for the problems were developed and computational experiments and tests related to the subject were carried out.

When we examined the literature in line with our study, we realized that the problems involving multiple trucks and multiple drones were rarely included in the literature, and we have directed our work specifically on this subject. The most comprehensive problem found in the literature is the Multiple Traveling Salesperson with Drones (mTSPD) problem proposed by Kitjacharoenchai et al. (2019). In this way, we steered our work by trying to achieve robust solutions with the General Variable Neighborhood Search (GVNS) heuristic that we developed. We tested and compared our experiments on datasets shared by Kitjacharoenchai et al. (2019).

1.4 Structure of the Thesis

In the next chapters, we cover the following topics. In Chapter 2 we present the literature review which consists TSP and VRP variants that drone and truck delivery problems are based on or have similar characteristics, also the different problem types of drone and truck delivery problems, and the studies carried out for these problems. In Chapter 3, we first present the mathematical model developed for a single truck and a single drone tandem. Then, we present the model and problem definition of the mTSPD problem, which is the main focus of the study, and the General Variable Neighborhood Search (GVNS) algorithm that we developed to solve the problem. We report the experimental results of the proposed General Variable Neighborhood Search (GVNS) algorithm in Chapter 4. Lastly in final Chapter 5, we state the summary of the study and future future work.

CHAPTER 2: LITERATURE REVIEW

In this chapter, we describe our literature review under 2 different sections. First, we give brief information about the TSP and VRP problems that lead to the problems where the drone and truck work simultaneously. Next, we present the problems that the drone and the truck deliver simultaneously and the solutions offered to these problems.

2.1 Travelling Salesperson Problem and Vehicle Routing Problem

In this section, we examine the relationship of routing problems involving the drone with other routing problems already found in the literature.

The package delivery tandem of truck and drone is relevant to the Covering Traveling Salesperson (CSP) in that the truck is not required to deliver to every customer. CSP is first proposed by Current and Schilling (1989), CSP is a generalized version of TSP that can be applied in many different fields and different problems. CSP aims to travel the least costly route keeping all customers within a predetermined area, but the truck does not have to visit every destination. This definition is an approach that can be taken as a basis in biomodal delivery systems, just like in scenarios where truck and drone deliver together except drone has to visit remaining customers.

In addition, if we consider the drone as a detachable part of the truck, we can find Vehicle Routing Problem (VRP) extensions suitable for this definition in the literature. In the Truck and Trailer Routing Problem (TTRP), customers have vehicle type constraints on which certain vehicles can be shipped. These are divided into following types, truck and trailer can deliver together or only truck can deliver. In cases where only the truck can enter, the truck must detached with the trailer at certain points and deliver on its own so that it can ship to customers that only the truck can visit. Chao (2002) has provided a solution to this problem with the Tabu Search method and Lin et al. (2009) with the Simulated Annealing approach to solve this problem.

Lin (2011) presents a study with similar characteristics and limitations to the simultaneous operation of a drone and truck. The study deals with the pickup and delivery problem, the operation where a heavy vehicle can carry a light vehicle in addition to the load. A light vehicle can pick up and deliver a load independently from a heavy vehicle, and a light vehicle can also travel with a heavy vehicle or independently. These constraints can be given as examples of the constraints found in problem types

consisting of drones and trucks.

2.2 Drone Extension of TSP and VRP

There are various studies regarding problems on truck and drone delivery. We can categorize the problems according to their drone and truck tandem or drone movement flexibility.

Dorling et al. (2016) propose, Drone Delivery Problem (DDP) as an extension of VRP. They present the DDP problem as two different mathematical models that vary according to its objective function. The first of these is MC-DDP, it minimizes the cost, the other mathematical model named as MT-DDP, which minimizes time in case of emergency. Sub-optimal results were aimed using Simulated Annealing, as this problem is difficult to solve optimally even in conditions with a small number of customers. While constructing Simulated Annealing, they also presented the DDP Cost Function, which includes factors such as the weight and cost of batteries and the load carried by drones.

Wang et al. (2017) propose Vehicle Routing Problem with Drones, and studied many different worst-case scenarios on VRP-D. They also propose several upper bounds on the ratio of time saving by using drones against other traditional routing problems that use only truck in the routes. Poikonen et al. (2017) extends their previous work by integrating the following factors: the battery life of the drone, cost metrics, and the fixed cost of deploying drones.

Murray and Chu (2015) define the Flying Sidekick Travelling Salesman Problem (FSTSP) that investigates simultaneous delivery of one truck and one drone tandem. If the drone ends its delivery by landing at the depot, it can not launch again from the depot, since FSTSP's motivation is the use of drones where depots are far from the delivery nodes. They also define another problem called the Parallel Drone Scheduling Travelling Salesman Problem (PDSTSP). In PDSTSP there is a fleet of one truck and one or more homogeneous drones. Drones are not binded with truck, thus they operate independently from trucks and directly deliver from depot to nodes. They propose mathematical models for FSTSP and PDSTSP; however they cannot find optimal solutions with their models and propose heuristics for each problem. For FSTSP, they present route and re-assign heuristic, where they first find TSP solution and then use a re-assign algorithm to add drones to the solution. They also present the numerical analysis of impact of different TSP heuristics to the solutions.

Ponza (2016) examined FSTSP in detail and investigated "Simulated Annealing", "Ant Colony Optimization" and "Naive Approach" as possible solution approaches for FSTSP. Among these solutions, he carried out experiments by using "Simulated Annealing" and presented a new instance.

Ferrandez et al. (2016) compare energy consumption of system by investigating the drone and truck speeds. They use Genetic Algorithm to find best route for TSP. They develop K -means algorithm to find launch locations of drones.

Since FSTSP can not find optimal solutions even for six node instances, Agatz et al. (2018) propose a new problem called TSP-D with a different mathematical formulation and they can find optimal solutions up to twelve nodes. In TSP-D, only one drone and one truck deliver simultaneously like FSTSP, however a drone can only launch from a truck and land into a truck in customer nodes. Also a truck can visit the same node more than once, which helps to increase drone usage by increasing the launching and landing nodes. Regarding NP-hardness of the problem, they propose route-first-cluster-second algorithm for TSP-D problem. They first obtain a TSP solution -which is also feasible for TSP-D when we consider no drone deliveries exist- then they divide the solution by assigning some nodes to drone and some nodes to truck. In the assignment phase, they use two different heuristics which are "A Greedy Partitioning Heuristic" and "An Exact Partitioning Algorithm". Bouman et al. (2018) present another solution approach to the TSP-D problem by using dynamic programming.

de Freitas and Penna (2020) propose a hybrid heuristic for FSTSP to improve solutions of instances from Ponza (2016) and also they rearrange their algorithm for TSP-D from Agatz et al. (2018). In their hybrid heuristic, they first use MILP TSP solver to find TSP solutions, then they use General Variable Neighborhood Search (GVNS) algorithm to improve solutions. Ha et al. (2018) propose a different variant of TSP-D which extends from FSTSP. Authors develop mathematical formulation and two different heuristics for min-cost TSP-D.

Murray and Raj (2020) define a new problem called The Multiple Flying Sidekicks Travelling Salesman Problem (mFSTSP), an extension of FSTSP. The mFSTSP consists of one truck and multiple heterogeneous drone fleet. Each package must be delivered by either a truck or a drone. Each customer has different drone acceptability, with the aim of achieving a solution for more realistic situations. They propose a

mathematical formulation and a three-phased heuristic solution approach to achieve solutions.

Regarding the drone delivery problems which extended from TSP, Kitjacharoenchai et al. (2019) propose a problem that has multiple drones and multiple trucks and has the most flexible drone movement named as Multiple Travelling Salesman Problem with Drones (mTSPD). In mTSPD, drones can depart from the depot or from a truck. Also drones can land into the depot or any of the trucks, not necessarily on the one they have departed from.

We present the classification of truck and drone delivery problems in Table 1.

In this thesis, we consider the problem defined by Kitjacharoenchai et al. (2019) and develop a variable neighborhood search heuristic with the aim of obtaining near-optimal solutions in reasonable time.

Table 1: Classification of Truck and Drone Delivery Problems

Authors	Problem Type	Movement Flexibility of Drone										Solution Method			
		One Truck		One Drone		Multiple Drones		Multiple Drones		Drone Departure Point		Drone Landing Point		Mathematical Model	Heuristic
		One Drone	Multiple Drones	One Drone	Multiple Drones	Multiple Drones	Multiple Drones	Depot	Truck	Depot	Truck	Different Truck			
Kitjacharoenchai et al. (2019)	mTSPD				X			X	X	X	X	X	X	X	GA + ADI
Murray and Raj (2020)	mFSTSP					X		X	X	X	X	X	X	X	Heuristic Approach
Murray and Chu (2015)	FSTSP, PDSTSP	X						X	X	X	X	X	X	X	Heuristic Approach
Ponza (2016)	FSTSP	X						X	X	X	X	X	X	X	Simulated Annealing
Agatz et al. (2018)	TSP-D	X						X	X	X	X	X	X	X	Heuristic Approach
Bouman et al. (2018)	TSP-D	X						X	X	X	X	X	X	X	Dynamic Programming
Ha et al. (2018)	TSP-D	X						X	X	X	X	X	X	X	Heuristic Approach
Ferreira et al. (2016)	TSP-D	X				X		X	X	X	X	X	X	X	K-means & GA
Freitas and Penna (2020)	FSTSP, TSP-D	X						X	X	X	X	X	X	X	Hybrid-GVNS
Dorling et al. (2016)	DDP	X						X	X	X	X	X	X	X	Simulated Annealing
Wang et al. (2017)	VRPD														Theorems
Poikonen et al. (2017)	VRPD														Theorems

CHAPTER 3: METHODOLOGY

In this chapter, we first present the mathematical model developed for a single truck and a single drone tandem, for better understanding the problems that we mentioned in Chapter 1 and Section 2.2. Next, we present the Multiple Traveling Salesperson Problem with Drones (mTSPD) defined by Kitjacharoenchai et al. (2019), which is the main focus of this study. Finally, we explain in detail the General Variable Neighborhood Search (GVNS) approach we proposed for the mTSPD problem.

3.1 Mathematical Model for Single Truck & Single Drone Tandem

We developed a mathematical model that covers TSP with drone problem. In our mathematical model, drone can depart from truck while truck is delivering to customer. However, truck must wait in delivery point until drone comes back from its delivery. c_{ij} (cost of truck delivery from i to j) is calculated by Manhattan Distance Formula and cd_{ij} (cost of drone delivery from i to j) is calculated by Euclidian Distance Formula, since drones do not need to follow the road, and can fly over the buildings and other barriers.

Sets

$i, j = 1 \dots n$ represents delivery points

Parameters

c_{ij} : cost of truck delivery from i to j .

cd_{ij} : cost of drone delivery from i to j .

Decision Variables

$$x_{ij} = \begin{cases} 1, & \text{if truck delivers a package to node } j \text{ using arc } (i, j) \\ 0, & \text{otherwise} \end{cases}$$

$$q_{ij} = \begin{cases} 1, & \text{if drone delivers a package to node } j \text{ using arc } (i, j) \\ 0, & \text{otherwise} \end{cases}$$

$$Tr_i = \begin{cases} 1, & \text{if truck delivers to node } i \\ 0, & \text{otherwise} \end{cases}$$

$$D_i = \begin{cases} 1, & \text{if drone delivers to node } i \\ 0, & \text{otherwise} \end{cases}$$

u_i : sub-route elimination variable

$$\text{Minimize } Z = \sum_{i=1}^n \sum_{j=1}^n (x_{ij} * c_{ij} + q_{ij} * cd_{ij} * 2) \quad (3.1)$$

Subject to

$$Tr_i + D_i = 1 \quad \forall i \quad (3.2)$$

$$\sum_{j=1}^n x_{ij} = Tr_i \quad \forall i | i \neq j \quad (3.3)$$

$$\sum_{i=1}^n x_{ij} = Tr_j \quad \forall j | i \neq j \quad (3.4)$$

$$\sum_{j=1}^n q_{ij} \leq 1 \quad \forall i \quad (3.5)$$

$$\sum_{i=1}^n q_{ij} = D_j \quad \forall j \quad (3.6)$$

$$q_{ij} \leq Tr_i \quad \forall i, j \quad (3.7)$$

$$u_i - u_j + n * x_{ij} \leq (n - 1) \quad \forall i, j | i > j \quad (3.8)$$

$$1 \leq u_i \leq n \quad \forall i | i > 1 \quad (3.9)$$

$$x_{ij}, q_{ij}, Tr_i, D_i \in \{1, 0\} \quad \forall i, j \quad (3.10)$$

$$u_i \geq 0 \quad \forall i \quad (3.11)$$

Objective function (3.1) minimizes the total distance travelled by drone and truck. Constraint (3.2) guarantees that each delivery point must be visited by either the truck or the drone. Restrictions (3.3) and (3.4) limit that if truck delivers to a node, there can be only one way to enter to the node and one way to exit from the node. Constraint (3.5) guarantees that drone can only return using just one arc. Constraint (3.6) guarantees that drone can use arc (i, j) only if node j is delivered by drone. Constraint (3.7) forces the drone to return back to the node that it departures from. Restrictions (3.8) and (3.9) are subtour elimination constraints. Constraints 3.10 and 3.11 are set constraints.

3.2 Multiple Travelling Salesperson Problem with Drones (mTSPD)

The mTSPD arises from the idea of simultaneous operation of trucks and drones for package delivery. As seen in Figures 1a and 1b, drones can depart from any truck or depot, they can also land on any truck or directly to depot. Nodes can be visited via a truck or a drone, but not both. Drones can only merge with trucks on nodes; they are not allowed to merge while trucks are moving. Whichever vehicle comes first to the meeting points has to wait until the other vehicle arrives. At each node, there can be only one drone landing or departure operation.

The mathematical model is built on FSTSP model, introduced by Murray and Chu (2015), with additional constraints regarding the definition of the problem. In mTSPD, the objective of problem is minimizing the arrival time of the vehicle which arrives latest to the depot. In mTSPD, there is only one depot, however they define the depot as two different nodes, refereed to as starting depot ($0(s)$) and ending depot ($0(r)$) in the mathematical model. Since there are multiple trucks in mTSPD, m trucks depart from $0(s)$ and visit n nodes with drones, then they have to return to $0(r)$. Set of customers is defined as $C = \{1, 2, 3, 4, \dots, n\}$. Additionally, $C_0 = C \cup \{0(s)\}$ is defined as the set of customers that includes starting depot and $C_+ = C \cup \{0(r)\}$ as the set of customers that includes ending depot. Travel times on the arcs for trucks and drones are different due to the structure and mobility of the vehicles. Truck travel time is defined as T_{ij}^T on arc (i, j) and drone travel time as T_{ij}^D on arc (i, j) . $F = \{(i, j, k)\}$ is used to define all possible node combinations that drone can use for delivery.

In addition, the following assumptions are considered:

- The departure node i cannot be ending depot. ($i \neq 0(r)$ or $i \in C_0$)
- The delivery node j cannot be equal to the departure node i . Additionally, delivery node j must be in customer set C . ($i \neq j, j \in C$)
- The landing node k cannot be equal to either departure node i or delivery node j , and landing node j must be an element of the set C_+ . ($k \neq i, k \neq j, k \in C_+$)

The following decision variables are defined: x_{ij} is equal to 1, if truck uses arc (i, j) to deliver node j , 0 otherwise ($i \in C_0, j \in C_+$). y_{ijk} is equal to 1, if drone departure from i to deliver node j and lands into node k , 0 otherwise ($i \in C_0, j \in C, k \in C_+$). T_j is defined as the truck arrival time at node j and similarly, D_j is used to denote the drone arrival time at node j ($j \in C_+$). Finally, u_i is a decision variable used for the subtour elimination constraints.

3.2.1 Mathematical Model

In this section, we present the mathematical model for mTSPD, developed by Kitjacharoenchai et al. (2019), for the sake of completeness.

Indices

i, j, k represent customers and depot.

Sets

C Set of customers $(1, 2, 3, \dots, n)$

C_0 Set of customer nodes including the starting depot, $C \cup 0(s)$

C_+ Set of customer nodes including the ending depot, $C \cup 0(r)$

F Set of all possible three-node sorties of the drone path

Parameters

$t_{i,j}^T$ Truck travel time between nodes i and j

$t_{i,j}^D$ Drone travel time between nodes i and j

m Number of trucks in the entire fleet

n Number of total customers to be visited

Decision Variables

Tl_i : Truck arrival time at node i

Dl_i : Drone arrival time at node i

$$x_{ij} = \begin{cases} 1, & \text{if a truck traverses arc } (i, j) \text{ from customer } i \text{ to customer } j \\ 0, & \text{otherwise} \end{cases}$$

$$y_{ijk} = \begin{cases} 1, & \text{if a drone traverses arcs } (i, j) \text{ and } (j, k) \\ 0, & \text{otherwise} \end{cases}$$

$$\text{Minimize } Z = Dl_{0(r)} \quad (3.12)$$

Subject to

$$\sum_{\substack{i \in C_0 \\ i \neq j}} x_{ij} + \sum_{\substack{i \in C_0 \\ i \neq j}} \sum_{\substack{k \in C_+ \\ (ijk) \in F}} y_{ijk} = 1 \quad \forall j \in C \quad (3.13)$$

$$\sum_{j \in C_+} x_{0(s),j} = m \quad (3.14)$$

$$\sum_{i \in C_0} x_{i,0(r)} = m \quad (3.15)$$

$$u_i - u_j + nx_{ij} + (n-2)x_{ji} \leq n-1 \quad \forall i, j \in C, i \neq j \quad (3.16)$$

$$1 + (n-2)x_{i,0(s)} + \sum_{\substack{j \in C \\ i \neq j}} x_{ij} \leq u_i \leq n - (n-2)x_{0(s),i} - \sum_{\substack{j \in C \\ i \neq j}} x_{ij} \quad \forall i \in C \quad (3.17)$$

$$\sum_{\substack{i \in C_0 \\ i \neq j}} x_{ij} = \sum_{\substack{k \in C_+ \\ k \neq j}} x_{jk} \quad \forall j \in C \quad (3.18)$$

$$\sum_{\substack{j \in C \\ i \neq j}} \sum_{\substack{k \in C_+ \\ (i,j,k) \in F}} y_{ijk} \leq 1 \quad \forall i \in C_0 \quad (3.19)$$

$$\sum_{\substack{i \in C_0 \\ i \neq k}} \sum_{\substack{j \in C \\ (i,j,k) \in F}} y_{ijk} \leq 1 \quad \forall k \in C_+ \quad (3.20)$$

$$2y_{ijk} \leq \sum_{\substack{h \in C_0 \\ h \neq i}} x_{hi} + \sum_{\substack{l \in C \\ l \neq k}} x_{lk} \quad \forall i, j \in C, \forall k \in C_+ \quad (3.21)$$

$$y_{0(s),j,k} \leq \sum_{\substack{h \in C_0 \\ h \neq k}} x_{hk} \quad \forall j \in C, \forall k \in C_+ \quad (3.22)$$

$$\sum_{\substack{i \in C_0 \\ i \neq j}} \sum_{\substack{k \in C_+ \\ (i,j,k) \in F}} y_{ijk} \leq 1 - \sum_{\substack{a \in C \\ a \leq j}} \sum_{\substack{b \in C_+ \\ (j,a,b) \in F}} y_{jab} \quad \forall j \in C \quad (3.23)$$

$$\sum_{\substack{i \in C_0 \\ i \neq j}} \sum_{\substack{k \in C_+ \\ (i,j,k) \in F}} y_{ijk} \leq 1 - \sum_{\substack{a \in C_0 \\ a \neq j}} \sum_{\substack{b \in C \\ (a,b,j) \in F}} y_{abj} \quad \forall j \in C \quad (3.24)$$

$$Dl_i \geq Tl_i - M \left(1 - \sum_{\substack{j \in C \\ i \neq j}} \sum_{\substack{k \in C_+ \\ (i,j,k) \in F}} y_{ijk} \right) \quad \forall i \in C \quad (3.25)$$

$$Dl_i \leq Tl_i + M \left(1 - \sum_{\substack{j \in C \\ i \neq j}} \sum_{\substack{k \in C_+ \\ (i,j,k) \in F}} y_{ijk} \right) \quad \forall i \in C \quad (3.26)$$

$$Dl_k \geq Tl_k - M \left(1 - \sum_{\substack{j \in C_0 \\ j \neq k}} \sum_{\substack{j \in C_+ \\ (i,j,k) \in F}} y_{ijk} \right) \quad \forall k \in C_+ \quad (3.27)$$

$$Dl_k \leq Tl_k + M \left(1 - \sum_{\substack{j \in C_0 \\ j \neq k}} \sum_{\substack{j \in C_+ \\ (i,j,k) \in F}} y_{ijk} \right) \quad \forall k \in C_+ \quad (3.28)$$

$$Tl_k \geq Tl_h + t_{hk}^T - M \left(1 - x_{hk} \right) \quad \forall h \in C_0, \forall k \in C_+ \quad (3.29)$$

$$Dl_k \geq Dl_i + t_{ij}^D + t_{jk}^D - M \left(1 - y_{ijk} \right) \quad \forall i \in C_0, \forall j \in C, \forall k \in C_+ \quad (3.30)$$

$$Dl_{0(s)} = 0 \quad (3.31)$$

$$Tl_{0(s)} = 0 \quad (3.32)$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j \in C \cup C_0 \cup C_+ \quad (3.33)$$

$$y_{ijk} \in \{0, 1\} \quad \forall i, j, k \in C \cup C_0 \cup C_+ \quad (3.34)$$

$$Dl_i \geq 0 \quad \forall i \in C \cup C_0 \cup C_+ \quad (3.35)$$

$$Tl_i \geq 0 \quad \forall i \in C \cup C_0 \cup C_+ \quad (3.36)$$

Objective (3.12) of the problem is to minimize the maximum length among all routes or it can be defined as to minimize the maximum time the last vehicle (drone or truck) returns to the depot. Constraint set (3.13) guarantees that each node is visited either by a truck or a drone. Constraints (3.14) and (3.15) make sure that the truck leaving the depot returns to the depot. Constraint sets (3.16) and (3.17) are the subtour elimination constraints. Constraint set (3.18) ensures that if a truck arrives at a node, it must leave that node. Constraint sets (3.19) and (3.20) guarantee that at most one drone can depart from or land onto a node (starting depot and ending depot are included). Constraint set (3.21) guarantees that trucks must visit the nodes where a drone lands or departs. Constraint set (3.22) ensures that if a drone lands on node k , then a truck has to arrive that node. Constraint set (3.23) does not allow more than one drone to follow the same route. Constraint set (3.24) makes sure that if a drone delivers to node j , no other drone can travel to that node. Constraint sets (3.25) and (3.26) ensure that departure time of drone and truck must be the same. Constraint sets (3.27) and (3.28) guarantee that if truck and drone merge on the same node, their arrival times must be the same. Constraints (3.25)-(3.28) ensure that if either truck or drone comes first to a node, it waits for the other to depart. Also, if a truck and drone merge at a node, their arrival and departure times will be the same. Constraint (3.29) and (3.30) calculates the truck and drone arrival time for each node, respectively. Constraints (3.31) and (3.32) ensure that truck and drone arrival times are 0 at the starting depot. Finally, constraints sets (3.33)-(3.36) are the set constraints.

3.2.2 Solution Approach

The mathematical model proposed by Kitjacharoenchai et al. (2019) can find the optimal solution to problems with at most 8 nodes. The optimal solutions to the problems with 25 nodes and 50 nodes defined by Kitjacharoenchai et al. (2019) could not be found in the time limit of one hour.

In this study, we propose a General Variable Neighborhood Search (GVNS) heuristic to find good quality solutions for mTSP with drones. We develop a general variable neighborhood search (GVNS) algorithm by using Union Variable Neighborhood

Descent (U-VND) in the local search / improvement phase to solve our problem. GVNS is a variant of Variable Neighborhood Search (VNS), which exceeds the limit of optimization possibilities in local search to overcome the complexity of mTSPD (Hansen et al., 2017). In the following sections, we explain the most commonly used variants of VNS for a better understanding of the approach that we use in this study.

3.2.2.1 Variable Neighborhood Search (VNS)

Variable Neighborhood Search (VNS) is a simple and effective metaheuristic that is widely used by researchers for solving combinatorial and global optimization problems. Its basic idea is systematic change of neighborhood both within a local search algorithm where there is a descent phase to find a local optimum and a perturbation phase to get out of the corresponding region (Mladenović and Hansen, 1997; Hansen and Mladenović, 2001; Hansen et al., 2017; Hansen and Mladenović, 2018). VNS diversifies into many variants based on how it is built and how it is applied.

3.2.2.2 Basic Variable Neighborhood Search (BVNS)

Basic VNS is proposed by Mladenović and Hansen (1997), consisting of 3 main parts called Shaking, Local Search and the part where it is decided how the neighborhood will change. Algorithm 2 shows the structure of the BVNS algorithm. In shaking step (line 4), it generates random x' from k th neighborhood to prevent getting stuck with local optima. In local search step (line 5), there are 2 methods to apply when searching for an improved solution, called *first improvement* and *best improvement*. In the *first improvement*, after the first improvement on solution x' , the heuristic stops running and algorithm moves on to the next step. In another method, *best improvement*, the best result obtained among all results obtained from solution x' in local search is found, and then algorithm moves on the next step which is *NeighborhoodChange* (line 6). In *NeighborhoodChange*, the algorithm compares incumbent solution x with the solution x'' which is obtained from *LocalSearch*. If x'' is better than x , x'' becomes the new incumbent solution ($x = x''$), and k will return to its starting value ($k \leftarrow 1$); otherwise the next k th neighborhood applies on next iteration. This process repeats until the ending condition is met. Ending condition can be defined as the total CPU time or total number of iterations for algorithm to run until there is no improvement, or various other stopping conditions can be defined.

```

1 repeat
2    $k \leftarrow 1$ 
3   while  $k \leq k_{max}$  do
4      $x' \leftarrow \text{Shake}(x, k)$ 
5      $x'' \leftarrow \text{LocalSearch}(x')$ 
6      $\text{NeighborhoodChange}(x, x'', k)$ 
7   end
8 until ending_condition met

```

Figure 2: Basic Variable Neighborhood Search Algorithm

3.2.2.3 Reduced Variable Neighborhood Search (RVNS)

Reduced Variable Neighborhood Search (RVNS) is a variant of VNS which is obtained discarding the *LocalSearch* step from BVNS (see Algorithm 2). Since local search is not included in RVNS, there will not be any descent made, and much faster results can be obtained. The reason for this is that the algorithm searches for new results by comparing the solution x' obtained randomly by using the k th neighborhood in the *Shake* (Algorithm 3, line 4) with incumbent solution x . Comparison is done in *NeighborhoodChange* step (line 5) as in BVNS. If a better solution is obtained, then solution x' becomes the new incumbent solution x ($x \leftarrow x'$), and k will return to the first value defined in the set of neighborhoods; otherwise the algorithm continues with next k th neighborhood. This fast structure of RVNS saves a significant amount of time when used in large-scale problems, and since fast and robust results are obtained in this way, its usage area is more common in large-scale problems.

```

1 repeat
2    $k \leftarrow 1$ 
3   while  $k \leq k_{max}$  do
4      $x' \leftarrow \text{Shake}(x, k)$ 
5      $\text{NeighborhoodChange}(x, x', k)$ 
6   end
7 until ending_condition met

```

Figure 3: Reduced Variable Neighborhood Search Algorithm

3.2.2.4 Variable Neighborhood Descent (VND)

Variable Neighborhood Descent (VND) is another common variant of VNS among researchers. VND differs from BVNS by doing a discrete search without using the *Shake* step (See Algorithm 2). Just like RVNS differs from BVNS by just doing a stochastic search. We can also express VND as BVNS without perturbation phase.

VND algorithm is based on the following facts:

- The local optimum reached by one neighborhood may not be the local optimum for the other neighborhoods.
- The local optimum reached by all neighborhoods is defined as the global optimum.

Since there is no perturbation phase in algorithm, VND needs more than one neighborhood in its structure to run effectively and to obtain robust solution, and the algorithm must be designed very well to avoid falling into the local optimum traps. According to empirical studies of Mjirda et al. (2016), there are 4 main decisions that affect the good functioning of the VND algorithm. As a result of these decisions, variants of VND occur and the structure of the algorithm is formed. Mjirda et al. (2016) point out these decisions as follows:

1. Run order of the neighborhoods in VND.
2. Deciding on search strategy. These are *first improvement* or *best improvement*. We mention about this search strategy in the section BVNS 3.2.2.2 where we explain *LocalSearch* step.
3. Moving strategy when improvement is achieved. This strategy names VND variants based on how neighborhoods change when improvement occurs.
 - Basic VND (B-VND): when improvement is achieved, it returns to the first defined neighborhood.
 - Pipe VND (P-VND): continues to run with the neighborhood where improvement takes place.
 - Cyclic VND (C-VND): when the improvement takes place, it continues to run with the next neighborhood in the neighborhood set.
 - Union VND (U-VND): in this variant, all defined neighborhoods operate as one whole neighborhood.
4. Obtaining initial solution. Considering that there is no perturbation in the VND, the initial solution can be critical not to drop to the local optimum too quickly.

When we evaluate these 4 decisions and VND variants that we mentioned, we can say that many different VND structures can be created for different problem types. Each variant has its own advantages and disadvantages. Mjirda et al. (2016) have carried out

a detailed empirical study comparing the performances of these variants on the TSP problem.

```

1 Define set of neighborhoods  $N_k$  ( $k = 1, \dots, k_{max}$ );
2  $k \leftarrow 1$ ;
3 repeat
4    $x' \leftarrow N_k(x)$ ;
5   NeighborhoodChange( $x, x', k$ )
6 until ending_condition met;

```

Figure 4: Variable Neighborhood Descent Algorithm

3.2.2.5 General Variable Neighborhood Search (GVNS)

General Variable Neighborhood Search (GVNS) can be defined as an improved variant of BVNS. It is possible to obtain hybrid variations when we use other metaheuristics in the *LocalSearch* phase in BVNS (Mjirda et al. (2016)). Besides, if we use one of the VND variations that we mentioned in Section 3.2.2.4 in the *LocalSearch* phase of BVNS (see Algorithm 2, line 5), we obtain GVNS. We present the basic structure of GVNS in Algorithm 3.2.2.5, we explain the details of GVNS in the following section 3.2.3 which is our proposed GVNS algorithm for mTSPD.

```

1 repeat
2    $k \leftarrow 1$ 
3   while  $k \leq k_{max}$  do
4      $x' \leftarrow \text{Shake}(x, k)$ 
5      $x'' \leftarrow \text{VND}(x')$ 
6     NeighborhoodChange( $x, x'', k$ )
7   end
8 until ending_condition met

```

Figure 5: General Variable Neighborhood Search Algorithm

3.2.3 Proposed General Variable Neighborhood Search Approach

In this section, we explain the GVNS algorithm that we develop in order to obtain robust and good results for mTSPD. When we examined the literature, we saw that GVNS algorithms developed for similar problems can achieve successful results. As an example, Soylu (2015) presented the GVNS algorithm for the mTSP problem, which consists Sequential-VND in the *LocalSearch* phase. Apart from that, de Freitas and Penna (2020) developed a Hybrid-GVNS algorithm for the FSTSP problem, which is more similar to our problem definition. They applied GVNS on the TSP results

obtained by using TSP-Solver in the HGVNS algorithm. In the *LocalSearch* phase, they use Randomized Variable Neighborhood Descent (RVND) algorithm.

The GVNS structure we build, unlike Kitjacharoenchai et al. (2019) and de Freitas and Penna (2020), performs the optimization of drone and truck routes together instead of obtaining the TSP result separately, thanks to its neighborhood structures. We describe these neighborhoods' structures in Section 3.2.3.5 and Section 3.2.3.7, and Figure 14 provides a visual representation of neighborhoods.

We present the pseudocode in Algorithm 6, which outlines the GVNS algorithm we develop. In initialization step (line 1), we start the GVNS algorithm by obtaining TSP or mTSP results by using Nearest Neighbor Algorithm according to the number of trucks (m), customer number (n) and coordinates of depot and customer nodes (*coordinates*) given in the problem set (See Sections 3.2.3.1-3.2.3.2). We determine the termination condition of our GVNS algorithm as to stop if no improvement is achieved after a specified number of iterations. Line 4 represents the *Shake* phase, line 5 represents the *LocalSearch* phase, and lines 6 to 10 represent the *NeighborhoodChange* phase as in the BVNS algorithm explained in Section 3.2.2.2. As we mentioned before, since we use *U-VND* in the *LocalSearch* phase, the algorithm we developed is called as GVNS. According to Mjirda et al. (2016)'s experiments, it was stated that *U-VND* takes more CPU time than the other VND variants mentioned in Section 3.2.2.4. As a result of our preliminary experiments and considering the definition of our problem, we found that neighborhoods working together as a whole rather than sequentially had a better effect on the outcome. If there is no improvement occurs at the end of the iteration, we move on to the next *Shake* neighborhood in the list. (See Fig 23 for flowchart of the proposed GVNS)

```

1  $x \leftarrow \text{Initialization}(m, n, \text{coordinates})$ 
2  $iter \leftarrow$  define the number of iterations for termination
3  $i \leftarrow 1$  repeat
4    $x' \leftarrow \text{Shake}(x, k)$ 
5    $x'' \leftarrow \text{U-VND}(x')$ 
6   if  $f(x'') < f(x)$  then
7      $x \leftarrow x''$ 
8   else
9      $k \leftarrow k + 1$ 
10  end
11   $i \leftarrow i + 1$ 
12 until  $i \leq iter$ 

```

Figure 6: Proposed General Variable Neighborhood Search Algorithm

3.2.3.1 Initialization

First of all, in the initialization phase, we need to create a robust solution vector for the GVNS algorithm to work effectively. In addition, it is a very critical issue that the operations to be performed by the neighborhoods can be easily applied on the solution vector. We develop two solution vectors that represent truck solution vector and drone solution vector separately. We create two vectors' structures as follows:

Truck Vector

As it is seen in Figure 7, first and only cell in the first section in the vector represents the truck number (m). In the next section, each number represents the number of nodes that trucks visit. In the last section of the vector, each color coded part in the third section represents the visiting sequence of each truck by the same order. The algorithm adds depot to starting and ending points (as seen in Figure 7) of each route while making calculations.



Figure 7: Truck Vector

Drone Vector

As in the truck vector structure, in the first section of the vector, the cell represents the number of drones used in delivery. The second section of the vector shows the number of nodes that are delivered by each drone. In the last section, each color coded part shows each drone by the same order. Each cell in each part represents departure node, visiting node and landing node, respectively (Figure 8).

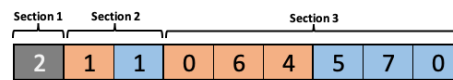


Figure 8: Drone Vector

3.2.3.2 Initial Solution with Nearest Neighbor Algorithm

In this section, we present the Nearest Neighbor Algorithm in Algorithm 9 that is used to create the initial solution. Before the GVNS algorithm starts to process,

we obtain a feasible (m)TSP solution without any drones using the Nearest Neighbor Algorithm according to the number of nodes (n) and trucks (m) in the problem. First of all, as stated in line 2, we create a separate travel time matrix for the truck (t^T) and the drone (t^D), using the number of nodes (n) and the coordinates of the nodes given in the problem set. In the next step, we create nested for loops that will assign each node to each truck. In this way, we assign nodes to each truck one by one. In line 6, we identify the *nearestNode* among the *remainingNodes* to the last node assigned to truck j . Then we assign this node to truck j and subtract it from the *remainingNodes* set. Finally, we update the tV according to the latest changes. (See Step 6, 7, 8, 9). After all nodes are allocated to the trucks, we create an empty dV in line 11 and we obtain our solution vectors and travel time matrices.

```

1  $tV, dV, t^D, t^T \leftarrow \text{NearestNeighbor}(m, n, \text{coordinates})$ 
2  $t^T, t^D \leftarrow \text{CreateMatrices}(n, \text{coordinates})$ 
3  $\text{remainingNodes} \leftarrow$  define set of  $n$  customers
4 for  $i = 0$  to  $n$  do
5   for  $j = 0$  to  $m$  do
6      $\text{nearestNode} \leftarrow$  determine the closest node to the previous node of
       truck  $j$  from  $\text{remainingNodes}$ 
7     assign  $\text{nearestNode}$  to truck  $j$ 
8     remove  $\text{nearestNode}$  from  $\text{remainingNodes}$ 
9      $tV \leftarrow$  update recent changes
10  end
11 end
12  $dV \leftarrow$  create blank vector
13 return  $tV, dV, tMatrix, dMatrix$ 

```

Figure 9: Nearest Neighbor Algorithm

3.2.3.3 Cost Evaluation

In this section, we present the algorithms where we calculate the objective function value of the solution using t^T, t^D and the solution vectors (tV, dV) that we constructed, with Algorithms 10 and 11. According to our objective function, we minimize the arrival time of the last vehicle (either drone or truck) that arrives at the depot. We present two different algorithms to calculate the arrival time at the depot. These are named as *calculate_arrival* and *cost_evaluation*. The *calculate_arrival* algorithm is a sub-algorithm that works recursively within the *cost_evaluation* algorithm and calculates the arrival time at each node.

When calculating the objective function, we classify each point under 3 categories:

- *landing* node: refers to the node where truck and drone meet.
- *drone* node: refers to the point where the drone delivers.
- other nodes: refers to the node where the drone does not deliver or land.

This classification facilitates the calculation of the objective function value. This can be explored in detail in the following sections.

```

1 arrival_time_node ← calculate_arrival(node, tD, tT, landing = False,
   drone = False)
2 if node == 0 then
3   return 0
4 else if node has arrival time then
5   return arrival_time_node
6 else
7   if landing == True then
8     pre_t = the truck node where the truck delivers before node
9     pre_d = the drone node where the drone delivers before lands to node
10    return max(calculate_arrival(pre_t) +
11              tTpre_t,node, calculate_arrival(pre_d, drone=True) +
12              tDpre_d,node), calculate_arrival(pre_t) + tTpre_t,node -
13              calculate_arrival(pre_d, drone=True) + tDpre_d,node
14   else if drone == True then
15     pre_t the drone and truck node where the drone departs before visits to
16     node
17     return calculate_arrival(pre_t) + tDpre_t,node
18   else
19     pre_t = the truck node where the truck delivers before node
20     return calculate_arrival(pre_t) + tTpre_t,node
21   end
22 end

```

Figure 10: Calculate Arrival Time

In the *cost_evaluation* algorithm, in line 2, we define an empty *arrival_array* for the arrival time of each node (See Algorithm 11). It has a structure that includes the arrival times of the truck and drone separately. Then, in line 3, we create the *landing* list of the nodes where the drone lands using *dV*, and in line 4, we create the *route_ending_time* list, which includes the arrival time of each route to the depot. Then, in line 5, we make the classifications that we mentioned above with the for loop that includes all the nodes in the *tV* and call the *calculate_arrival* function according to the determined conditions. We classify each node from line 5 to 15. If *tV* or *dV* is infeasible, we take the objective function value as infinite and terminate the *cost_evaluation* algorithm (See line 8). Then, in line 17, the arrival time of each route to the warehouse is calculated and recorded in the *route_ending_time* list. In line 18,

if any drone lands at the warehouse, its arrival time at the warehouse is added to the *route_ending_time* list. Finally, in *route_ending_time* list, we return the maximum value as the value of the objective function.

We calculate the *arrival_time* of each node using the following formulas in the *calculate_arrival* function:

Let Tl_i is arrival time of truck at node i and Dl_i is arrival time of drone at node i .

- *landing node arrival_time*: $\max(Tl_{i-1} + t_{i-1,i}^T, Dl_{i-1} + t_{i-1,i}^D)$
- *drone node arrival_time*: $Tl_{i-1} + t_{i-1,i}^D$
- *other nodes arrival_time*: $Tl_{i-1} + t_{i-1,i}^T$

We show the above formulas in lines 10, 13, 16 which are integrated into Algorithm 10 and they work recursively.

```

1 arrival_array, objective, route_ending_time ← cost_evaluation( $t^D, t^T,$ 
   $tV, dV$ )
2 arrival_array ← create empty array for arrival times
3 landing_nodes ← identify drone landing nodes from  $dV$ 
4 route_ending_time ← create empty list for each route
5 foreach node in  $tV$  do
6   if node  $\neq 0$  and node in landing_nodes then
7     if  $tV$  or  $dV$  is infeasible then
8       | set objective =  $\infty$  and exit
9     end
10    pre_d = the drone node where the drone delivers before lands to node
11    arrival_array1,pre_d ← calculate_arrival(pre_d, drone = True)
12    arrival_array1,node, arrival_array2,node ←
      calculate_arrival(node, landing = True)
13  else if node  $\neq 0$  then
14    | arrival_array1,node ← calculate_arrival(node)
15  end
16  foreach last_node of  $tV$  do
17    | route_ending_timeroute ← arrival_arraylast_node +  $t_{last\_node,0}^T$ 
18  end
19  if any drone lands into depot then
20    | route_ending_timedrone ← arrival_arraydrone +  $t_{drone,0}^D$ 
21  end
22  objective = max(route_ending_time)
23  return objective, arrival_array, route_ending_time

```

Figure 11: Cost Evaluation

3.2.3.4 Shake Phase

We use four different neighborhoods named as *random_swap*, *random_oneMove*, *random_visitorSwap* and *random_droneRemove* in the shaking phase. The main structure of these neighborhoods is designed to move to different solutions to avoid local optimum traps by operating randomly while making changes.

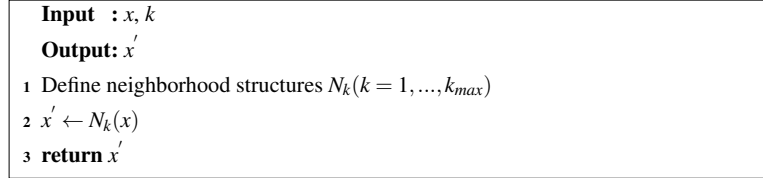


Figure 12: Shake Phase

3.2.3.5 Shake Neighborhoods

In this section, we explain the neighborhoods we use in the *Shake* phase, which can also be called as perturbation phase. The neighborhoods that we present aims to slightly worsen the result by performing a stochastic search to escape the local optimum, hoping for the result to get better. We present neighborhoods as follows.

1. **random_swap Neighborhood**: This neighborhood swaps positions of two randomly selected nodes visited by trucks.
2. **random_oneMove Neighborhood**: This neighborhood moves one randomly selected node from a randomly selected truck to another randomly selected truck.
3. **random_visitorSwap Neighborhood**: This neighborhood swaps the vehicle type (drone or truck) of delivery point randomly.
4. **random_droneRemove Neighborhood**: This neighborhood removes a random node that is visited by drone and assigns it to a random truck.

3.2.3.6 Local Search / U-VND

In Union Variable Neighborhood Descent (U-VND), we develop neighborhoods that are different from the *Shake* phase. The main purpose of this approach is to boost the probability of improvement of our objective. Regarding our objective, our main direction is to minimize the longest route's completion time. In the U-VND phase, our neighborhoods' main focus is routes that have the worst objective function value in each iteration to directly affect the objective function value. We develop nine

neighborhoods which are named as *swapMove*, *visitorSwap*, *swap*, *oneMove*, *2-Opt*, *departure*, *landing*, *droneAdd* and *random_droneAdd*.

```

Input :  $x'$ 
Output:  $x''$ 
1 Define set of neighborhoods  $N_l$  ( $l = 1, \dots, l_{max}$ )
2  $l \leftarrow 1$ 
3 while  $l < l_{max}$  do
4    $x' \leftarrow N_l(x')$ 
5    $l \leftarrow l + 1$ 
6 end
7  $x'' \leftarrow x'$ 
8 return  $x''$ 

```

Figure 13: Union Variable Neighborhood Descent

3.2.3.7 Local Search / U-VND Neighborhoods

In this section, we explain the neighborhoods that we use in the U-VND phase in detail and present their pseudo codes. Apart from that, we have created Fig 14 to give a better explanation of the effect of the neighborhoods on solutions. We present neighborhoods as follows.

swapMove Neighborhood

This move basically combines *swap* and *oneMove* neighborhoods. If the number of trucks in the problem set is less than 2, the neighborhood stops running because there is no other route to move or swap (See Algorithm 15, line 2). First, the worst route is defined as *maxCostRoute* list. If there are less than 2 nodes assigned to *maxCostRoute*, neighborhood stops running because if it moves a node to another route, there will be no node assigned to the *maxCostRoute* and this creates an infeasible solution in our problem (lines 5-8). After this condition, the list of nodes that truck visits (except *maxCostRoute*) is defined as *truckNodes* using the *tV* (line 8). *swapMove* neighborhood swaps a node from worst truck route with a node from another route. Next, it takes the adjacent node to swapped node and moves to the same route that swap has been made. If any improvement is achieved, it terminates the process; otherwise, it continues and tries other combinations and returns the best possible combination among all.

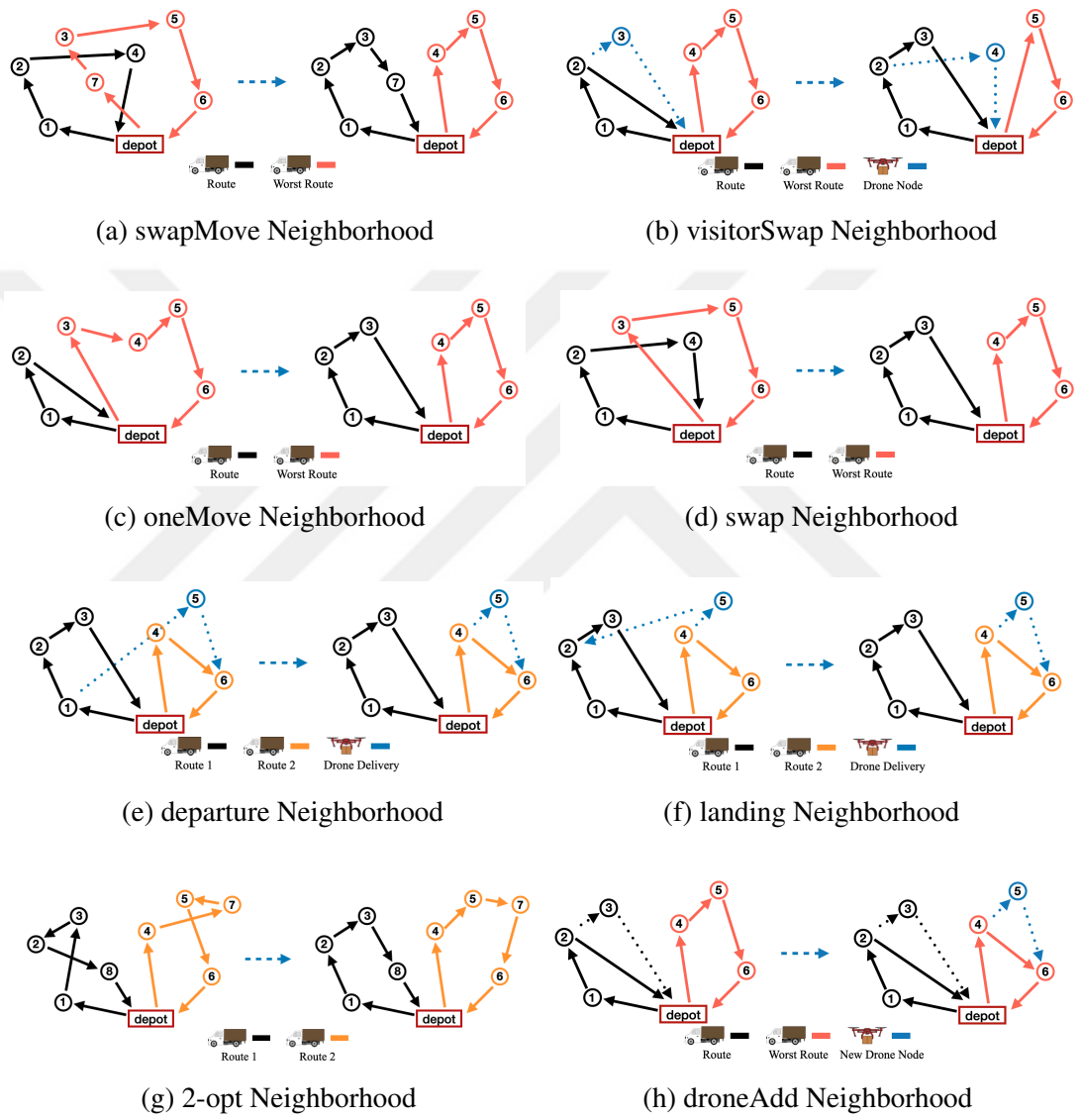


Figure 14: U-VND Neighborhoods

```

1  $tV' \leftarrow \text{swapMove}(\text{truckNumber}, tV)$ 
2 if  $\text{truckNumber} < 2$  then
3   | exit
4 end
5  $\text{maxCostRoute} \leftarrow$  specify max cost route
6 if # of nodes in  $\text{maxCostRoute} < 2$  then
7   | exit
8 end
9  $\text{truckNodes} \leftarrow$  specify truck visiting nodes from  $tV$ 
10 set  $\text{improved} = \text{False}$ 
11 for  $i = 0$  to  $\text{len}(\text{maxCostRoute}) - 1$  do
12   for  $j = 0$  to  $\text{len}(\text{truckNodes})$  do
13     if  $\text{improved} == \text{True}$  then
14       | break
15     end
16     if  $\text{truckNodes}_j$  in  $\text{maxCostRoute}$  then
17       | pass
18     else
19        $\text{first\_node} \leftarrow \text{maxCostRoute}_i$ 
20        $\text{second\_node} \leftarrow \text{maxCostRoute}_{i+1}$ 
21        $\text{swap\_node} \leftarrow \text{truckNodes}_j$ 
22       insert  $\text{second\_node}$  after  $\text{swap\_node}$ 
23       swap  $\text{first\_node}$  with  $\text{swap\_node}$ 
24        $tV' \leftarrow$  update recent changes
25        $\text{new\_objective} \leftarrow \text{cost\_evaluation}(t^D, t^T, tV', dV)$ 
26       if  $\text{new\_objective} < \text{objective}$  then
27         | set  $\text{improved} = \text{True}$ 
28       end
29       set  $tV' \leftarrow tV$ 
30     end
31   end
32 end
33 return Best  $tV'$ 

```

Figure 15: swapMove Neighborhood

visitorSwap Neighborhood

This neighborhood swaps the types of vehicles that visit a node Figure 14b. We present the pseudocode of neighborhood in Algorithm 16. If the drone does not exist in the incumbent solution vectors, neighborhood terminates without executing a move (line 2). First the neighborhood identifies the nodes where the drone delivers as droneNodes in line 5, then in line 6, it identifies the worst route as maxCostRoute , without including the nodes that already exist in droneNodes . In lines 8-20, two nested for loops with all options are created between the nodes that can be swapped, where the truck delivers in maxCostRoute and the nodes where the drone delivers in the droneNodes . It swaps truck_node with drone_node and update tV' , dV' according to the most recent swap. Then it checks in line 16 if there is an improvement. If any

improvement is achieved, it terminates the process; otherwise, it continues and tries other combinations.

```

1  $tV', dV' \leftarrow \text{visitorSwap}(tV, dV)$ 
2 if  $dV$  is empty then
3   | exit
4 end
5  $\text{droneNodes} \leftarrow$  identify drone visiting nodes
6  $\text{maxCostRoute} \leftarrow$  specify max cost route
7 set  $\text{improved} = \text{False}$ 
8 for  $\text{truck\_node}$  in  $\text{maxCostRoute}$  do
9   for  $\text{drone\_node}$  in  $\text{droneNodes}$  do
10    if  $\text{improved} == \text{True}$  then
11      | break
12    else
13      swap  $\text{truck\_node}$  with  $\text{drone\_node}$ 
14       $tV', dV' \leftarrow$  update recent changes
15       $\text{new\_objective} \leftarrow \text{cost\_evaluation}(t^D, t^T, tV', dV')$ 
16      if  $\text{new\_objective} < \text{objective}$  then
17        | set  $\text{improved} = \text{True}$ 
18      end
19      set  $tV' \leftarrow tV$ 
20      set  $dV' \leftarrow dV$ 
21    end
22  end
23 end
24 return  $\text{Best } tV', \text{Best } dV'$ 

```

Figure 16: visitorSwap Neighborhood

swap Neighborhood

This neighborhood differs from the *random_swap* in the *Shake* phase in terms of the structure of the neighborhood. In *swap*, a discrete search is performed as opposed to a stochastic search in *random_swap*. See the Figure 14d to better understand the moves that neighborhood executes. In Algorithm 17 we present the pseudocode of neighborhood. If there are less than 2 trucks in the problem, the neighborhood terminates the process without changing solution vectors. First, in *swap*, the worst route is defined as *maxCostRoute*, then the nodes that are delivered by a truck are defined as *remainNodes*, but the nodes in *maxCostRoute* are not included in the *remainNodes* (lines 5-6). Then in lines 8-20 a nested for loop is created, it contains the nodes in *maxCostRoute* and *remainNode*. Thus, *swap* neighborhood includes all nodes that can be swapped in the iteration. In lines 13-14, it swaps *first_node* from *maxCostRoute* with *second_node* from *remainNodes*, and updates tV' according to this change. In the next step it checks whether the objective function value improves or not. Neighborhood terminates the process as soon as the improvement is achieved,

otherwise it continues to try other possible swaps.

```

1  $tV' \leftarrow \text{swap}(\text{truckNumber}, tV)$ 
2 if  $\text{truckNumber} < 2$  then
3   | exit
4 end
5  $\text{maxCostRoute} \leftarrow$  specify max cost route
6  $\text{remainNodes} \leftarrow$  identify all truck visiting nodes except  $\text{maxCostRoute}$ 
7 set  $\text{improved} = \text{False}$ 
8 for  $\text{first\_node}$  in  $\text{maxCostRoute}$  do
9   for  $\text{second\_node}$  in  $\text{remainNodes}$  do
10    | if  $\text{improved} == \text{True}$  then
11      | break
12    | else
13      | swap  $\text{first\_node}$  with  $\text{second\_node}$ 
14      |  $tV' \leftarrow$  update recent changes
15      |  $\text{new\_objective} \leftarrow \text{cost\_evaluation}(t^D, t^T, tV', dV)$ 
16      | if  $\text{new\_objective} < \text{objective}$  then
17        | set  $\text{improved} = \text{True}$ 
18      | end
19      | set  $tV' \leftarrow tV$ 
20    | end
21  | end
22 end
23 return  $\text{Best } tV'$ 

```

Figure 17: swap Neighborhood

oneMove Neighborhood

This neighborhood varies from the *random_oneMove* in the *Shake* stage because of the construction of the neighborhood. See the Figure 14c to better understand the moves that neighborhood executes. We present the pseudocode of *oneMove* neighborhood in Algorithm 18. Firstly, *oneMove* defines the worst route in line 5 as *maxCostRoute*, and in the next step, it checks the number of nodes that *maxCostRoute* has. If the number of nodes assigned to *maxCostRoute* is less than 2, neighborhood terminates without making any changes, because if it moves the only existing node to another route, it will get an infeasible result. Between lines 11-28, nested for loops are created containing the nodes of *maxCostRoute* and the nodes of *truckNodes*. In this way, *oneMove* includes all positions that nodes in *maxCostRoute* can move through. In line 16, if a node in *truckNodes* exists in *maxCostTruck*, neighborhood does not perform an operation and thus saves run time. In the following steps, *first_node* from *maxCostRoute* is placed after *second_node* from *truckNodes* and *tV'* is updated according to this change. In the next step it checks whether the objective function value improves or not. If it achieves any improvement, stops the process and executes that move. Otherwise, it continues to try other possible moves until all possible moves

have been tried, then returns the best possible move among them.

```

1  $tV' \leftarrow \text{oneMove}(\text{truckNumber}, tV)$ 
2 if  $\text{truckNumber} < 2$  then
3   | exit
4 end
5  $\text{maxCostRoute} \leftarrow \text{specify max cost route}$ 
6 if # of nodes in  $\text{maxCostRoute} < 2$  then
7   | exit
8 end
9  $\text{truckNodes} \leftarrow \text{specify truck visiting nodes from } tV$ 
10 set  $\text{improved} = \text{False}$ 
11 for  $i = 0$  to  $\text{len}(\text{maxCostRoute})$  do
12   | for  $j = 0$  to  $\text{len}(\text{truckNodes})$  do
13     | if  $\text{improved} == \text{True}$  then
14       | break
15     end
16     if  $\text{truckNodes}_j$  in  $\text{maxCostRoute}$  then
17       | pass
18     else
19       |  $\text{first\_node} \leftarrow \text{maxCostRoute}_i$ 
20       |  $\text{second\_node} \leftarrow \text{truckNodes}_j$ 
21       | insert  $\text{first\_node}$  after  $\text{second\_node}$ 
22       |  $tV' \leftarrow \text{update recent changes}$ 
23       |  $\text{new\_objective} \leftarrow \text{cost\_evaluation}(t^D, t^T, tV', dV)$ 
24       | if  $\text{new\_objective} < \text{objective}$  then
25         | set  $\text{improved} = \text{True}$ 
26       end
27       | set  $tV' \leftarrow tV$ 
28     end
29   end
30 end
31 return  $\text{Best } tV'$ 

```

Figure 18: oneMove Neighborhood

2-opt Neighborhood

The 2-opt algorithm was first presented by Croes (1958) in order to solve the TSP problem. It has been rearranged to affect each truck route for *U-VND* stage. Visualization of the 2-opt neighborhood can be observed in Figure 14g.

departure Neighborhood

The *departure* neighborhood changes the departure node of a drone. Figure 14e shows the operations applied in this neighborhood. In Algorithm 19, we present the pseudocode of the neighborhood. If the dV solution vector is empty, or in other words if there is no drone in the solution, the neighborhood terminates without any action (line 2). First of all in line 5, the neighborhood determines the nodes where the drone has no interaction as *available_dep*, which are the points where the trucks are making delivery and no drone lands or departs. Next, it defines the drones' delivery nodes as

droneNodes list. We create nested for loops in lines 8-20 to find the best departure point from all *available_deps*. In line 13, it replaces the *i*th node in *available_dep* with the departure node of the drone and updates the dV' according to this replacement. After this modification, the new objective function value is calculated in line 23 and if an improvement occurs, it stops running and returns the newly formed dV' . If no improvement occurs, it continues searching and returns the best dV' after trying all possible replacements.

```

1  $dV' \leftarrow \text{departure}(dV)$ 
2 if  $dV$  is empty then
3   | exit
4 end
5  $available\_dep \leftarrow$  identify truck visiting nodes and depot where drones are not
   departure or landing
6  $droneNodes \leftarrow$  specify drone visiting nodes
7 set  $improved = False$ 
8 for  $i = 0$  to  $droneNodes$  do
9   for  $j = 0$  to  $available\_dep$  do
10    if  $improved == True$  then
11      | break
12    end
13    make  $available\_dep_j$  departure node of  $droneNodes_i$ 
14     $dV' \leftarrow$  update recent changes
15     $new\_objective \leftarrow \text{cost\_evaluation}(r^D, r^T, rV, dV')$ 
16    if  $new\_objective < objective$  then
17      | set  $improved = True$ 
18    end
19     $dV' \leftarrow dV$ 
20  end
21 end
22 return Best  $dV'$ 

```

Figure 19: departure Neighborhood

landing Neighborhood

The *landing* neighborhood works exactly the same as the *departure* neighborhood. The only difference is, it tries to improve the landing point, not the departure point. This can be seen in line 13 of the *landing* neighborhood Algorithm 20. The moves that *landing* neighborhood executes can be observed in Figure 14f.

```

1  $dV' \leftarrow \text{landing}(dV)$ 
2 if  $dV$  is empty then
3   | exit
4 end
5  $available\_land \leftarrow$  identify truck visiting nodes and depot where drones are
   not departure or landing
6  $droneNodes \leftarrow$  specify drone visiting nodes
7 set  $improved = False$ 
8 for  $i = 0$  to  $droneNodes$  do
9   for  $j = 0$  to  $available\_land$  do
10    | if  $improved == True$  then
11      | break
12    | end
13    | make  $available\_dep_j$  landing node of  $droneNodes_i$ 
14    |  $dV' \leftarrow$  update recent changes
15    |  $new\_objective \leftarrow cost\_evaluation(t^D, t^T, tV, dV')$ 
16    | if  $new\_objective < objective$  then
17      | set  $improved = True$ 
18    | end
19    |  $dV' \leftarrow dV$ 
20   | end
21 end
22 return Best  $dV'$ 

```

Figure 20: landing Neighborhood

droneAdd & random_droneAdd Neighborhoods

We build the *droneAdd* neighborhood in two different structures. Besides *droneAdd*, we also develop the *random_droneAdd* neighborhood. The reason is, *droneAdd* takes a lot of time to run in a solution with no drones or solutions with a small number of drones. This problem occurs because there are too many available nodes to choose when adding drones into the solution. Even if *random_droneAdd* adds a poorly assigned drone to the result, the operations of the *departure*, *landing*, and *visitorSwap* neighborhoods allow these poorly added drones to have a better impact on the result. As the number of nodes that the new drones can use decreases, *droneAdd* comes into play, saving time and more effectively assigns drones to the nodes. We present the visualisation of *droneAdd* in Figure 14h and the pseudocode in Algorithm 21. *droneAdd* and *random_droneAdd* do not work together in the same iteration. We can find the number of drones that can be added in a given instance using the following formula:

$$\left\lfloor \frac{n+2}{3} \right\rfloor \quad (3.37)$$

The +2 in this formula represents the start and end depots in the problem definition. 3 represents the number of nodes required (departure, visit, land) to add the drone to the result. If the current number of drones is half or less than the number we get as

a result of this formulation, we use *random_droneAdd*, otherwise *droneAdd* works in iteration. In line 8, if the number of nodes required to add the drone is not available, the neighborhood terminates without any changes. In lines 11-21, we create nested for loops to assign drones to the available nodes. tV' and dV' are updated according to the changes in line 14 and the new objective function value is calculated. Neighborhood returns the best dV' and tV' after trying all possible options.

```

1  $tV', dV' \leftarrow \text{droneAdd}(\text{droneNumber}, tV, dV)$ 
2 if  $\text{droneNumber} \leq \lfloor \frac{n+2}{3} \rfloor / 2$  then
3   | exit
4 end
5  $\text{maxCostRoute} \leftarrow$  specify max cost route's nodes that no drone interactions
6  $\text{available\_dep} \leftarrow$  identify truck visiting nodes and depot where drones are not
  departure or landing
7  $\text{available\_land} \leftarrow$  identify truck visiting nodes and depot where drones are
  not departure or landing
8 if  $\text{len}(\text{maxCostRoute}) \leq 1$  or
    $\text{len}(\text{available\_dep}) + \text{len}(\text{available\_land}) \leq 3$  then
9   | exit
10 end
11 for  $i = 0$  to  $\text{maxCostRoute}$  do
12   for  $j = 0$  to  $\text{available\_dep}$  do
13     for  $k = 0$  to  $\text{available\_land}$  do
14       remove  $\text{maxCostRoute}_i, \text{available\_dep}_j, \text{available\_land}_k$  from
          $\text{maxCostRoute}$ 
15        $tV', dV' \leftarrow$  update recent changes
16        $\text{new\_objective} \leftarrow \text{cost\_evaluation}(t^D, t^T, tV', dV')$ 
17       set  $tV' \leftarrow tV$ 
18       set  $dV' \leftarrow dV$ 
19     end
20   end
21 end
22 return  $\text{Best } tV', \text{Best } dV'$ 

```

Figure 21: droneAdd Neighborhood

3.3 Lower Bound

In this section, we propose the lower bound we created for the mTSPD problem. We use 3 cases when calculating the lower bound, and separate these cases according to the possible drone movements (See Fig 22). We present the cases as follows.

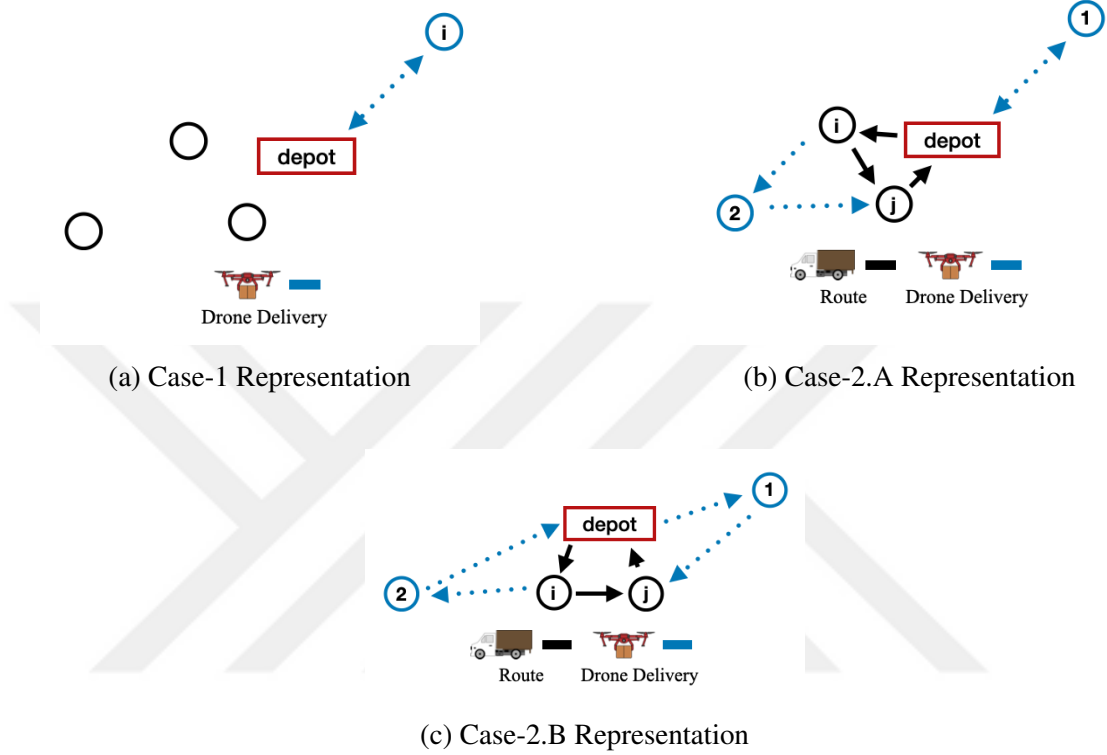


Figure 22: Visualization of Cases Used in Lower Bound Calculation

Let t_{ij}^D drones's travel time between i and j , and let t_{ij}^T is truck travel time between i and j , and d corresponds to the depot.

3.3.1 Case-1

Case-1 represents the movement where the drone departs from the depot and lands back in the depot. Let's say node i is the furthest node from depot in the problem set. We calculate the Case-1 value with following formula:

- $LB^1 = t_{di}^D * 2$ or it can be shown as $LB^1 = t_{di}^D + t_{id}^D$

3.3.2 Case-2.A

Case-2.A represents the situations where first drone departs from depot and lands on the depot, and the seconds drone departs from the truck and lands on the truck again

(See Fig 22b).

First, let's say 1 is farthest node from the depot, and 2 is the second farthest node from the depot.

$$\bullet LB_{ij}^{A12} = \max\{t_{d1}^D * 2; t_{di}^T + \max(t_{ij}^T; t_{i2}^D + t_{2j}^D) + t_{jd}^T\} \quad \forall i, j$$

Last, let's say 2 is farthest node from the depot, and 1 is the second farthest node from the depot.

$$\bullet LB_{ij}^{A21} = \max\{t_{d2}^D * 2; t_{di}^T + \max(t_{ij}^T; t_{i1}^D + t_{1j}^D) + t_{jd}^T\} \quad \forall i, j$$

According to the formulations we mentioned above, we define LB^A as follows:

$$\bullet LB^A = \min\{LB_{ij}^{A12}; LB_{ij}^{A21}\} \quad \forall i, j$$

3.3.3 Case-2.B

Case 2.B represents situations where the first drone departs from the depot and lands on the truck, and the second drone departs from truck and lands on the depot.

First, let's say 1 is farthest node from the depot, and 2 is the second farthest node.

$$\bullet LB_{ij}^{B1} = \max\{\max(t_{di}^T + t_{ij}^T; t_{d1}^D + t_{1j}^D) + t_{jd}^T; t_{di}^D + t_{i2}^D + t_{2d}^D\} \quad \forall i, j$$

Last, let's say 2 is farthest node from the depot, and 1 is the second farthest node.

$$\bullet LB_{ij}^{B2} = \max\{\max(t_{di}^T + t_{ij}^T; t_{d2}^D + t_{2j}^D) + t_{jd}^T; t_{di}^D + t_{i1}^D + t_{1d}^D\} \quad \forall i, j$$

According to the formulations we mentioned above, we define LB^B as follows:

$$\bullet LB^B = \min\{LB_{ij}^{B1}; LB_{ij}^{B2}\} \quad \forall i, j$$

3.3.4 LB Calculation

After calculating the LB of the cases we mentioned, now we calculate the LB of the problem set with the formulation below:

$$\bullet LB = \max\{LB^1; \min(LB^A; LB^B)\}$$

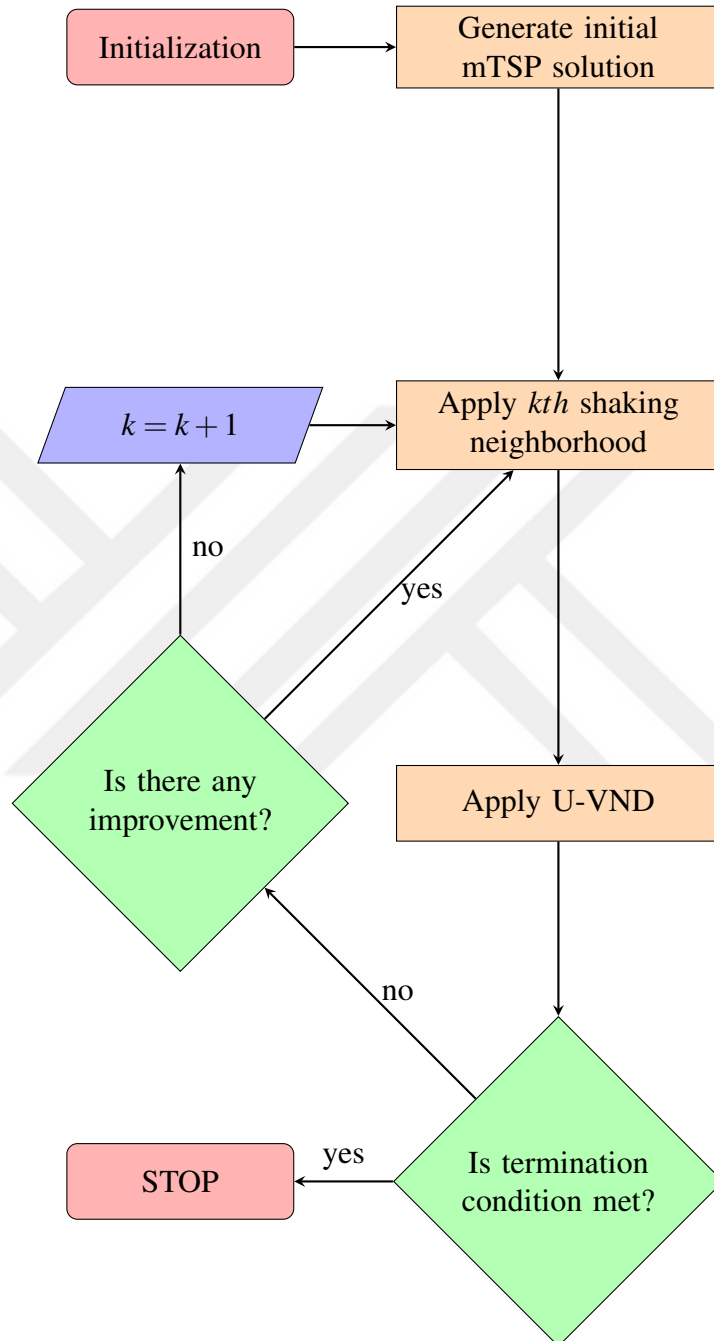


Figure 23: Flowchart of the Proposed General Variable Neighborhood Search Algorithm

CHAPTER 4: EXPERIMENTAL RESULTS

We carried out computational experiments to evaluate the performance of the proposed General Variable Neighborhood Search (GVNS) algorithm. We developed the algorithm by using Python programming language and run on 2,7 GHz Dual-Core Intel Core i5 processor with 8 GB 1867 MHz DDR3 Ram. We used the test data set from Kitjacharoenchai et al. (2019) to compare our GVNS with their proposed algorithms. In the test data set, there are small, medium and large sized problems with 8, 25 and 50 nodes, respectively. Apart from the number of nodes, each problem type divided into different categories such as the location of the depot and the distribution of the nodes (See section 4.1). We ran each problem set with the number of trucks varying from 1 to 5 and we ran each instance 20 times as in Kitjacharoenchai et al. (2019). We applied different neighborhoods and neighborhood orders in *Shake* and *U-VND* during the experiments. Based on the preliminary experiments and observations that we have done, we decided to use the order of the neighborhoods which are given in Table 2. We ran medium sized problem sets on the termination condition that 300 iterations will end if there is no improvement. For large problem sets we decreased the termination condition to 200 iterations, for the purpose of decreasing the run time of the algorithm.

Table 2: Order of Neighborhoods Used in GVNS

Phase	Neighborhoods in order
Shake	random_swap, random_oneMove, random_visitorSwap, random_droneRemove
U-VND	swapMove, visitorSwap, swap, oneMove, twoOpt, dep, land, dep, land, random_droneAdd, droneAdd, dep, land, dep, land

4.1 Datasets

We used the datasets provided by Kitjacharoenchai et al. (2019) during the testing process of the GVNS algorithm that we developed. As we mentioned before, these datasets consist of 5 different types. These are named according to their structure from Type 1 to 5. When mentioning about these datasets in results, we abbreviate T1, T2, T3, T4, and T5 to represent Type 1-5, respectively. They generated these instances in different types in a 1000×1000 *unit*² area. The first type of instance T1, consists of the central warehouse located at (500, 500) and customers uniformly distributed around it. The second type of instance T2, includes a depot located at (500, 0) which is the bottom of the area and customers are uniformly distributed around it, just like in T1. The third type of problem T3, consists of the central depot (500, 500) as in

T1, and the customers distributed in a circle at a radius of 500. The fourth problem type T4, involves a centrally located depot and circularly distributed customers. Unlike T3, customers take the form of a ring because they located the customers far from the depot. The last problem type T5, includes a central depot and customers located in 5 different regions. For a better understanding, we plotted the instances (50 Nodes) that we used in our experiments in Figure 24.

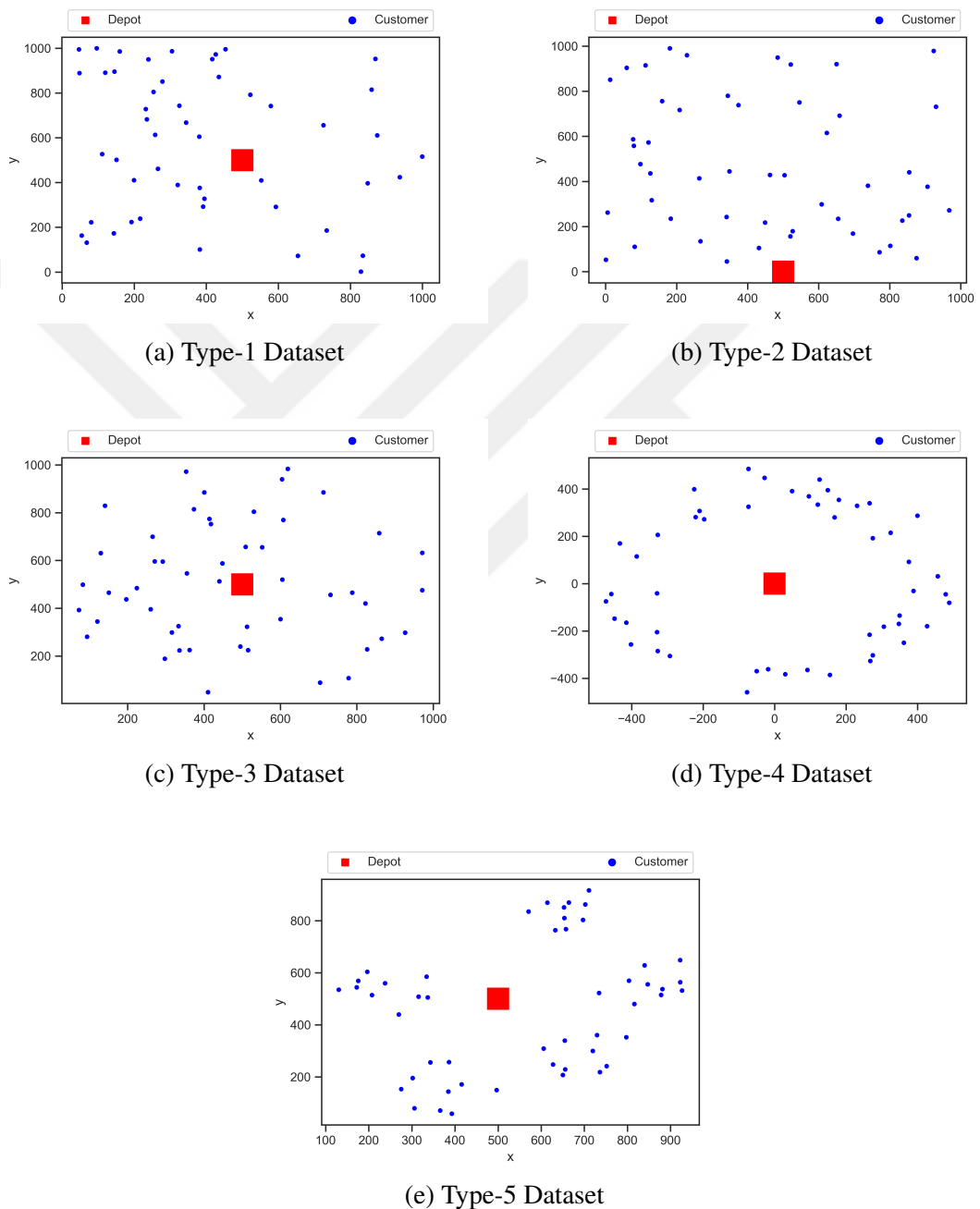


Figure 24: Dataset Visualisation

4.2 Results

In Tables 12 and 13, we present computational results of GVNS and compare with GA-ADI and K-means ADI proposed by Kitjacharoenchai et al. (2019). Under the name of each algorithm, we list best found solution, average of objective function values after 20 runs and run time in seconds for each instance. Kitjacharoenchai et al. (2019) proposed another algorithm called Random ADI, however we discard it from comparison since this algorithm does not perform as good as the other ones.

We compare the results based on three criteria, which are best found solution, average objective value after 20 runs and run time in Tables 14 and 15. We present the best values of each criterion for all algorithms and also we show the deviation from best value for each algorithms separately in Table 14 and Table 15. The deviations from best values are calculated with following formula:

- $GAP = algorithmValue / bestValue - 1$.

In order to examine the results we obtained, we determined 3 main categories. These are the number of trucks, the instance type, and finally the number of nodes. We evaluated these categories according to the 3 criteria as we mentioned before. These evaluations can be examined in the following sections.

4.2.1 Examination of the Results Based on the Number of Trucks

In this section, we examine how the results we obtained vary according to the number of trucks in the problem. There are 10 instances of each vehicle number, with 5 each in 25 and 50 node instances. According to the criteria that we mentioned before, we present Table 3, to show the number of times each algorithm finds the best solution after 20 runs. Apart from that, we also present Table 4 to show how much the solutions found by each algorithm deviate from the best results as a percentage.

Table 3: Number of Best Solutions According to the Number of Trucks

Algorithm	Minimum Objective			Average Objective			Run Time		
	GA-ADI	K-means ADI	GVNS	GA-ADI	K-means ADI	GVNS	GA-ADI	K-means ADI	GVNS
Truck									
1	0	0	10	0	0	10	0	2	8
2	5	2	4	5	2	3	3	2	5
3	0	5	5	2	4	4	1	4	5
4	1	4	5	2	4	4	2	3	5
5	2	1	8	2	4	4	1	4	5
Total	8	12	32	11	14	25	7	15	28

Table 4: Average Gaps According to the Number of Trucks

Algorithm	Best Found Gap			Average Best Gap			Average Run Time Best Gap		
	GA-ADI	K-means ADI	GVNS	GA-ADI	K-means ADI	GVNS	GA-ADI	K-means ADI	GVNS
Truck									
1	5.54%	10.97%	0.00%	4.52%	8.30%	0.00%	141.69%	89.31%	2.58%
2	2.13%	15.49%	2.25%	0.54%	12.65%	4.69%	62.88%	38.50%	27.94%
3	5.80%	3.67%	2.00%	2.18%	3.57%	4.57%	62.13%	34.17%	18.36%
4	4.49%	5.33%	1.36%	2.82%	4.06%	4.26%	56.99%	36.37%	21.83%
5	5.01%	11.92%	0.86%	2.16%	7.04%	3.16%	66.61%	44.12%	12.89%
Average	4.59%	9.47%	1.29%	2.45%	7.12%	3.34%	78.06%	48.49%	16.72%

When we examine the tables, we can clearly see that in instances where there is only one truck, the GVNS finds better results than the GA-ADI and K-means ADI algorithms in all instances, both in the average of 20 runs and in the best results. Among them, GVNS falls behind K-means ADI in run time in only 2 instances. In Table 4, when we look at the deviations from the best results, we can say that GVNS is better than the others except the average of the instances with 2 trucks. On the other hand, when we look at the deviations from the averages of 20 runs, we can say that GVNS falls behind GA-ADI except for single truck instances.

4.2.2 Examination of the Results Based on the Instance Types

In this section, we examine how the results obtained with the GVNS algorithm vary according to the problem types. Each instance includes 5 different sub-problems according to the number of trucks from 1 to 5. When we include both 25 and 50 nodes, we can state that each instance consists of 10 sub-problems according to the problem types. We present Table 5 that shows the number of times each algorithm finds the best solution according to the instances in given criteria. Table 6 shows the deviation of the solutions found by each algorithm from the best results as a percentage.

Table 5: Number of Best Solutions According to Instance Type

Algorithm	Minimum Objective			Average Objective			Run Time		
	GA-ADI	K-means ADI	GVNS	GA-ADI	K-means ADI	GVNS	GA-ADI	K-means ADI	GVNS
Instance									
T1	2	3	5	2	3	5	1	4	5
T2	1	0	9	1	0	9	0	4	6
T3	0	2	8	1	3	6	4	0	6
T4	1	3	6	0	7	3	0	4	6
T5	4	4	4	7	1	2	2	3	5
Total	8	12	32	11	14	25	7	15	28

Table 6: Average Gaps According to Instance Type

Algorithm	Best Found Gap			Average Best Gap			Average Run Time Best Gap		
	GA-ADI	K-means ADI	GVNS	GA-ADI	K-means ADI	GVNS	GA-ADI	K-means ADI	GVNS
Instance									
T1	5.71%	5.57%	1.15%	2.39%	3.78%	1.08%	87.97%	54.19%	22.70%
T2	5.75%	13.04%	1.09%	3.09%	9.47%	0.77%	86.16%	52.06%	16.25%
T3	7.63%	7.60%	0.50%	3.47%	4.38%	1.76%	71.45%	58.87%	14.21%
T4	2.49%	3.44%	1.24%	2.29%	1.25%	5.89%	68.10%	35.80%	21.21%
T5	1.39%	17.72%	2.50%	0.99%	16.72%	7.19%	76.63%	41.54%	9.23%
Average	4.59%	9.47%	1.29%	2.45%	7.12%	3.34%	78.06%	48.49%	16.72%

When we examine Table 5, the first noticeable difference emerges in the T2 problem type. In the T2 instances, both the best results and the average of 20 runs, the GVNS algorithm reached the best value in 9 instances. We can state that it falls behind GA-ADI in only 1 instance. On the other hand, if we look at Table 6, we can say that GVNS algorithm in T4 instances is worse than K-means ADI and GA-ADI, also in T5 instances it is worse than GA-ADI on the average of 20 runs. However, in the T1, T2 and T3 instances, it outperformed other algorithms in the average of 20 runs, and shows a better performance than other algorithms in terms of best results among all problem types except T5.

4.2.3 Examination of the Results Based on the Number of Nodes

In this section, we group the instances according to the number of nodes and present the results as a kind of summary. As in the previous sections, in Table 7, we present the number of best results each algorithm achieves according to the criteria we specified, and in Table 8 we report the deviations of the solutions found by each algorithm from the best result as a percentage.

Table 7: Number of Best Solutions According to the Number of Nodes

Algorithm	Minimum Objective			Average Objective			Run Time		
	GA-ADI	K-means ADI	GVNS	GA-ADI	K-means ADI	GVNS	GA-ADI	K-means ADI	GVNS
Node									
25	5	8	14	6	9	10	0	0	25
50	3	4	18	5	5	15	7	15	3
Total	8	12	32	11	14	25	7	15	28

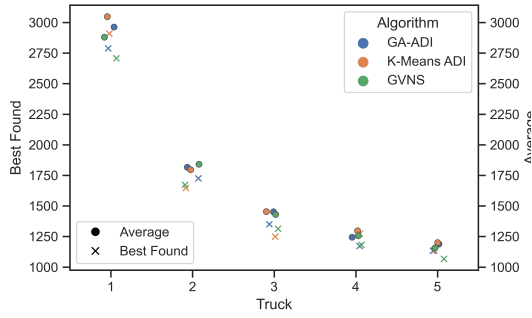
Table 8: Average Gaps According to the Number of Nodes

Algorithm	Best Found Gap			Average Best Gap			Average Run Time Best Gap		
	GA-ADI	K-means ADI	GVNS	GA-ADI	K-means ADI	GVNS	GA-ADI	K-means ADI	GVNS
Node									
25	3.44%	6.66%	1.51%	2.07%	4.90%	4.05%	150.76%	92.92%	0.00%
50	5.75%	12.29%	1.08%	2.82%	9.35%	2.62%	5.36%	4.06%	33.44%
Average	4.59%	9.47%	1.29%	2.45%	7.12%	3.34%	78.06%	48.49%	16.72%

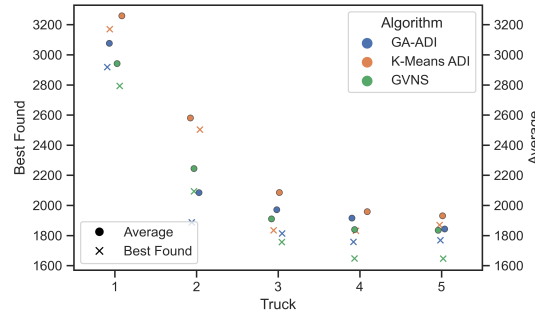
When we look at Table 7, in the instances with 25 nodes, we observe that GVNS finds the best result 14 times among 25 instances. In the average of 20 runs, we can say that GVNS surpassed other algorithms by catching the best average 10 times. When we examine the average run time criterion, GVNS has an overwhelming advantage over other algorithms. When we examine Table 8 for instances with 25 nodes, we can say that GVNS is superior in defined criteria except the deviation from the average of 20 runs. On the other hand, for the instances with 50 nodes, when we examine Tables 7 and 8, we observe that GVNS falls behind in average run time, unlike 25 node problems. However, when we examine the criteria for the best result found and the average of 20 runs, we can state that GVNS has more superiority over the other algorithms.

4.2.4 Graphical Comparison of Results

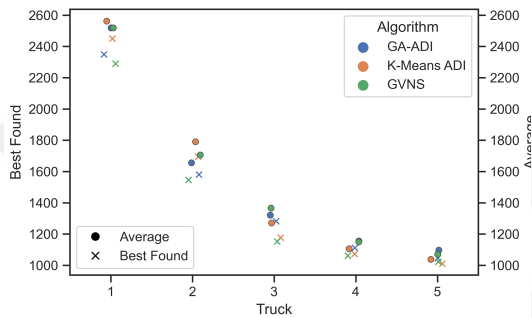
In this section we present the graphs that show the results of the algorithms we compared in the previous sections. In Figure 25, we present a comparison of the best results found by the algorithms for instances with 25 nodes and the average results obtained after 20 runs. In Figure 26, we share the same comparison for instances with 50 nodes.



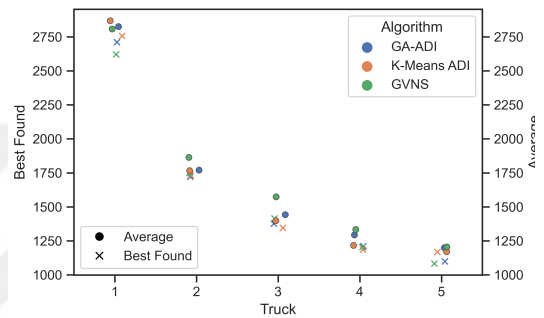
(a) Results of T1-25 Nodes



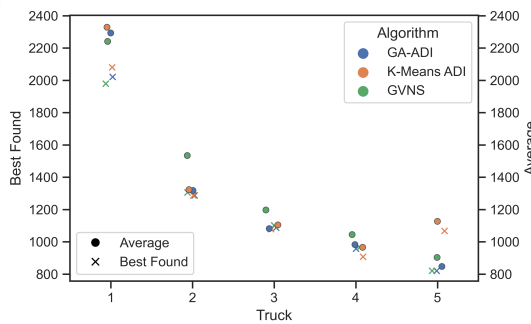
(b) Results of T2-25 Nodes



(c) Results of T3-25 Nodes

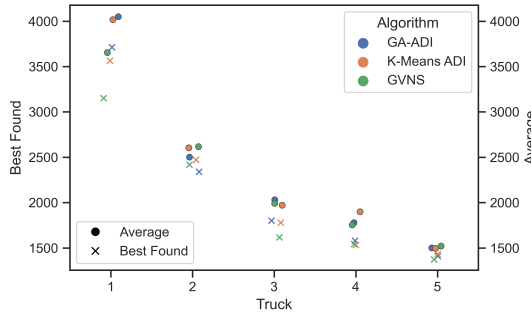


(d) Results of T4-25 Nodes

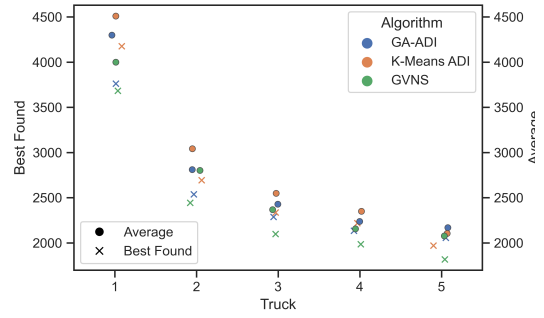


(e) Results of T5-25 Nodes

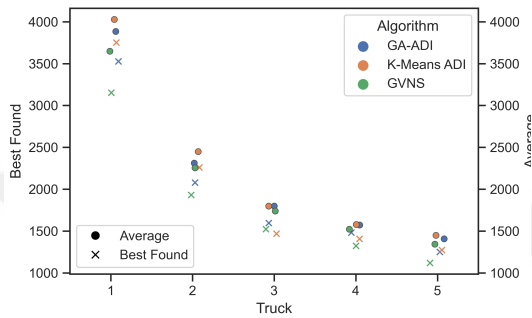
Figure 25: Comparison of Results for Instances with 25 Nodes



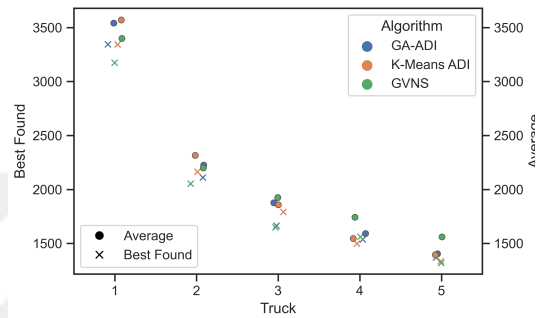
(a) Results of T1-50 Nodes



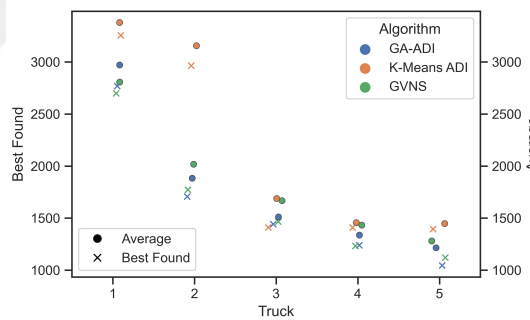
(b) Results of T2-50 Nodes



(c) Results of T3-50 Nodes



(d) Results of T4-50 Nodes



(e) Results of T5-50 Nodes

Figure 26: Comparison of Results for Instances with 50 Nodes

4.2.5 Small Instances

In this section, we present the results of our experiments on the small instances that Kitjacharoenchai et al. (2019) proposed. Small size instances are included in the experiments to show how successful the proposed GVNS algorithm is at finding optimal results. The results of 35 different 8'node instances shared by Kitjacharoenchai et al. (2019) are as follows.

Table 9: Comparison of Small Instances

Instance	MIP		GA-ADI				GVNS					
	Objective	Time (sec)	Best Objective	Best GAP (%)	Average	Average GAP (%)	Average Time (sec)	Best Objective	Best GAP (%)	Average	Average GAP (%)	Average Time (sec)
T1A	1442	111.3	1442	0.00%	1458.35	1.13%	4.01	1442	0.00%	1456.45	1.00%	1.28
T1B	1422	60.4	1422	0.00%	1423.65	0.12%	4.25	1422	0.00%	1496.90	5.27%	1.35
T1C	1350	62.42	1350	0.00%	1350.00	0.00%	4.61	1350	0.00%	1351.45	0.11%	1.26
T1D	1469	63.63	1469	0.00%	1469.00	0.00%	7.84	1469	0.00%	1517.35	3.29%	1.41
T1E	1193	112.22	1193	0.00%	1198.30	0.44%	5.63	1193	0.00%	1271.95	6.62%	1.26
T1F	1105	83.45	1105	0.00%	1105.00	0.00%	5.44	1105	0.00%	1197.40	8.36%	1.36
T1G	1186	67.49	1186	0.00%	1186.00	0.00%	7.53	1186	0.00%	1198.45	1.05%	1.30
T2A	1932	85.33	1932	0.00%	1937.90	0.31%	5.26	1932	0.00%	1999.85	3.51%	1.30
T2B	1662	92.45	1662	0.00%	1662.00	0.00%	5.38	1662	0.00%	1686.20	1.46%	1.34
T2C	1725	87.4	1725	0.00%	1733.95	0.52%	5.06	1725	0.00%	1844.50	6.93%	1.37
T2D	1721	76.58	1721	0.00%	1721.00	0.00%	5.76	1721	0.00%	1721.00	0.00%	1.32
T2E	1509	67.54	1509	0.00%	1509.00	0.00%	4.83	1509	0.00%	1530.45	1.42%	1.26
T2F	1880	73.44	1880	0.00%	1882.85	0.15%	5.82	1880	0.00%	1990.65	5.89%	1.33
T2G	1925	74.63	1925	0.00%	1925.00	0.00%	8.08	1925	0.00%	2044.10	6.19%	1.29
T3A	1181	86.2	1181	0.00%	1181.00	0.00%	5.07	1181	0.00%	1223.20	3.57%	1.34
T3B	1481	73.68	1481	0.00%	1498.85	1.21%	5.57	1481	0.00%	1562.75	5.52%	1.32
T3C	647	80.3	647	0.00%	647.35	0.05%	5.44	647	0.00%	743.15	14.86%	1.34
T3D	1261	71.69	1261	0.00%	1272.75	0.93%	4.46	1261	0.00%	1344.75	6.64%	1.29
T3E	1148	118.45	1148	0.00%	1148.00	0.00%	5.29	1148	0.00%	1167.00	1.66%	1.26
T3F	1426	73.44	1426	0.00%	1426.00	0.00%	5.74	1426	0.00%	1426.00	0.00%	1.28
T3G	1509	76.01	1509	0.00%	1512.90	0.26%	6.41	1509	0.00%	1532.90	1.58%	1.35
T4A	1613	77.6	1613	0.00%	1636.45	1.45%	3.97	1613	0.00%	1688.45	4.68%	1.31
T4B	1563	71.27	1563	0.00%	1575.45	0.80%	4.32	1563	0.00%	1658.55	6.11%	1.41
T4C	1481	79.88	1481	0.00%	1498.85	1.21%	5.57	1310	-11.55%	1444.95	-2.43%	1.44
T4D	1548	78.27	1548	0.00%	1552.50	0.29%	4.46	1548	0.00%	1565.70	1.14%	1.28
T4E	1589	97.25	1589	0.00%	1613.60	1.55%	6.27	1589	0.00%	1672.60	5.26%	1.25
T4F	1426	74.13	1426	0.00%	1426.00	0.00%	5.74	1652	15.85%	1739.60	21.99%	1.37
T4G	1358	74.47	1358	0.00%	1360.25	0.17%	7.02	1358	0.00%	1407.50	3.65%	1.29
T5A	1066	93.22	1066	0.00%	1066.00	0.00%	5.7	1066	0.00%	1081.65	1.47%	1.35
T5B	944	135.6	944	0.00%	944.00	0.00%	4.54	944	0.00%	944.00	0.00%	1.32
T5C	878	131.76	878	0.00%	878.00	0.00%	4.67	878	0.00%	885.25	0.83%	1.30
T5D	1133	123.22	1138	0.44%	1138.50	0.49%	4.68	1138	0.44%	1168.35	3.12%	1.35
T5E	1008	77.9	1014	0.60%	1018.55	1.05%	4.31	1014	0.60%	1030.30	2.21%	1.29
T5F	1270	95.32	1283	1.02%	1287.80	1.40%	4.38	1283	1.02%	1296.30	2.07%	1.36
T5G	1188	140.21	1202	1.18%	1215.80	2.34%	3.75	1202	1.18%	1243.30	4.65%	1.29

When we examine Table 9, we observe that the GVNS algorithm can reach optimal results in small size problems such as GA-ADI. As a result, we also prove the robustness of our proposed GVNS. On the other hand, we can say that the GVNS algorithm falls behind the GA-ADI in the averages obtained after 20 runs.

4.3 Lower Bounds of Problem Sets

In this section we present the LB values of the small sized problem sets in Table 10 and the LB values of the medium and large sized problem sets in Table 11.

Table 10: Lower Bounds of Small Sized Problems

Instance	Lower Bound
T1A	903
T1B	862
T1C	960
T1D	859
T1E	798
T1F	715
T1G	876
T2A	1337
T2B	1363
T2C	1213
T2D	1295
T2E	1318
T2F	1416
T2G	1458
T3A	817
T3B	751
T3C	458
T3D	787
T3E	702
T3F	913
T3G	895
T4A	948
T4B	972
T4C	846
T4D	781
T4E	941
T4F	1021
T4G	882
T5A	675
T5B	764
T5C	733
T5D	707
T5E	604
T5F	610
T5G	755

Table 11: Lower Bounds of Medium and Large Sized Problems

Instance	Node	Lower Bound
T1	25	833
T2	25	1381
T3	25	793
T4	25	911
T5	25	716
T1	50	913
T2	50	1431
T3	50	695
T4	50	780
T5	50	713

4.4 Dashboard

We also develop a Python based dashboard to facilitate the examination and monitoring of the results of our GVNS algorithm. With the help of this dashboard, we can observe the evolution of best solution of the selected instance over each iteration where an improvement is obtained. As seen in Figure 27, we can see the best found solution's visualisation on the left chart and we can see the improvement of the objective value over iterations on the right chart. Moreover, when we activate the details by using top-left switch, we can observe each truck routes' objective function values and solution vectors (See Figure 28). In addition to this, we can track each iteration by changing the iteration number using the slider at the top of the dashboard. After changing the iteration number, you will see that the charts dynamically update themselves according to the result in that iteration.

Figure 27: Dashboard Overview

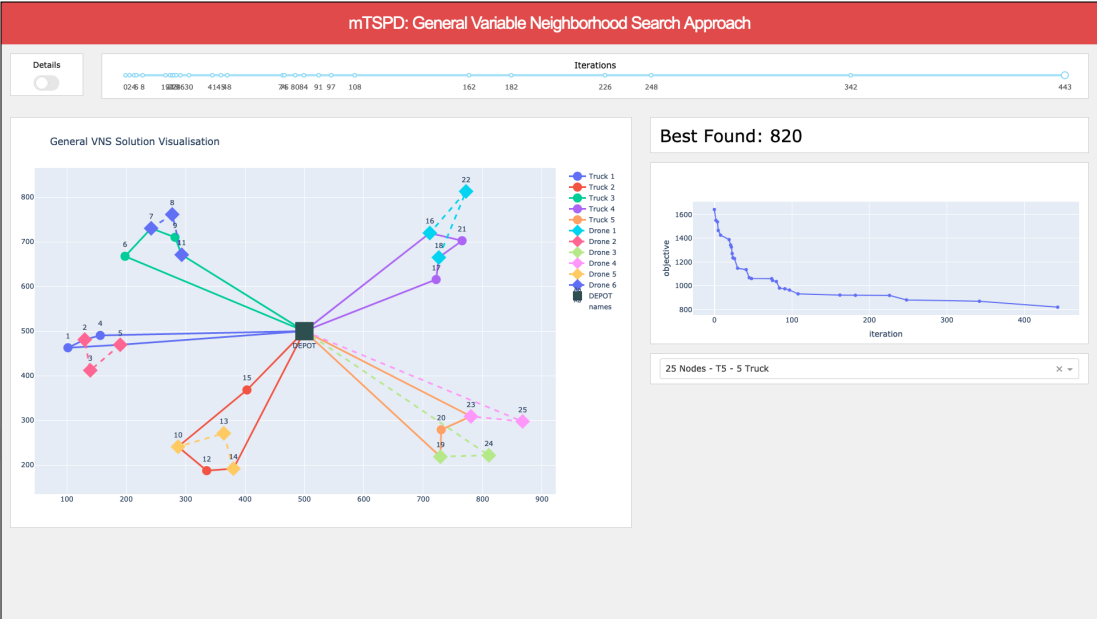


Figure 28: Dashboard Details

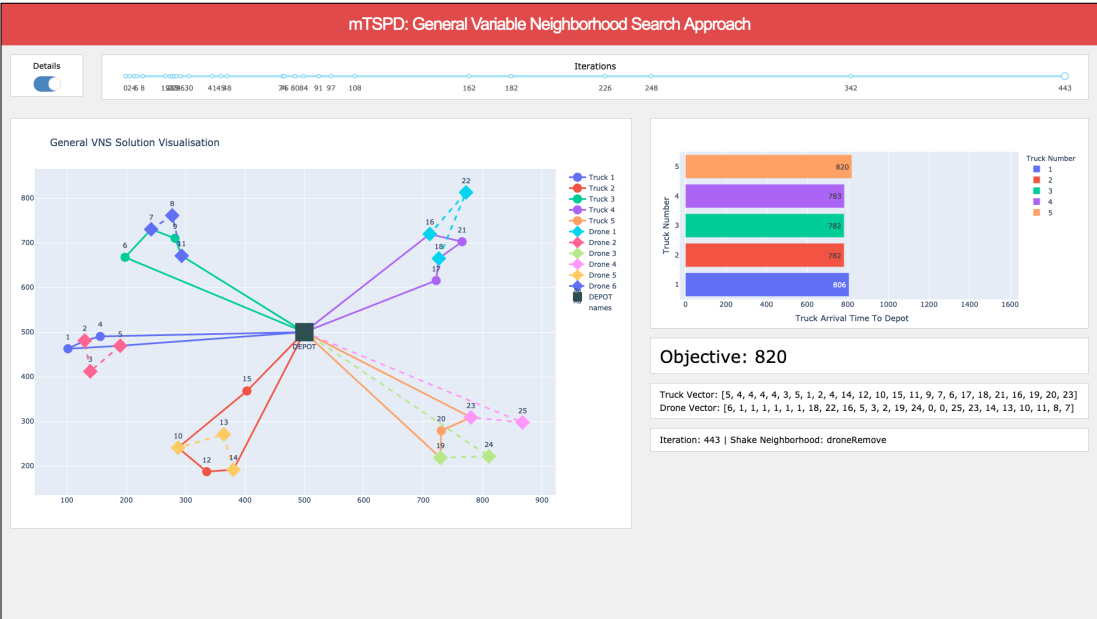


Table 12: The Results of The Algorithms in Instances with 25 Nodes

Instance	m	General VNS			GA-ADI			K-means ADI		
		Min. Obj.	Avg. Obj.	Avg. Time	Min. Obj.	Avg. Obj.	Avg. Time	Min. Obj.	Avg. Obj.	Avg. Time
T1	1	2707	2879.80	10.13	2788	2962.60	43.11	2909	3047.55	32.52
	2	1672	1840.45	25.04	1725	1816.05	52.25	1646	1795.65	40.73
	3	1313	1430.15	27.22	1350	1451.80	65.46	1249	1453.35	48.43
	4	1181	1255.75	32.24	1172	1243.55	72.91	1273	1296.35	56.17
	5	1067	1155.90	30.61	1133	1187.80	77.93	1136	1199.85	62.67
T2	1	2793	2941.50	13.26	2918	3076.10	43.67	3170	3258.75	29.35
	2	2093	2244.20	21.56	1887	2084.20	53.45	2503	2580.30	41.94
	3	1756	1909.95	32.34	1813	1970.80	67.01	1834	2085.20	47.76
	4	1647	1839.05	30.66	1757	1915.30	76.78	1831	1957.95	64.21
	5	1646	1834.50	28.27	1768	1842.95	80.92	1869	1930.50	66.68
T3	1	2290	2518.60	10.89	2349	2518.65	40.26	2450	2561.80	31.34
	2	1546	1705.45	20.20	1580	1655.50	50.52	1695	1790.20	41.07
	3	1152	1365.75	27.46	1282	1320.50	55.14	1177	1270.50	47.54
	4	1061	1149.20	35.40	1111	1155.55	61.53	1072	1105.65	55.90
	5	1023	1070.05	31.16	1045	1096.80	64.13	1010	1037.65	60.78
T4	1	2621	2808.15	12.74	2711	2825.85	42.11	2757	2868.80	31.92
	2	1747	1863.40	29.41	1721	1770.00	52.12	1732	1765.70	39.34
	3	1414	1573.30	30.98	1376	1441.85	66.16	1345	1397.70	46.81
	4	1203	1332.65	32.48	1210	1293.00	71.84	1185	1215.95	53.88
	5	1082	1204.55	34.84	1098	1198.95	73.53	1167	1169.70	54.39
T5	1	1979	2240.80	11.03	2021	2293.10	45.09	2080	2329.00	29.89
	2	1305	1534.05	23.77	1287	1317.60	54.58	1287	1322.90	39.10
	3	1101	1196.80	25.49	1087	1081.45	60.5	1078	1104.35	45.67
	4	965	1044.50	35.92	957	982.30	65.63	907	965.55	49.04
	5	820	902.95	39.46	820	846.95	71.14	1067	1126.45	58.35
Average				26.10			60.31			47.02

Table 13: The Results of The Algorithms in Instances with 50 Nodes

Instance	m	General VNS			GA-ADI			K-means ADI		
		Min Obj	Avg Obj	Avg Time	Min Obj	Avg Obj	Avg Time	Min Obj	Avg Obj	Avg Time
T1	1	3151	3653.55	97.72	3712	4049.45	92.99	3564	4017.10	89.11
	2	2418	2616.95	180.27	2340	2500.95	91.10	2472	2604.15	92.21
	3	1617	1991.45	139.06	1801	2031.30	103.69	1779	1971.05	101.60
	4	1543	1754.00	166.32	1580	1778.90	117.23	1532	1899.05	107.10
	5	1373	1521.05	150.75	1410	1500.65	128.08	1433	1495.95	118.40
T2	1	3682	3998.60	79.14	3761	4297.75	91.13	4175	4508.50	88.45
	2	2443	2801.50	134.62	2537	2810.35	98.80	2693	3041.35	86.68
	3	2098	2367.30	127.01	2287	2428.00	104.79	2336	2548.10	99.46
	4	1985	2155.45	153.79	2134	2237.10	106.60	2218	2349.40	105.50
	5	1817	2076.80	148.78	2056	2168.50	116.79	1970	2104.60	111.30
T3	1	3152	3647.45	79.85	3526	3883.65	91.17	3751	4027.25	102.90
	2	1930	2254.20	132.82	2078	2310.60	90.66	2259	2448.25	103.40
	3	1522	1738.00	127.70	1595	1797.80	98.90	1468	1797.18	111.50
	4	1322	1520.90	162.95	1481	1571.34	111.05	1406	1576.95	126.80
	5	1118	1342.10	149.53	1250	1406.15	124.85	1272	1447.85	126.80
T4	1	3174	3399.00	86.85	3344	3541.25	94.44	3343	3570.40	87.41
	2	2053	2199.35	146.76	2110	2225.75	92.99	2164	2315.15	91.75
	3	1648	1924.85	159.18	1664	1876.40	100.64	1791	1858.10	96.87
	4	1561	1742.20	159.23	1537	1591.25	111.35	1498	1545.50	105.90
	5	1318	1560.10	160.17	1369	1403.40	126.30	1335	1394.25	116.50
T5	1	2698	2806.25	99.91	2768	2970.35	94.97	3254	3378.75	86.03
	2	1772	2017.20	118.28	1707	1882.65	98.76	2964	3155.55	109.60
	3	1466	1667.60	119.50	1442	1511.10	105.24	1410	1687.95	95.12
	4	1232	1432.50	132.26	1239	1336.70	110.12	1409	1456.40	116.20
	5	1121	1280.40	129.68	1045	1215.30	125.02	1394	1447.45	117.20
Average				133.69			105.11			103.75

Table 14: Deviations of Algorithms from the Best Results in Instances with 25 Nodes

Instance	m	Best Solution	Deviation From Best Solution			Average Best	Deviation From Average Best			Average Run Time Best	Deviation From Average Run Time Best		
			GVNS	GA-ADI	K-means ADI		GVNS	GA-ADI	K-means ADI		GVNS	GA-ADI	K-means ADI
T1	1	2707	0.00%	2.99%	7.46%	2879	0.00%	2.90%	5.85%	10.13	0.00%	325.70%	221.12%
	2	1646	1.58%	4.80%	0.00%	1795.65	2.47%	1.14%	0.00%	25.04	0.00%	108.65%	62.64%
	3	1249	5.12%	8.09%	0.00%	1430	0.00%	1.52%	1.63%	27.22	0.00%	140.52%	77.95%
	4	1172	0.77%	0.00%	8.62%	1243.55	0.92%	0.00%	4.25%	32.24	0.00%	126.15%	74.22%
	5	1067	0.00%	6.19%	6.47%	1155	0.00%	2.84%	3.88%	30.61	0.00%	154.62%	104.76%
T2	1	2793	0.00%	4.48%	13.50%	2941	0.00%	4.59%	10.80%	13.26	0.00%	229.40%	121.38%
	2	1887	10.92%	0.00%	32.64%	2084.2	7.67%	0.00%	23.80%	21.56	0.00%	147.87%	94.49%
	3	1756	0.00%	3.25%	4.44%	1909	0.00%	3.24%	9.23%	32.34	0.00%	107.19%	47.67%
	4	1647	0.00%	6.68%	11.17%	1839	0.00%	4.15%	6.47%	30.66	0.00%	150.43%	109.43%
	5	1646	0.00%	7.41%	13.55%	1834	0.00%	0.49%	5.26%	28.27	0.00%	186.21%	135.84%
T3	1	2290	0.00%	2.58%	6.99%	2518	0.00%	0.03%	1.74%	10.89	0.00%	269.79%	187.86%
	2	1546	0.00%	2.20%	9.64%	1655.5	2.99%	0.00%	8.14%	20.20	0.00%	150.11%	103.33%
	3	1152	0.00%	11.28%	2.17%	1270.5	7.44%	3.94%	0.00%	27.46	0.00%	100.78%	73.10%
	4	1061	0.00%	4.71%	1.04%	1105.65	3.92%	4.51%	0.00%	35.40	0.00%	73.80%	57.90%
	5	1010	1.29%	3.47%	0.00%	1037.65	3.12%	5.70%	0.00%	31.16	0.00%	105.83%	95.08%
T4	1	2621	0.00%	3.43%	5.19%	2808	0.00%	0.64%	2.17%	12.74	0.00%	230.47%	150.50%
	2	1721	1.51%	0.00%	0.64%	1765.7	5.51%	0.24%	0.00%	29.41	0.00%	77.25%	33.79%
	3	1345	5.13%	2.30%	0.00%	1397.7	12.54%	3.16%	0.00%	30.98	0.00%	113.53%	51.08%
	4	1185	1.52%	2.11%	0.00%	1215.95	9.54%	6.34%	0.00%	32.48	0.00%	121.18%	65.88%
	5	1082	0.00%	1.48%	7.86%	1169.7	2.93%	2.50%	0.00%	34.84	0.00%	111.03%	56.10%
T5	1	1979	0.00%	2.12%	5.10%	2240	0.00%	2.37%	3.97%	11.03	0.00%	308.71%	170.93%
	2	1287	1.40%	0.00%	0.00%	1317.6	16.42%	0.00%	0.40%	23.77	0.00%	129.57%	64.46%
	3	1078	2.13%	0.83%	0.00%	1081.45	10.59%	0.00%	2.12%	25.49	0.00%	137.37%	79.18%
	4	907	6.39%	5.51%	0.00%	965.55	8.12%	1.73%	0.00%	35.92	0.00%	82.69%	36.51%
	5	820	0.00%	0.00%	30.12%	846.95	6.50%	0.00%	33.00%	39.46	0.00%	80.26%	47.85%
# of Best Average			14	5	8	10	6	9	25	0	0		
			1.51%	3.44%	6.66%	4.03%	2.08%	4.91%	0.00%	150.76%	92.92%		

Table 15: Deviations of Algorithms from the Best Results in Instances with 50 Nodes

Instance	m	Best Solution	Deviation From Best Solution			Average Best	Deviation From Average Best			Average Run Time Best	Deviation From Average Run Time Best		
			GVNS	GA-ADI	K-means ADI		GVNS	GA-ADI	K-means ADI		GVNS	GA-ADI	K-means ADI
T1	1	3151	0.00%	17.80%	13.11%	3653	0.00%	10.85%	9.97%	89.11	9.66%	4.35%	0.00%
	2	2340	3.33%	0.00%	5.64%	2500.95	4.60%	0.00%	4.13%	91.10	97.88%	0.00%	1.22%
	3	1617	0.00%	11.38%	10.02%	1971.05	1.01%	3.06%	0.00%	101.60	36.87%	2.06%	0.00%
	4	1532	0.72%	3.13%	0.00%	1754	0.00%	1.42%	8.27%	107.10	55.30%	9.46%	0.00%
	5	1373	0.00%	2.69%	4.37%	1495.95	1.67%	0.31%	0.00%	118.40	27.32%	8.18%	0.00%
T2	1	3682	0.00%	2.15%	13.39%	3998	0.00%	7.50%	12.77%	79.14	0.00%	15.14%	11.76%
	2	2443	0.00%	3.85%	10.23%	2801	0.00%	0.33%	8.58%	86.68	55.31%	13.98%	0.00%
	3	2098	0.00%	9.01%	11.34%	2367	0.00%	2.58%	7.65%	99.46	27.70%	5.36%	0.00%
	4	1985	0.00%	7.51%	11.74%	2155	0.00%	3.81%	9.02%	105.50	45.78%	1.04%	0.00%
	5	1817	0.00%	13.15%	8.42%	2076	0.00%	4.46%	1.38%	111.30	33.68%	4.93%	0.00%
T3	1	3152	0.00%	11.87%	19.00%	3647	0.00%	6.49%	10.43%	79.85	0.00%	14.18%	28.87%
	2	1930	0.00%	7.67%	17.05%	2254	0.00%	2.51%	8.62%	90.66	46.51%	0.00%	14.05%
	3	1468	3.68%	8.65%	0.00%	1738	0.00%	3.44%	3.41%	98.90	29.12%	0.00%	12.74%
	4	1322	0.00%	12.03%	6.35%	1520	0.00%	3.38%	3.75%	111.05	46.74%	0.00%	14.18%
	5	1118	0.00%	11.81%	13.77%	1342	0.00%	4.78%	7.89%	124.85	19.77%	0.00%	1.56%
T4	1	3174	0.00%	5.36%	5.32%	3399	0.00%	4.19%	5.04%	86.85	0.00%	8.74%	0.65%
	2	2053	0.00%	2.78%	5.41%	2199	0.00%	1.22%	5.28%	91.75	59.96%	1.35%	0.00%
	3	1648	0.00%	0.97%	8.68%	1858.1	3.55%	0.98%	0.00%	96.87	64.32%	3.89%	0.00%
	4	1498	4.21%	2.60%	0.00%	1545.5	12.71%	2.96%	0.00%	105.90	50.36%	5.15%	0.00%
	5	1318	0.00%	3.87%	1.29%	1394.25	11.89%	0.66%	0.00%	116.50	37.49%	8.41%	0.00%
T5	1	2698	0.00%	2.59%	20.61%	2806	0.00%	5.86%	20.41%	86.03	16.14%	10.39%	0.00%
	2	1707	3.81%	0.00%	73.64%	1882.65	7.14%	0.00%	67.61%	98.76	19.77%	0.00%	10.98%
	3	1410	3.97%	2.27%	0.00%	1511.1	10.32%	0.00%	11.70%	95.12	25.63%	10.64%	0.00%
	4	1232	0.00%	0.57%	14.37%	1336.7	7.13%	0.00%	8.95%	110.12	20.11%	0.00%	5.52%
	5	1045	7.27%	0.00%	33.40%	1215.3	5.32%	0.00%	19.10%	117.20	10.65%	6.67%	0.00%
# of Best Average			18	3	4	15	5	5	3	7	15		
			1.08%	5.75%	12.29%	2.61%	2.83%	9.36%	33.44%	5.36%	4.06%		

CHAPTER 5: CONCLUSION

In this thesis, we have stated the new optimization problems that occur with the entry of the unmanned aerial vehicles into the cargo transportation sector with the help of developing technology. First of all, we developed a mathematical model consisting of a single drone and a single truck tandem. This problem definition first entered the literature with FSTSP, afterwards this problem examined from a different perspective with TSP-D. Since the mTSPD problem proposed by Kitjacharoenchai et al. (2019) is the most comprehensive TSP problem in the literature, including multiple drones and multiple trucks, we focused our study on mTSPD problem.

In this direction, we started to work on the VNS algorithm and its variants, which is a fast and effective metaheuristic, and we proposed the GVNS algorithm in order to obtain robust and good results for mTSPD in reasonable time. We used the datasets generated by Kitjacharoenchai et al. (2019) to test our GVNS algorithm and compare its efficiency and robustness with the algorithms they proposed.

The datasets that we used in our experiments are divided into 3 main categories as small, medium and large according to the number of nodes, which are 8, 25 and 50 nodes, respectively. In addition, these problem sets are divided into 5 sub-problems, named as T1, T2, T3, T4 and T5, according to the location of the customers and the depot in given area.

As a result of our experiments on these given problem sets, we compared the results we obtained on the basis of 3 different criteria. These are determined as best found solution, average objective function value after 20 runs and average run time of algorithm. According to the experiments we have done, the GVNS algorithm we have proposed has found better results in 32 instances out of 50 instances. It also performed better on the average objective function value after 20 runs across 25 instances.

For future research, we can modify our proposed GVNS algorithm for different problem types. We can change the objective function of the problem and specify multiple objectives for different conditions. For example, we can expand the problem by creating a objective that minimizes cost for situations where cost is important and different objective that minimizes time for emergencies. In addition, the charging capacities of drones can also be included in the problem and we can include the

heterogeneous drone fleet in the problem. This will provide a more realistic situation and introduce more different types of problems, as different types of drones will provide variation in flight distances and maximum speeds.



REFERENCES

- Agatz, N., Bouman, P. and Schmidt, M. (2018), 'Optimization approaches for the traveling salesman problem with drone', *Transportation Science* 52(4), pp. 739–1034.
- Bouman, P., Agatz, N. and Schmidt, M. (2018), 'Dynamic programming approaches for the traveling salesman problem with drone', *Networks* 72(4), pp. 528–542.
- Chao, I.-M. (2002), 'A tabu search method for the truck and trailer routing problem', *Computers & Operations Research* 29(1), pp. 33–51.
- Croes, G. A. (1958), 'A method for solving traveling-salesman problems', *Operations Research* 6(6), pp. 791–812.
- Current, J. R. and Schilling, D. A. (1989), 'The covering salesman problem', *Transportation Science* 23(3), pp. 208–213.
- de Freitas, J. C. and Penna, P. H. V. (2020), 'A variable neighborhood search for flying sidekick traveling salesman problem', *International Transactions in Operational Research* 27(1), pp. 267–290.
- Dorling, K., Heinrichs, J., Messier, G. and Magierowski, S. (2016), 'Vehicle routing problems for drone delivery', *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 47, pp. 1–16.
- Ferrandez, S., Harbison, T., Weber, T., Sturges, R. and Rich, R. (2016), 'Optimization of a truck-drone in tandem delivery network using k-means and genetic algorithm', *Journal of Industrial Engineering and Management* 9(2), pp. 374–388.
- Ha, Q. M., Deville, Y., Pham, Q. D. and Hà, M. H. (2018), 'On the min-cost traveling salesman problem with drone', *Transportation Research Part C: Emerging Technologies* 86, pp. 597–621.
- Hansen, P. and Mladenović, N. (2001), 'Variable neighborhood search: Principles and applications', *European Journal of Operational Research* 130(3), pp. 449–467.
- Hansen, P. and Mladenović, N. (2018), 'Variable neighborhood search', *Handbook of Heuristics*, (R. Martí, P. Pardalos, M. Resende, eds.), Springer, Cham pp. 759–787.
- Hansen, P., Mladenovic, N., Todosijević, R. and Hanafi, S. (2017), 'Variable neighborhood search: basics and variants', *EURO Journal on Computational Optimization* 5, pp. 423–454.

- Kitjacharoenchai, P., Ventresca, M., Moshref-Javadi, M., Lee, S., Tanchoco, J. M. and Brunese, P. A. (2019), 'Multiple traveling salesman problem with drones: Mathematical model and heuristic approach', *Computers & Industrial Engineering* 129, pp. 14–30.
- Lin, C. (2011), 'A vehicle routing problem with pickup and delivery time windows, and coordination of transportable resources', *Computers & Operations Research* 38(11), pp. 1596–1609.
- Lin, S.-W., Yu, V. F. and Chou, S.-Y. (2009), 'Solving the truck and trailer routing problem based on a simulated annealing heuristic', *Computers & Operations Research* 36(5), pp. 1683–1692. Selected papers presented at the Tenth International Symposium on Locational Decisions (ISOLDE X).
- Mjirda, A., Todosijević, R., Hanafi, S., Hansen, P. and Mladenovic, N. (2016), 'Sequential variable neighborhood descent variants: An empirical study on the traveling salesman problem', *International Transactions in Operational Research* 24(3), pp. 615–633.
- Mladenović, N. and Hansen, P. (1997), 'Variable neighborhood search', *Computers & Operations Research* 24(11), pp. 1097–1100.
- Murray, C. C. and Raj, R. (2020), 'The multiple flying sidekicks traveling salesman problem: Parcel delivery with multiple drones', *Transportation Research Part C: Emerging Technologies* 110, pp. 368–398.
- Murray, C. and Chu, A. (2015), 'The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery', *Transportation Research Part C: Emerging Technologies* 54, pp. 86–109.
- Poikonen, S., Wang, X. and Golden, B. (2017), 'The vehicle routing problem with drones: Extended models and connections', *Networks* 70(1), pp. 34–43.
- Ponza, A. (2016), Optimization of drone-assisted parcel delivery, Master's thesis, Università Degli Studi Di Padova.
- Soylu, B. (2015), 'A general variable neighborhood search heuristic for multiple traveling salesmen problem', *Computers & Industrial Engineering* 90, pp. 390–401.
- Wang, X., Poikonen, S. and Golden, B. (2017), 'The vehicle routing problem with drones: Several worst-case results', *Optimization Letters* 11.