ALNS ALGORITHM FOR LOAD HANDLING AND AGV ROUTING WITH TROLLEYS

A Thesis

by

Reşide Özkır

Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in the
Department of Industrial Engineering

Özyeğin University
September 2024

ALNS ALGORITHM FOR LOAD HANDLING AND AGV ROUTING WITH TROLLEYS

Approved by:

Assoc. Prof. Elvin Çoban Göktürk,
Advisor
Dept. of Industrial Eng.
Özyeğin University

Prof. Ali Ekici
Dept. of Industrial Eng.
Özyeğin University

Asst. Prof. Tonguç Yavuz
Dept. of Industrial Eng.
Bilgi University

Date Approved: August 9, 2024

This thesis is dedicated to my beloved husband, whose unwavering support and encouragement made this journey possible. Your patience, love, and understanding have been my pillars of strength.

I also dedicate this work to our little one, who is on the way. You have already brought so much joy and inspiration into my life, and I can't wait to share all my accomplishments with you.

ABSTRACT

Automated Guided Vehicles (AGVs) play a critical role in industrial logistics, providing efficient material handling solutions. This study addresses a real-world problem of AGV routing and load handling at Vestel Electronics, specifically focusing on their refrigerator factory, which produces a diverse range of models, including no-frost refrigerators, minibars, and built-in refrigerators. The diversity in refrigerator models necessitates the transportation of various materials such as refrigerator doors, compressors, and shelves within strict delivery time windows. Given the limited number of AGVs and trolleys available to increase their load capacity, determining the optimal routing and scheduling of pickups and deliveries presents a complex challenge. To address this, we develop a Mixed Integer Linear Programming (MILP) model to formalize the AGV routing and load handling problem, incorporating trolley allocation and aiming to minimize total costs, including travel, earliness, and tardiness penalties. The model also considers practical constraints like time windows and load capacities, making it suitable for real-world applications. However, due to the computational inefficiency of the MILP model in solving medium to large-sized instances, we propose an Adaptive Large Neighborhood Search (ALNS) algorithm. This algorithm, with its adaptive mechanisms, efficiently explores diverse neighborhood structures, finding near-optimal solutions. Extensive computational experiments show that the ALNS algorithm is robust across varying instance sizes, delivering efficient solutions and reducing computation times significantly. A real-life case study at Vestel Electronics highlights the approach's potential, reducing operational costs to as low as 1/48th of the current costs, thus offering substantial cost savings and operational efficiency.

ÖZETÇE

Otomatik Kılavuzlu Araçlar (AGV'ler), endüstriyel lojistikte kritik bir rol oynayarak etkili malzeme taşıma çözümleri sunmaktadır. Bu çalışma, Vestel Elektronik'in buzdolabı fabrikasında AGV yönlendirme ve yük taşıma problemini ele almakta olup, no-frost buzdolapları, minibarlar ve ankastre buzdolapları gibi çeşitli modeller üreten bu fabrikada karşılaşılan zorluklara odaklanmaktadır. Buzdolabı modellerindeki çeşitlilik, buzdolabı kapıları, kompresörler ve raflar gibi çeşitli malzemelerin belirli teslimat zaman aralıkları içerisinde taşınmasını zorunlu kılmaktadır. AGV'lerin ve yük kapasitelerini artırmak için kullanılan yük taşıma ünitelerinin sınırlı sayıda olması, yükleme ve teslimatların optimal yönlendirme ve zamanlamasını zorlaştırmaktadır. Bu sorunu çözmek için, AGV yönlendirme ve yük taşıma problemini modelleyen ve seyahat, erken varış ve geç varış cezalarını minimize etmeyi amaçlayan bir Karışık Tamsayılı Doğrusal Programlama (MILP) modeli geliştirilmiştir. Zaman pencereleri ve yük kapasiteleri gibi pratik kısıtları dikkate alarak model uygulanabilir hale getirilmiştir. Ancak, MILP modelinin orta ve büyük boyutlu örneklerde hesaplama açısından verimsiz olması nedeniyle, uyarlamalı büyük komşuluk arama (ALNS) algoritması alternatif bir çözüm yaklaşımı olarak önerilmiştir. Bu algoritma, adaptif mekanizmalarla çeşitli komşuluk yapılarını etkili bir şekilde keşfederek neredeyse optimal çözümler bulur. Geniş çaplı hesaplama deneyleri, ALNS algoritmasının farklı örnek boyutlarında sağlam olduğunu ve hesaplama sürelerini önemli ölçüde azaltarak etkili çözümler sunduğunu göstermektedir. Vestel Elektronik'te yapılan bir saha çalışması, bu yaklaşımın operasyonel maliyetleri mevcut maliyetlerin 1/48'ine kadar azaltabileceğini ve bu sayede önemli maliyet tasarrufları ve operasyonel verimlilik sağlama potansiyelini ortaya koymaktadır.

ACKNOWLEDGEMENTS

I wish to express my deepest gratitude to Assoc. Prof. Elvin Çoban Göktürk, my thesis advisor, for her invaluable guidance, unwavering support, and profound expertise. Her insights have been instrumental in shaping this research and bringing it to fruition.

My heartfelt thanks go to my husband, Onur Özkır, for his steadfast encouragement, patience, and love, which have been a constant source of strength throughout this journey. Your belief in me has made this achievement possible.

I am also profoundly grateful to Vestel Electronics for providing the essential equipment and technical expertise that were crucial to this study. The support provided by your organization has played a significant role in the success of this research.

To my family, your unwavering love and support have been the foundation upon which I have built my academic pursuits. I am eternally grateful for your presence in my life.

Lastly, I express my sincere gratitude to all the authors, researchers, and scholars whose works I have cited in this thesis. Their groundbreaking contributions have paved the way for academic progress and have profoundly influenced my perspectives. It is my earnest hope that this thesis will also serve as a meaningful contribution to fellow researchers. With deep appreciation, I conclude this acknowledgment.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

AGVs are increasingly becoming vital in industrial logistics, providing essential services in material handling and transport within warehouses, manufacturing plants, and distribution centers [1]. Vestel Electronics, a leading manufacturer of consumer electronics and household appliances, has integrated AGVs into its operations to enhance productivity and operational flexibility [2, 3]. However, the complexities associated with AGV routing and load handling present ongoing challenges that require innovative solutions.

1.1 Industry Background

The Vestel Group, comprising 28 companies and over 20,000 employees, is a global leader in consumer electronics, household appliances, and other technology solutions [4]. Since its expansion into white goods in 1997, refrigerators have been one of Vestel's primary products, with production capacities reaching up to 12,000 units per day [5]. In the refrigerator factory, AGVs are critical for transporting components, such as, refrigerator parts to various workstations, as shown in Figure 1.

Figure 1: AGV in Vestel Electronics Factory (1)

1.2 Optimization Challenges in AGV Systems

Optimizing AGV routing and scheduling is crucial to ensure timely and efficient deliveries. Both late and early deliveries can be problematic, with early deliveries potentially blocking workstations [6]. The importance of optimizing AGV operations has been recognized in the literature, where various models and algorithms have been proposed to address the routing and scheduling challenges in dynamic and complex environments. Additionally, trolleys attached to AGVs provide flexibility since AGVs can carry more load (see Figure 2). However, trolley attachments complicate the AGV routing and load handling, as the number of trolleys to be attached to each AGV should also be optimized.

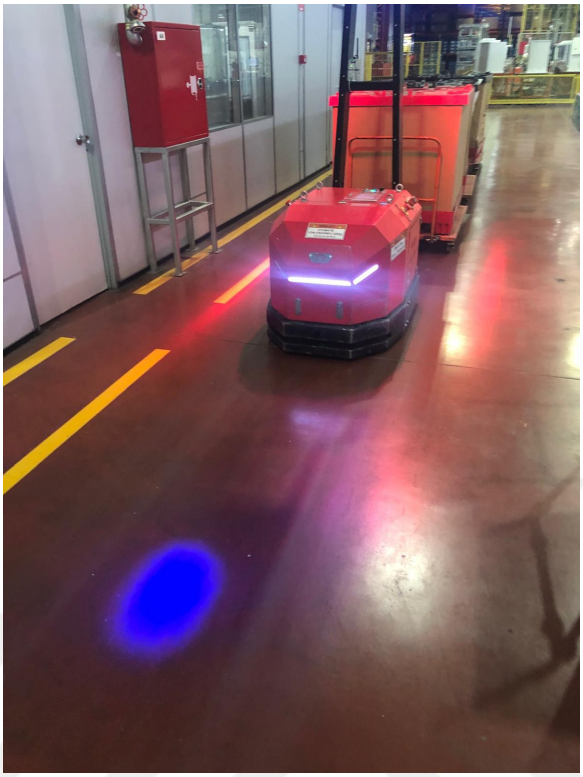

Figure 2: AGV in Vestel Electronics Factory (2)

This thesis integrates the specific challenges faced by Vestel Electronics, such as the diversity of components and the use of trolleys. Previous models have often focused on more generic problems, but this study aims to provide a more tailored solution to a real-life industrial application.

1.3 Research Motivation

This study addresses the challenges associated with AGV routing and load handling through a real-life problem at Vestel Electronics, focusing specifically on their refrigerator factory, where AGVs play a crucial role in material handling. In this factory, AGVs are essential for transporting components, such as screws and refrigerator doors, including both freezer and cooler doors, to various workstations (as depicted in Figure 3). Given the diverse nature of the materials and the critical importance of timing in their delivery, optimizing the AGV routing and scheduling processes is

vital for maintaining production efficiency.

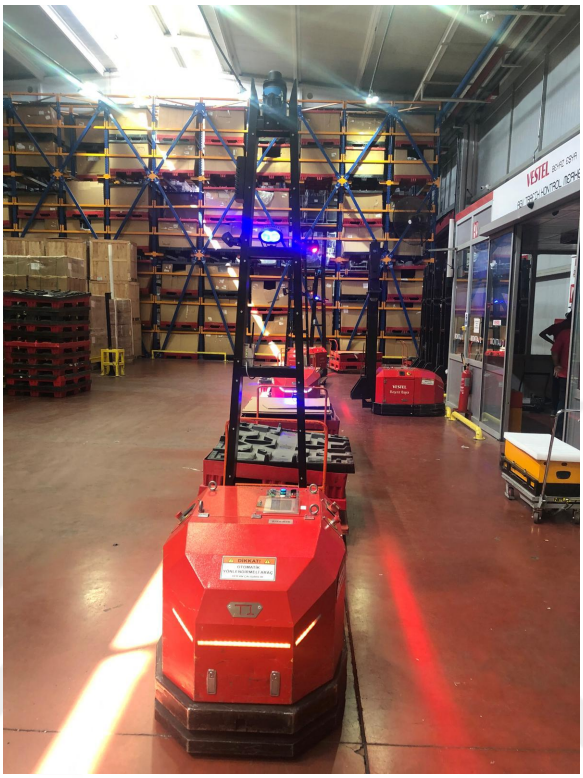

Figure 3: AGV in Vestel Electronics Factory (3)

In this research, we consider the operational decision-making process regarding the attachment of trolleys to AGVs, similar to the motivated problem at Vestel Electronics. We develop a MILP model integrating the AGV routing and load handling problems while considering trolleys and delivery time windows. Our objective is to minimize the total cost of travel, earliness, and tardiness of deliveries. Then, we propose an ALNS algorithm and demonstrate its efficiency and performance through a detailed computational analysis. The case study on Vestel Electronics illustrates that the implementation of our solution could reduce operational costs significantly.

This research contributes to the broader field of AGV optimization by offering a practical solution tailored to the specific needs of an industrial setting, which can also be adapted to similar environments.

CHAPTER II

LITERATURE REVIEW

The study of AGVs has attracted considerable research interest over the past few decades, largely due to the increasing demand for efficient material handling and transport solutions in industrial environments [7]. The primary challenge in AGV systems is to optimize the routing of multiple AGVs while managing the specific requirements of the loads they carry, such as meeting strict delivery time windows [8]. These AGVs must also fulfill their tasks while minimizing travel time and avoiding collisions, all within complex and dynamic industrial environments. Such environments require AGVs to navigate around unforeseen obstacles and adapt to continuously changing demands, making routing and load handling particularly challenging problems.

Early research in AGV routing has focused on traditional pathfinding algorithms, such as Dijkstra's algorithm [9, 10] and the A* algorithm [11, 12]. While these algorithms are effective in static and predictable environments, they are often inadequate for real-time adjustments and complexities encountered in dynamic industrial settings, such as unexpected obstacles and changes in demand [3]. To overcome these limitations, heuristic algorithms have gained prominence, as they provide approximate solutions where exact algorithms may be computationally impractical, especially in large-scale applications [13], [3]. Among these heuristics, genetic algorithms (GA) [14, 15], simulated annealing (SA) [16], ant colony optimization (ACO) [17], and Adaptive Large Neighborhood Search (ALNS) [18] have demonstrated promising results.

AGV routing problems are often studied within the broader context of vehicle

routing problems with simultaneous pickup and delivery (VRPSPD) [3]. [19] highlight the increased complexity of VRPSPD and review various meta-heuristic approaches that have been proposed to address these challenges effectively. For example, [20] explore the integration of location decisions and routing for simultaneous pickups and deliveries using a branch-and-cut algorithm, incorporating several valid inequalities and a local search procedure based on simulated annealing. A comprehensive review of the literature on the simultaneous pickup and delivery problem can be found in the works of [21].

ALNS has been particularly noted for its effectiveness in solving pickup and delivery problems with time windows, as it offers a flexible and adaptive search process capable of navigating large solution spaces by systematically destroying and repairing parts of the solution [22]. Although ALNS can be integrated with other heuristics, such as in the work of [23] who developed a hybrid ALNS algorithm combined with genetic algorithms to address various vehicle routing problems, it is also effective when used independently. For example, [24] applied ALNS to a mixed fleet vehicle routing problem with recharging constraints, and [25] utilized ALNS to address the multi-depot dynamic vehicle routing problem with time windows, demonstrating its effectiveness in dynamically re-optimizing routes in response to emerging customer demands. Similarly, [26] employed ALNS in location-routing planning for cash transportation, showcasing its ability to optimize complex logistics operations under risk constraints.

Routing AGVs presents unique challenges compared to traditional VRPSPD due to the specific characteristics of the environments in which AGVs operate. AGVs may need to navigate through unidirectional or bidirectional paths, which significantly impact routing decisions. Additionally, battery life is a critical factor, requiring AGVs to complete their routes within the constraints of battery depletion [27]. Speed limitations are also crucial, particularly in scenarios where products must be delivered

within strict time windows, such as in manufacturing environments or large-scale facilities like container ports [28]. In such cases, early arrival of AGVs can result in blockages, disrupting workflows, while late arrivals may lead to production delays, adding further complexity to the problem. Furthermore, AGVs are deployed across diverse settings, from industrial production facilities to extensive port operations, each presenting unique operational challenges. Unlike traditional vehicle routing problems, AGV fleets are often heterogeneous, comprising different types of AGVs with varying capabilities and constraints [29], adding another layer of complexity to the routing problem.

Recent advancements in the field have also seen the integration of machine learning techniques with traditional optimization methods to enhance AGV performance. For instance, [30] proposed a deep reinforcement learning approach to optimize AGV path planning in complex environments, achieving significant improvements in route efficiency and collision avoidance. Similarly, [31] combined deep learning with genetic algorithms to dynamically adjust AGV routes in real-time based on environmental changes and fluctuating demand.

Despite the significant progress made in AGV routing and load handling, the specific problem of AGV routing and load handling considering the attachment of additional trolleys, as encountered in real-life manufacturing environments such as Vestel Electronics, has not been extensively studied in the literature. This problem introduces a unique complexity not addressed by the general consideration of heterogeneous AGVs, as the number of trolleys attached to each AGV must also be optimized. Each trolley attachment increases the AGV's travel time, creating a tradeoff between the ability to pick up and deliver more items and the associated travel time. Our research addresses this gap by proposing an ALNS algorithm that optimizes AGV routing and load handling, considering the dynamic nature of fluctuating demand. We conclude that the computational efficiency and robustness of the

ALNS algorithm make it well-suited for daily operations in dynamic environments, enabling real-time decisions regarding trolley attachments and improving the overall efficiency of AGV operations.

CHAPTER III

PROBLEM DEFINITION

In this study, we address the AGV routing and load handling problem within the context of Vestel Electronics' refrigerator manufacturing operations, with a particular focus on the complexities introduced by trolley attachments and the diverse range of refrigerator models produced. The AGVs are deployed to transport various refrigerator components across different workstations, adhering to strict delivery time windows, while navigating the factory floor via predefined paths that must also account for potential obstacles. The primary objective is to determine the optimal routing of AGVs to minimize the total operational cost, which includes both the travel cost and the penalty costs associated with early or late deliveries.

To increase their load capacity, AGVs at Vestel Electronics are often equipped with trolleys, which allow for the simultaneous transport of multiple components. These trolleys, which can vary in design and size, play a crucial role in optimizing material flow within the factory. Figure 4 illustrates a typical example of the trolleys used in this context.

Figure 4: An example of trolleys used in Vestel Electronics' refrigerator manufacturing operations

There is a limited number of AGVs that are expected to satisfy the predefined n orders where each order i is defined by a pickup node i ($i \in P$ where $P = \{1, .., n\}$) and the corresponding delivery node $n + i$ at the factory layout. In other words, delivery points are from the set D where $D = \{n + 1, .., 2n\}$. There is also a limited number of trolleys that can be attached to AGVs. However, attaching a trolley and increasing the capacity of an AGV also cause a decrease in the maneuverability of the AGV which is captured by a constant increase in the traveling time between any of the two nodes in the factory. Hence, Vestel Electronics aims to compute when to utilize trolleys and how many trolleys will be utilized for each AGV. Additionally, overloading AGVs is not permitted due to safety rules, however, an AGV can carry less than its capacity.

We define K be the set of AGVs, and AGVs may not service all demand requests as there may be no possible path that AGVs can use between two workstations at the factory due to safety reasons. Hence, each AGV k has a specific set of nodes that it

can visit, defined by the set N_k, $N_k = P_k \cup D_k$ where N_k, P_k, and D_k are appropriate subsets of N (i.e., $P \cup D$), P, and D, respectively. We define the network of each AGV by G_k where $G_k = (V_k, A_k)$ when V_k is defined as the union of N_k and the origin depot and the destination depot of AGV k (i.e., $\{(o_k, d_k)\}$). A_k represents the set of all feasible arcs defined as $V_k x V_k$. The nomenclature is in Table 1.

Table 1: Nomenclature

Indices	
k	AGV, $k \in K$
i	pickup node if $i \in P$; delivery node if $i \in D$
Parameters	
α	Cost of penalty for one unit of earliness
β	Cost of penalty for one unit of tardiness
C	Capacity of a trolley
M_{max}	Total number of trolleys
TIM	Trolley impact time
t_{ijk}	Cost of per unit travel time from node i to node j by AGV k
s_i	Service time at node i
a_i	Earliest delivery time defined for node i
b_i	Latest delivery time defined for node i
l_i	Load to be picked up or delivered at node i
o_k	Origin depot of AGV k
d_k	Destination depot of AGV k
P	Set of all pickup nodes
D	Set of all delivery nodes
P_k	Set of pickup nodes that can be served by AGV k
D_k	Set of delivery nodes that can be served by AGV k
N_k	All pickup and delivery nodes that can be served by AGV k, $N_k = P_k \cup D_k$
A_k	Pairs of routes that can be traveled by AGV k
Decision Variables	
x_{ijk}	1 if arc $(i, j) \in A_k$ is used by AGV k; 0 otherwise
T_{ik}	Service starting time of AGV k at node i
L_{ik}	Load of AGV k after service at node i
Y_k	Number of trolleys attached to the AGV k
TA_i	Tardiness of delivery at node i, $i \in D$
E_i	Earliness of delivery at node i, $i \in D$

The proposed MILP model is formulated as follows.

$$\min_{k \in K} \sum_{(i,j) \in A_k} t_{ijk} x_{ijk} + \alpha \sum_{i \in D} E_i + \beta \sum_{i \in D} TA_i$$

s.t.

$$\sum_{k \in K} \sum_{j \in N_k \cup \{d_k\}} x_{ijk} = 1 \quad \forall i \in P \tag{1}$$

$$\sum_{j \in N_k} x_{ijk} - \sum_{j \in N_k} x_{j(n+i)k} = 0 \quad \forall k \in K, \forall i \in P_k \tag{2}$$

$$\sum_{j \in P_k \cup \{d_k\}} x_{o_k jk} = 1 \quad \forall k \in K \tag{3}$$

$$\sum_{i \in N_k \cup \{o_k\}} x_{ijk} - \sum_{i \in N_k \cup \{d_k\}} x_{jik} = 0 \quad \forall k \in K, \forall j \in N_k \tag{4}$$

$$\sum_{i \in D_k \cup \{o_k\}} x_{id_k k} = 1 \quad \forall k \in K \tag{5}$$

$$E_i \geq a_i - T_{ik} \quad \forall k \in K, \forall i \in D_k \tag{6}$$

$$E_i \geq 0 \quad \forall i \in D \tag{7}$$

$$TA_i \geq T_{ik} - b_i \quad \forall k \in K, \forall i \in D_k \tag{8}$$

$$TA_i \geq 0 \quad \forall i \in D \tag{9}$$

$$T_{ik} + (s_i + t_{ijk})x_{ijk} + (Y_k \cdot TIM) \leq T_{jk} + M(1 - x_{ijk}) \quad \forall k \in K, \forall (i,j) \in A_k \tag{10}$$

$$T_{ik} + t_{i(n+i)k} + (Y_k \cdot TIM) \leq T_{(n+i)k} \quad \forall k \in K, \forall i \in P_k \tag{11}$$

$$L_{ik} + l_j \cdot x_{ijk} - L_{jk} \leq M(1 - x_{ijk}) \quad \forall k \in K, \forall (i,j) \in A_k \tag{12}$$

$$-L_{ik} - l_j \cdot x_{ijk} + L_{jk} \leq M(1 - x_{ijk}) \quad \forall k \in K, \forall (i,j) \in A_k \tag{13}$$

$$l_i \leq L_{ik} \leq Y_k \cdot C \quad \forall k \in K, \forall i \in P_k \tag{14}$$

$$0 \leq L_{(n+i)k} \leq Y_k \cdot C - l_i \quad \forall k \in K, \forall n+i \in D_k \tag{15}$$

$$L_{(o_k)k} = 0 \quad \forall k \in K \tag{16}$$

$$x_{ijk} \in \{0,1\} \quad \forall k \in K, \forall (i,j) \in A_k \tag{17}$$

$$M_{max} \geq Y_k \geq 1 \quad \forall k \in K \tag{18}$$

$$Y_k \text{ integer} \quad \forall k \in K \tag{19}$$

The objective function aims to minimize the total travel cost, earliness, and tardiness penalties. Constraint (1) ensures that each pickup node is visited exactly once. Constraint (2) ensures that once an AGV visits a pickup node, the same AGV must visit the corresponding delivery node exactly once. Constraint (3) guarantees that each AGV starts from its origin depot. Constraint (4) ensures the flow conservation for each AGV. Constraint (5) ensures that each AGV finishes its route at its destination depot. Constraint (6) and Constraint (7) define the earliness of delivery node i as the nonnegative difference between the defined earliest delivery time and the actual arrival time at delivery node i, whereas Constraint (8) and Constraint (9) define the tardiness of delivery node i as the nonnegative difference between the actual arrival time and the defined latest delivery time. Constraint (10) ensures the time feasibility of the routes, considering the service time, travel time, and trolley impact time (TIM) if there is an attachment of trolleys. In other words, this constraint links the service starting time variables at an AGV's route. An AGV visits a pickup node before its corresponding delivery node by Constraint (11). If an AGV travels from node i to j, then that AGV needs to satisfy the load balance between node i and j which is enforced by Constraints (12) and (13). The upper bound and the lower bound on the load capacity of an AGV at a node depending on the type of the node (pickup or delivery node) are defined by Constraints (14) and (15). Constraint (16) sets the initial load of each AGV as zero at its origin depot. Constraint (17) defines x_{ijk} as binary variables. Lastly, Constraint (18) limits the number of trolleys attached to each AGV, ensuring that each AGV has at least one trolley and does not exceed the maximum allowable number of trolleys, and Constraint (19) defines the number of trolley attachments at each AGV as integer variables.

CHAPTER IV

SOLUTION APPROACH

The AGV routing and load handling problem, as introduced in Section 3, becomes increasingly complex when considering trolleys. While the proposed MILP model offers a foundational approach, it encounters significant computational challenges when applied to larger problem instances. To address these limitations, we have developed an ALNS algorithm, detailed in Section 4.1. Additionally, we have enhanced the sorting-based solution currently employed at Vestel Electronics by integrating more sophisticated sorting rules.

4.1 ALNS Algorithm

The ALNS algorithm represents a cutting-edge methodology for tackling complex optimization problems, leveraging the flexibility and adaptability of large neighborhood search techniques. By extending the traditional Large Neighborhood Search (LNS) framework, ALNS dynamically selects from a set of destruction and repair operators, iteratively refining solutions to converge towards optimal outcomes [32, 27, 22, 33].

ALNS algorithm represented in Algorithm 1 where a *request* is defined by a pickup node and its corresponding delivery node. Our algorithm begins with an initial random solution assigning all requests to AGVs while satisfying all feasibility checks, such as the capacity of each AGV. Hence, all requests are placed in the set of served requests (*servedRequest*). We also initialize the best solution (i.e., *BestSolution*) according to this random initial solution, minimum and maximum deviation percent from the current best solution, and the number of AGVs. Throughout the iterations, destruction and repair operators are randomly selected and applied to the current solution. If the new solution, $BestSolution_{new}$, proves to be better than the previously

recorded best solution, $BestSolution_{best}$, it replaces the latter, ensuring continuous improvement.

Algorithm 1 The General Framework of ALNS Algorithm

`Initialize` $max\_iteration$, $BestSolution_{best} = \{\}$, $BestRoutes = \{\}$, weights of destroy operators, weights of repair operators

Run `Random Insertion Algorithm`: compute an initial assignment for the AGVs
Update $BestSolution_{best}$ and $BestRoutes_{best}$

while $iteration \leq max\_iteration$ **do**
    Randomly choose a destroy operator
    Randomly choose a repair operator
    Apply the selected destroy operator
    Apply the selected repair operator
    Compute total cost after then update $BestSolution_{new}$ and $BestRoutes_{new}$
    if $BestSolution_{new} < BestSolution_{best}$ **then**
        $BestSolution_{best} := BestSolution_{new}$
        $BestRoutes_{best} := BestRoutes_{new}$
    end
    Update the randomness of destroy and repair operators
    $iteration := iteration + 1$
end

During the ALNS algorithm, we update the randomness of the destroy and repair operators' selection. For example, if the chosen destroy operator improves the current best solution at that iteration, its probability of reselection at future iterations increases, whereas, if the chosen destroy operator does not improve the current best solution at that iteration, its probability of reselection at future iterations decreases. In other words, the randomness of destroy and repair operators are adaptive based on their contribution to the objective function value.

4.1.1 Destruction Operators

One of the key innovations in our ALNS algorithm is the incorporation of a diverse set of seven destruction operators, each tailored to address specific aspects of the problem. The following sections detail the functionality of these operators, including Random Request Removal and Worst Cost Destroy, both of which are foundational

techniques in the literature.

4.1.1.1 Random Request Destroy

This operator, outlined in Algorithm 2, randomly selects and removes requests from the set of served requests, thereby introducing diversity by disrupting the current solution. This approach is inspired by similar techniques used in the scheduling literature [34, 35].

Algorithm 2 Random Request Destroy Algorithm

```
Input: neighborhood size, servedRequestSet, notServedRequestSet
 for i ← 1 to neighborhood size do
    if servedRequestSet is empty then
       break
    end
     Randomly choose a request in servedRequestSet
     Delete it from servedRequestSet
     Add it to notServedRequestSet
end
```

4.1.1.2 Worst Cost Destroy

This operator, presented in the Algorithm 3, targets and removes requests with the highest associated cost calculated according to distance and early/late arrival penalties. There are separate destroy algorithms in the literature, such as distance destroy, tardiness destroy [22, 34]. We establish a destroy mechanism by considering all costs according to our objective.

Algorithm 3 Worst Cost Destroy Algorithm

Input: neighborhood size, $BestRoutes_{best}$, TIM, $servedRequestSet$, $notServedRequestSet$

```
for i ← 1 to neighborhood size do
    if servedRequestSet is empty then
        break
    end
    for k in BestRoutes_best do
        for j ← 1 to number of nodes in route k do
            previous_node := The previous node relative to the current node j
            current_node := The current node in the route sequence
            Cost_j^distance := The distance cost calculated between previous_node and
            current_node
            current_time := The current time computed by considering the distance
            cost, service time, TIM and trolley_count
            Initialize Cost_j^tw_penalty := 0
            if current_node is a delivery node then
                if current_time is earlier than the start time window of current_node
                then
                    Cost_j^tw_penalty ← calculate and add the earliness penalty
                end
                if current_time is later than the end time window of current_node
                then
                    Cost_j^tw_penalty ← calculate and add the tardiness penalty
                end
            end
            Add the sum of the Cost_j^distance and Cost_j^tw_penalty to the Cost_request list
        end
    end
    Sort Cost_request in descending order
    Select the request with the highest cost (first element in Cost_request)
    Remove this request from servedRequestSet
    Add this request to notServedRequestSet
end
```

4.1.1.3 Worst Time Destroy

This operator, presented in Algorithm 4, calculates time deviations of the workstations where the time deviation is defined as the difference between the service starting time and the earliest time window for each delivery node (i.e., workstation). The algorithm ranks these time deviations in descending order and selects the workstation with the largest time deviation. This process continues until a predefined percentage of requests are selected

Algorithm 4 Worst Time Destroy Algorithm

Input: neighborhood size, $BestRoutes_{best}$, $servedRequestSet$, $notServedRequestSet$

```
Initialize an empty list to service_time_difference
for i ← 1 to neighborhood size do
    if servedRequestSet is empty then
        break
    end
    for k in BestRoutes_best do
        for j ← 1 to number of nodes in route k do
            Calculate the time difference between the start of the time window and
            the current time for node j
            Add the calculated difference to service_time_difference list
        end
    end
    Sort service_time_differences in descending order
    Select the request corresponding to the highest time difference (first element)
    Remove this request from servedRequestSet
    Add this request to notServedRequestSet
end
```

4.1.1.4 Time Based Request Destroy

This operator, presented in Algorithm 5, is chosen by the destroy operator at the previous iteration of the ALNS algorithm is set as the base request and its assigned AGV before applying the destroy operator is set as the base AGV. The request assigned to an AGV other than the base AGV that has the closest earliest delivery time to the base request's earliest delivery time is removed from its current route.

Algorithm 5 Time Based Request Destroy Algorithm

Input: neighborhood size, $BestRoutes_{best}$, $servedRequestSet$, $notServedRequestSet$, Large Number

```
last_time_based_location := A randomly chosen request from servedRequestSet
for i ← 1 to neighborhood size do
    if servedRequestSet is empty then
        break
    end
    smallest_diff := Large Number
    chosen_location := Empty List
    last_chosen_location := last_time_based_location
    for k in BestRoutes_best do
        for j ← 1 to number of nodes in route k do
            tw_diff := The difference calculated between the start time windows of
            last_chosen_location and node j
            if tw_diff is smaller than smallest_diff then
                Update chosen_location with node j
                Update smallest_diff with tw_diff
            end
        end
    end
    Update last_time_based_location to chosen_location
    Delete chosen_location from servedRequestSet
    Add chosen_location to notServedRequestSet
end
```

4.1.1.5 Demand Based Request Destroy

This operator, presented in Algorithm 6 is chosen by the destroy operator at the previous iteration of the ALNS algorithm is set as the base request, and its assigned AGV before applying the destroy operator is set as the base AGV. The request assigned to an AGV other than the base AGV that has the closest load value to the base request's load is removed from its current route. This operator aims to increase the probability of removing loads of requests fairly.

Algorithm 6 Demand Based Request Destroy Algorithm

Input: neighborhood size, $BestRoutes_{best}$, $servedRequestSet$, $notServedRequestSet$, Large Number

```
last_demand_based_location := A randomly chosen request in servedRequestSet
for i ← 1 to neighborhood size do
    if servedRequestSet is empty then
        break
    end
    smallest_diff := Large Number
    chosen_location := Empty List
    last_chosen_location := last_demand_based_location
    for k in BestRoutes_best do
        for j ← 1 to number of nodes in route k do
            demand_diff := The demand difference calculated between the
            last_chosen_location and node j
            if demand_diff is smaller than smallest_diff then
                Update chosen_location with node j
                Update smallest_diff with demand_diff
            end
        end
    end
    Update last_demand_based_location to chosen_location
    Delete chosen_location from servedRequestSet
    Add chosen_location to notServedRequestSet
end
```

4.1.1.6 Random Route Destroy

This operator, presented in Algorithm 7, aims to increase the variety of solutions by removing nodes from randomly selected routes. The algorithm randomly chooses a

route (in other words, an AGV) and then removes the request with the highest total cost at that route. This process continues until a predefined percentage of requests are selected.

Algorithm 7 Random Route Destroy Algorithm

Input: neighborhood size, TIM, $BestRoutes_{best}$, $servedRequestSet$, $notServedRequestSet$

for $i \leftarrow 1$ **to** *neighborhood size* **do**
  if *servedRequestSet is empty* **then**
    break
  end
  random_route := A randomly selected route from $BestRoutes_{best}$ that contains at least one request
  current_time := 0
  trolley_count := The number of trolleys in *random_route*
  for $j \leftarrow 1$ **to** *number of nodes in random_route* **do**
    previous_node := The previous node relative to the current node j
    current_node := The current node in the route sequence
    $Cost_j^{distance}$:= The distance calculated between *previous_node* and *current_node*
    current_time := The current time computed by considering the distance cost, service time, TIM and *trolley_count*
    if *current_node is a delivery node* **then**
      if *current_time is earlier than the start time window of current_node* **then**
        $Cost_j^{tw\_penalty}$:= earliness penalty
      end
      if *current_time is later than the end time window of current_node* **then**
        $Cost_j^{tw\_penalty}$:= tardiness penalty
      end
    end
    Compute the total sum of $Cost_j^{distance}$ and $Cost_j^{tw\_penalty}$
    Update the $Cost_{request}$ list
  end
  Sort $Cost_{request}$ in descending order
  Select the request with the highest cost
  Remove this request from *servedRequestSet*
  Add this request to *notServedRequestSet*
end

4.1.1.7 Worst Neighborhood Destroy

This operator, presented in Algorithm 8, calculates the total cost due to traveling and determines the ratio of each request's contribution to this traveling cost (i.e., earliness and tardiness costs are ignored). Then, these ratios are sorted in descending order, and the request with the largest ratio is chosen by this destroy operator. This process continues until a predefined percentage of requests are selected.

Algorithm 8 Worst Neighborhood Destroy Algorithm

Input: neighborhood size, $BestRoutes_{best}$, $servedRequestSet$, $notServedRequestSet$
for $i \leftarrow 1$ **to** *neighborhood size* **do**
  if *servedRequestSet is empty* **then**
    break
  end
  Initialize empty lists named *destroy_list* and *distance_list*
  for k *in* $BestRoutes_{best}$ **do**
    $total\_distance := 0$
    for $j \leftarrow 1$ **to** *number of nodes in a route* k **do**
      $previous\_node$:= The previous node relative to the current node j
      $current\_node$:= The current node in the route sequence
      $Cost_j^{distance}$:= The distance between $previous\_node$ and $current\_node$
      Append $Cost_j^{distance}$ to $distance\_list$
      $total\_distance := total\_distance + Cost_j^{distance}$
    end
  end
  foreach d *in* $distance\_list$ **do**
    Append $destroy\_list$ to (d / $total\_distance$)
  end
  Sort $Cost_{request}$ in descending order
  Select the request with the highest cost
  Remove this request from $servedRequestSet$
  Add this request to $notServedRequestSet$
end

We define *unserved request* set as the set of requests chosen by the destroy operators, sorted with respect to the difference that they cause due to their removal from their current route. When a request is chosen to be included in the route of an AGV, the following feasibility conditions overall AGVs must be always satisfied: (i)

the origin and the final destination nodes of an AGV's route have to be depot, (ii) overloading an AGV should not be permitted, (iii) the delivery nodes of every pickup node have to be included in the same AGV's route, and (iv) all pickup nodes have to be visited once. Additionally, the order at the AGV's route should be updated considering the newly inserted request (i.e., its pickup and delivery nodes). Since the delivery node is not necessarily visited just right after its corresponding pickup node, we evaluate both the pickup and delivery nodes' orders at the route. The common rule applied during insertion is that a delivery node cannot be visited before its corresponding pickup node.

Once the destruction phase is completed and unserved requests are identified, the next step involves the use of repair operators to reinsert these requests into the AGV routes.

4.1.2 Repair Operators

In the context of our AGV routing and load handling problem, repair operators play a crucial role in reintegrating requests into the AGV routes after they have been removed by the destroy operators. Two key repair operators, Greedy Insertion and Random Insertion, are employed for this task, each designed to optimize the assignment and scheduling of requests while considering the total cost, including traveling, earliness, and tardiness. Both of these repair operators rely on a fundamental insertion procedure, termed the Insert Request Algorithm, which is utilized to assess and determine the optimal placement of each request within the available routes.

4.1.2.1 Insert Request Algorithm

The Insert Request, presented in the Algorithm 9, is integral to both repair strategies. It evaluates each node within an AGV's current route to identify the most suitable insertion points for a new request's pickup and delivery nodes. This evaluation involves simulating the insertion of the request into different points along the route

and calculating the associated costs, which include distance traveled and penalties for early or late deliveries relative to the time windows. The insertion point that results in the lowest total cost is selected as the optimal position for the request. If the calculated cost exceeds a predefined acceptable threshold, the algorithm dynamically adjusts this threshold and reruns the insertion process to ensure cost-effectiveness and feasibility.

Algorithm 9 Insert Request Algorithm

Input: Large Number, $convenientMaxInterval$, $convenientMinInterval$, $ConvenientIntervalFactor$, $Large\ Number$, Local Parameter: $request$, k

```
chosenAGV := None
chosenMinCost := Large Number
for i ← 1 to number of nodes in a route k do
    for j ← i + 1 to each nodes in a route k do
        Simulate the insertion of a receiving and delivering node of request at the
        node range i and j in the list of requests in route k
        if Check feasibility of updated route k then
            Cost_j^distance := the distance cost after inserting pickup location
            Cost_j^distance* := the distance cost after inserting delivery location
            Cost_j^tw_penalty := the early/late penalty cost after inserting pickup and
            delivery location
            Cost_j^total ← Cost_j^distance + Cost_j^distance* + Cost_j^tw_penalty
            if Cost_j^total < minCost then
                chosenAGV := k
                chosenMinCost := Cost_j^total
            end
        end
    end
end
if chosenAGV is None then
    if chosenMinCost > convenientMaxInterval then
        Increase convenientMaxInterval value by the ConvenientIntervalFactor
        chosenAGV := None
        chosenMinCost := Large Number
    end
    else
        Decrease convenientMaxInterval value by the ConvenientIntervalFactor
        if convenientMaxInterval value is less than the convenientMinInterval
        value then
            convenientMaxInterval := convenientMinInterval
        end
    end
end
return chosenAGV, chosenMinCost
```

4.1.2.2 Greedy Insertion

This operator, presented in the Algorithm 10, chooses the requests starting from the first element in the *unserved request* set and inserts each request into any of the routes (including possible new ones if idle AGVs are present) by selecting the best insertion point (i.e., the nodes at an AGV's route) resulting in the minimum cost by applying a greedy search, hence ensuring cost minimization [36]. This insertion is performed until all requests are chosen in order from the *unserved request* set, and are assigned to routes.

Algorithm 10 Greedy Insertion Algorithm

Data: $notServedRequestSet$, $Large\ Number$, $CountOfAGVs$, $BestRoutes_{best}$, $BestSolution_{best}$

```
while notServedRequestSet is not empty do
    foreach request in notServedRequestSet do
        chosenAGV := None
        chosenMinCost := Large Number
        for k in BestRoutes_best do
            chosenAGV, chosenMinCost := The AGV and minimum cost determined by the Insert Request Algorithm for route k with the request parameter
            if chosenMinCost < BestSolution_best then
                Delete k from BestRoutes_best
                Assign chosenAGV to BestRoutes_best
            end
            else
                if size of BestRoutes_best < CountOfAGVs then
                    Create a new route that starts and ends at the depot and includes the request, then assign this new route to the BestRoutes_best list
                end
                else
                    Try inserting request to all routes, even if it increases the cost, and insert it to the route with the lowest cost.
                end
            end
        end
        Add request to servedRequestSet
        Delete request from notServedRequestSet
    end
end
```

4.1.2.3 Random Insertion

This operator, presented in the Algorithm 11, takes a less deterministic approach. It selects requests randomly from the unserved request set and attempts to insert them into randomly chosen routes. If the insertion violates any feasibility constraints, such as exceeding capacity or breaching time windows, the algorithm checks for idle AGVs that could accommodate the request. If no such AGVs are available, another route is chosen at random, and the insertion process is attempted again. This process is repeated until all requests from the *unserved request* set are successfully assigned to routes.

Algorithm 11 Random Insertion Algorithm

Data: $notServedRequestSet$, $Large\ Number$, $CountOfAGVs$, $BestRoutes_{best}$, $BestSolution_{best}$

```
while notServedRequestSet is not empty do
    chosenAGV := None
    chosenMinCost := Large Number
    Select a random request from the notServedRequestSet list and assign it to the
    request variable
    while BestRoutes_best is not empty do
        Select a random route from the BestRoutes_best list and assign it to the k from
        the Insert Request Algorithm
        chosenAGV, chosenMinCost := The AGV and minimum cost determined
        by the Insert Request Algorithm for route k with the request parameter
        if chosenMinCost < BestSolution_best then
            Delete k from BestRoutes_best
            Assign chosenAGV to BestRoutes_best
        end
        else
            if size of BestRoutes_best < CountOfAGVs then
                Create a new route that starts and ends at the depot and includes the
                request, then assign this new route to the BestRoutes_best list
            end
            else
                Try inserting request to all routes, even if it increases the cost, and
                insert it to the route with the lowest cost.
            end
        end
        Add request to servedRequestSet
        Delete request from notServedRequestSet
    end
end
```

The flexibility and adaptability of these insertion mechanisms enable the repair operators to handle a wide range of scenarios, ensuring that the AGV routing problem can be effectively managed even after significant disruptions caused by the destroy operators. These methods, when combined with the adaptive capabilities of the ALNS framework, provide a robust solution approach capable of delivering near-optimal results in complex and dynamic environments.

Note that the number of trolleys may be updated during repair operators. For example, inserting a request may require an AGV to be attached to an additional trolley, hence, its total cost needs to be recomputed at every stage due to trolley impact times. Thus, we recalculate the total cost during repair operators. Therefore, the impact of the number of trolleys and the route on the AGV is carefully assessed during each insertion process.

Our ALNS algorithm not only builds upon the strengths of conventional ALNS but also is tailored to capture our problem's essence. The combination of our destroy and repair operators, along with adaptive interval adjustments, enhances the ALNS algorithm's flexibility and robustness. This allows our algorithm to escape local optima and converge towards a global optimum solution for our AGV routing and load handling problem.

4.2 Current Solution Approach at Vestel Electronics

At Vestel Electronics, the current solution approach prioritizes time efficiency by relying on sorting mechanisms based on cost and the worst distance. While effective, this method can be further enhanced. We propose an improved solution approach, termed the Rule-of-Thumb (RoT) approach, which incorporates additional sorting rules. The RoT approach continues to leverage quick sorting techniques while yielding better objective function values for some instances compared to the current solution approach.

4.2.1 RoT

The RoT approach introduces four distinct sorting mechanisms: *cost*, *greedy selection*, *worst distance*, and *random tardiness.* Each of these methods is applied independently to solve the AGV routing and load handling problems. After applying all four sorting methods, the RoT approach selects the best result among them. This ensures that the solution is both time-efficient and optimized for the given objective function values. Each sorting approach is detailed as follows:

4.2.1.1 Cost

In this sorting method, each request's cost is calculated based on travel expenses, earliness, and tardiness penalties, assuming that an AGV serves only that request. The AGV starts from the depot, visits the pickup node, proceeds to the corresponding delivery node, and finally returns to the depot. Requests are then prioritized and sorted in ascending order of their computed costs, and assigned to AGVs sequentially. This method utilizes all available AGVs when the number of requests exceeds the number of AGVs.

4.2.1.2 Greedy Selection

This sorting method chooses requests based on the sequence of orders as they are entered into the system. The selected request is then inserted into any of the routes (including possible new ones if idle AGVs are present) by identifying the best insertion point (i.e., the locations within an AGV's route) that results in the minimum total cost. The total cost is minimized by applying a greedy search, ensuring that the solution is as cost-effective as possible. We introduce this sorting rule, motivated by a potential cutoff for the number of requests entered into the system before a possible deadline.

4.2.1.3 Worst Distance

Similar to sorting with respect to *cost*, each request's cost is computed assuming that an AGV only serves that request, but regarding only the traveling cost. Then, the request with the smallest cost will be chosen and assigned to an AGV. This sequential assignment of requests to AGVs will be utilized till all requests are assigned. Note that this sorting utilizes all AGVs if the number of requests is larger than the number of AGVs.

4.2.1.4 Random Tardiness

We first assign all requests randomly to all AGVs. Then, we compute the tardiness of each request based on these random routes and sort the tardiness values in descending order. Afterwards, we assign each request from the sorted request list to AGVs sequentially.

CHAPTER V

COMPUTATIONAL RESULTS

We evaluate the computational performance of the proposed ALNS algorithm in terms of the objective function values and CPU times via a comprehensive numerical study using two types of distinct data sets. We first introduce 1095 hypothetical instances motivated from real life to evaluate the ALNS algorithm in Section 5.1. Then, we present a real case study of Vestel Electronics' refrigerator factory with 50 pickup nodes and 41 delivery nodes with a total of 53 requests in Section 5.2.

All solution approaches are implemented using the Python programming language, and Google OR-Tools is utilized for the MILP. The MILP, the ALNS algorithm, and the RoT are solved for all hypothetical instances, whereas only the ALNS algorithm and the RoT are solved for the Vestel case study as the MILP cannot solve this case with 24 hours computation time limit. All computational experiments were performed on a computer system with 32 GB of RAM, running Windows 11, and equipped with a 10th Generation Intel(R) Core(TM) i7-10750H @ 2.60GHz processor (12 CPUs).

5.1 Hypothetical instances

In order to evaluate the quality of the proposed ALNS algorithm, we first generated 1095 instances categorized into three distinct groups: small, medium, and large instances. Small instances contain 11 service points, while medium and large instances consist of 15 and 19 service points, respectively. When the number of service points exceeds 19, computation times for the MILP to compute optimal or feasible solutions increase significantly. Therefore, we created these hypothetical instances to thoroughly assess the performance of the MILP model. The parameters for these instances include the number of AGVs (selected from the set $1, 2, 3$), trolley impact time

(from the set $20, 30, 40$), the maximum number of trolleys that can be attached to each AGV (from the set $1, 2, 3$), the number of service points (from the set $11, 15, 19$, corresponding to 5, 7, and 9 delivery requests, respectively, plus one depot point), and the earliness/tardiness penalty values (from the set $10, 20, 30$).

Detailed examples of hypothetical datasets with 11, 15, and 19 service points are provided in the Appendix. These datasets illustrate the different scenarios and configurations used in our study, allowing readers to explore the data structures in more detail and replicate the experiments. By modifying parameters such as demand, location, service times, and time windows, we generated additional variations based on these initial datasets. Furthermore, by adjusting the AGV set, trolley impact time, trolley set, and earliness/tardiness values, a total of 1215 dataset samples were created and tested. Some datasets were excluded due to infeasibility (e.g., a demand exceeding the trolley capacity of 60 under certain constraints). Consequently, the final set of tests was conducted on 1095 feasible datasets.

We first compare the performance of the MILP model, and the ALNS algorithm with respect to computation time as in Figure 5. The computational time limit is set to 20 minutes (1,200 seconds) for the hypothetical instances. The x-axis represents the ID of the hypothetical instances (smaller IDs meaning data sets with a smaller number of service points), whereas the y-axis shows the computation time in seconds. Since RoT's computation time is very short (less than 0.02 seconds on average), its computation time is excluded from Figure 5. One-third of the instances belong to the small instance set, another third to the medium instance set, and the remaining third to the large instance set. It is obvious from Figure 5 that the MILP model cannot compute the optimal solutions even for some instances from the small instance set and cannot solve most of the instances from medium and large instance sets. ALNS algorithm results in a more robust solution time even when the instances get larger and solves all instance sets in less than 5 minutes.

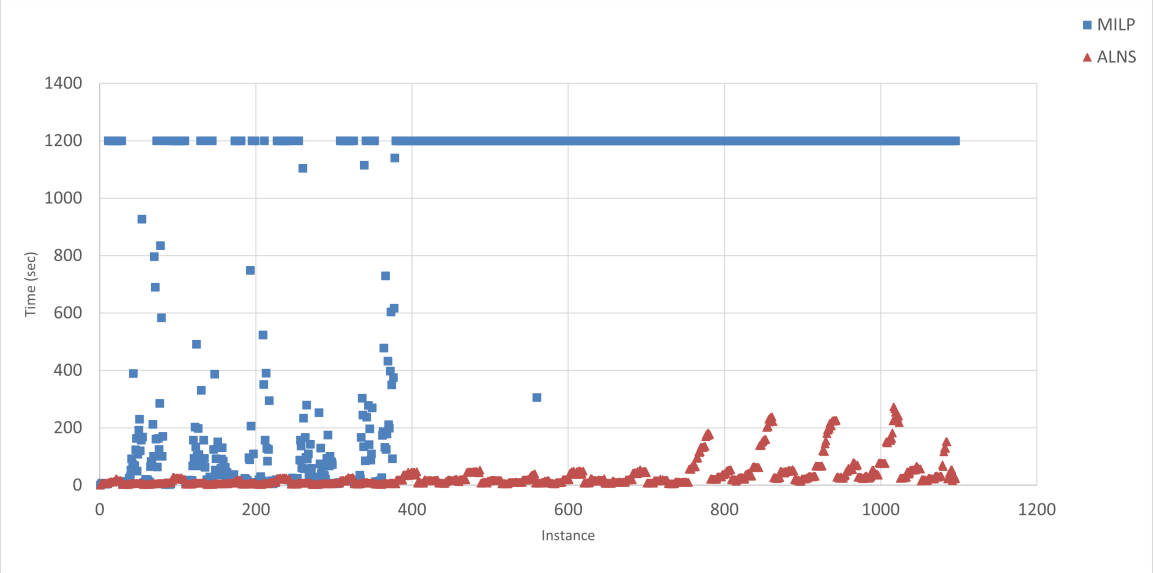

Figure 5: Comparison of CPU time: the MILP and the ALNS algorithm

Table 2 shows average and standard deviation of computation times. We break down the computation times of MILP into three groups as follows.

(i) OPT: instances whose optimal solution can be computed by the MILP within the computation time limit

(ii) FEAS: instances where the MILP cannot compute the optimal solution but can find at least one feasible solution within the computation time limit

(iii) NFEAS: instances whose first initial feasible solution cannot be computed by the MILP within the computation time limit

For the ALNS algorithm and the RoT, we use NOPT for referring to the combined group of FEAS and NFEAS. Table 2 shows that the ALNS algorithm is significantly faster than solving the MILP, and the RoT approach is the fastest approach.

Table 2: Statistics of computation times for the MILP, the ALNS algorithm, and the RoT

Approach	Average Computation Time (sec)	Standard Deviation	# of Instances
MILP_{OPT}	110.6	184.7	280
$\text{MILP}_{\text{FEAS}}$	1200.0	0.0	709
$\text{MILP}_{\text{NFEAS}}$	1200.0	0.0	106
ALNS	29.8	45.1	989
$\text{ALNS}_{\text{NOPT}}$	43.3	37.6	106
RoT	0.01	0.02	989
RoT_{NOPT}	0.02	0.02	106

While evaluating the quality of the solutions, we define the base method as the RoT since the MILP could not find a feasible solution for 106 instances out of 1095 instances. We define gap as $(Obj_{RoT}\text{-}Obj_x)/Obj_{RoT}$ where x represents the MILP or the ALNS algorithm. Table 3 presents statistics of gap with respect to varying numbers of service points (i.e., the pickup and delivery nodes and AGV starting depot). We consider the instances at which the MILP can compute at least one feasible solution (in other words, FEAS and OPT) and break down these instances such that the ones whose MILP objective function values are either equal to or better than the ones computed via the RoT and the instances at which MILP results in worse objective function values than the RoT's, as MILP+ (i.e., resulting in nonnegative gap values) and MILP- (i.e., resulting in negative gap values), respectively. Additionally, ALNS gap performance is reported in two categories: NFEAS and the rest of the instances.

Table 3: Statistics of gap with respect to varying number of service points

# of Service Points		MILP-	MILP+	ALNS	$\text{ALNS}_{\text{NFEAS}}$
11	# of Instances	26	352	378	0
	Average Gap	-1.90	0.07	0.06	0
	Standard Deviation	5.80	0.10	0.10	0
15	# of Instances	255	120	375	0
	Average Gap	-2.0	0.12	0.10	0
	Standard Deviation	4.20	0.20	0.10	0
19	# of Instances	198	38	236	106
	Average Gap	-34.8	0.12	0.13	0.08
	Standard Deviation	79.6	0.20	0.10	0.09

The performance of MILP- degrades significantly as the number of service points increases as shown in Table 3. In other words, as the instances get larger, the MILP computes feasible solutions much worse than the computed solution via RoT. Additionally, the standard deviation of the gap increases to 79.6 from 5.8. Conversely, in 510 instances of MILP+, both the average gap and its standard deviation show a slight upward trend; the average gap rises from 7% to 12%, while the standard deviation increases from 10% to 20%. However, a challenge with MILP+ is that as the number of service points increases, the number of instances where MILP+ finds a better solution than RoT within the CPU time limit decreases. This is due to the fact that larger instances often fall into the MILP- category. Table 3 shows that the ALNS algorithm results between 6% and 13% gap on average. Note that, $\text{ALNS}_{\text{NFEAS}}$ has an average gap of 8% with a standard deviation of 9% for the instances that the MILP cannot compute a feasible solution within the time limit. Consequently, the performance of MILP+ and ALNS algorithm appears comparable in less complex instances (510 MILP+ instances); however, in the remaining 585 instances, the ALNS algorithm significantly outperforms within the computation time limit.

We next examine the effect of the maximum number of trolley attachments, as shown in Table 4. For MILP-, the performance significantly degrades as the maximum

number of trolleys that can be attached to each AGV increases. This degradation is reflected in both the average gap and the standard deviation, which show substantial growth. As the problem complexity increases, MILP- struggles to maintain its performance. For MILP+ instances, there is a slight increase in both the average gap (from 7% to 10%) and the standard deviation (from 11% to 15%) as the maximum number of trolleys for each AGV increases. This indicates that while MILP+ maintains a relatively stable performance, there is a small rise in variability with more complex instances. Additionally, as the maximum number of attached trolleys increases, the total number of MILP+ instances decreases. This reduction in effectiveness results in larger and more complex instances often being classified under the MILP- category. Regarding the ALNS algorithm, the results show that there is no significant change in the gap as the maximum number of trolleys increases. This gap stability suggests that ALNS is more effective in handling larger and more complex instances compared to the MILP. Additionally, the comparison between MILP- and MILP+ indicates that while MILP- tends to struggle with larger instances, MILP+ shows a more consistent performance, albeit with a slight increase in variability. This highlights the importance of selecting the appropriate methodology based on the instance complexity. The increasing gap for MILP- suggests that as the problem complexity increases, the ability to compute optimal solutions may diminish, making alternative approaches like the ALNS algorithm more viable for larger instances.

Table 4: Statistics of the gap with respect to varying maximum numbers of trolley attachments for each AGV

Max # of Trolley Attachments		MILP-	MILP+	ALNS	$ALNS_{NFEAS}$
1	# of Instances	61	195	256	68
	Average Gap	-1.68	0.07	0.08	0.05
	Standard Deviation	5.41	0.11	0.11	0.09
2	# of Instances	185	158	343	35
	Average Gap	-10.86	0.07	0.09	0.14
	Standard Deviation	35.16	0.10	0.11	0.07
3	# of Instances	233	157	390	3
	Average Gap	-22.96	0.10	0.11	0.09
	Standard Deviation	69.53	0.15	0.14	0

We also study the effect of trolley impact time, which is the fixed amount of time added for each trolley attachment on an AGV during every move between nodes, in Table 5. As the trolley impact time becomes longer, MILP- shows a significant increase in the average and standard deviations of gap, indicating more variable and poorer performance. In contrast, the ALNS algorithm (including $ALNS_{NFEAS}$) consistently delivers low and stable gap values that are between 6% and 12%, proving its reliability. Thus, for systems with high impact time values, the ALNS method is preferred due to its consistent performance.

Table 5: Statistics of gap with respect to varying trolley impact times

Trolley Impact Time		MILP-	MILP+	ALNS	$ALNS_{NFEAS}$
20	# of Instances	130	199	329	39
	Average Gap	-13.08	0.12	0.12	0.10
	Standard Deviation	40.64	0.15	0.14	0.09
30	# of Instances	171	163	334	32
	Average Gap	-13.82	0.07	0.09	0.06
	Standard Deviation	60.69	0.11	0.12	0.07
40	# of Instances	178	148	326	35
	Average Gap	-19.08	0.05	0.07	0.09
	Standard Deviation	54.96	0.07	0.11	0.10

In addition to trolley impact time, we also analyze the effect of penalty values of

tardiness and earliness at the objective function in Table 6. We observe that the performance of the MILP- consistently degrades compared to the RoT with increasing standard deviation as the penalty value increases. Specifically, the highest deviations observed range from approximately 159 times to 745 times. This significant variance underscores the inefficiency of the MILP model compared to the RoT approach under increasing penalty conditions; even the improving performance of MILP+ with increasing penalty coefficients is not enough to motivate utilizing the MILP. Since the MILP exhibits inferior performance, with higher average gaps and standard deviations across all penalty values, the ALNS algorithm should be chosen as a solution method for varying penalty values.

Table 6: Statistics of gap with respect to varying penalty values

Penalty Value		MILP-	MILP+	ALNS	$\text{ALNS}_{\text{NFEAS}}$
10	# of Instances	150	173	323	41
	Average Gap	-8.03	0.08	0.09	0.09
	Standard Deviation	22.85	0.13	0.11	0.09
20	# of Instances	164	174	338	28
	Average Gap	-13.60	0.08	0.10	0.07
	Standard Deviation	34.16	0.10	0.13	0.08
30	# of Instances	165	163	328	37
	Average Gap	-24.41	0.09	0.10	0.09
	Standard Deviation	81.42	0.13	0.14	0.09

Finally, the effect of the number of AGVs, represented in Table 7, shows that as the number of AGVs increases, MILP- performs worse, with the average gap and the standard deviation increasing significantly, whereas, the ALNS algorithm maintains a low and stable average gap, proving its reliability and robustness.

Table 7: Statistics of gap with respect to varying number of AGVs

# of AGVs		MILP-	MILP+	ALNS	$ALNS_{NFEAS}$
1	# of Instances	196	145	341	24
	Average Gap	-3	0.03	0.04	0.07
	Standard Deviation	15.19	0.08	0.07	0.07
2	# of Instances	131	189	320	46
	Average Gap	-21.57	0.13	0.15	0.1
	Standard Deviation	73.23	0.15	0.17	0.1
3	# of Instances	152	176	328	36
	Average Gap	-26.62	0.08	0.1	0.09
	Standard Deviation	61.97	0.09	0.1	0.08

Table 8 shows the breakdown of the objective function value as traveling, earliness, and tardiness costs, assuming each instance's computed objective function value corresponds to 100%. When the MILP can compute the optimal solution, the earliness cost contribution to the optimal objective function value is the smallest on average, whereas, the traveling cost contribution is the largest on average. However, in cases where the MILP can compute any feasible solution but not the optimal solution within the computation time limit, the tardiness cost exceeds the traveling cost on average. The ALNS algorithm and the RoT, on the other hand, generate routes that balance distance and tardiness costs on average. Overall, the earliness cost is the lowest one on average for 1095 instances. The management can use this breakdown analysis to make strategic and operational decisions, such as buying more AGVs, changing the layout of the depot and workstations, and negotiating with production planners while defining the latest time of the deliveries.

Table 8: Cost breakdown in terms of percentages

Approaches	**Travelling Cost**	**Earliness Cost**	**Tardiness Cost**
MILP OPT	63.1%	0.4%	36.5%
MILP FEAS	22.8%	0.3%	76.9%
ALNS	50.2%	0.5%	49.3%
RoT	48.8%	0.4%	50.8%

5.2 Case Study: Vestel Refrigerator Factory

The case study examines one of Vestel Electronics' refrigerator factories, illustrated in Figure 6, spanning approximately 68,000 square meters. The layout includes a single depot, 50 pickup points marked with green dots, and 41 delivery points marked with red dots. We consider the demand for a half-day period (08:00 AM - 12:00 PM) that includes 53 delivery requests. To maintain consistency with our notations, we duplicate some pickup and delivery points to generate 53 requests (i.e., some requests may share the same delivery point because their original delivery points correspond to the same workstation, but they will be assigned different IDs, as the pickup nodes are labeled from $1, .., n$ and the delivery nodes are labeled from $n + 1, .., 2n$ for n requests).

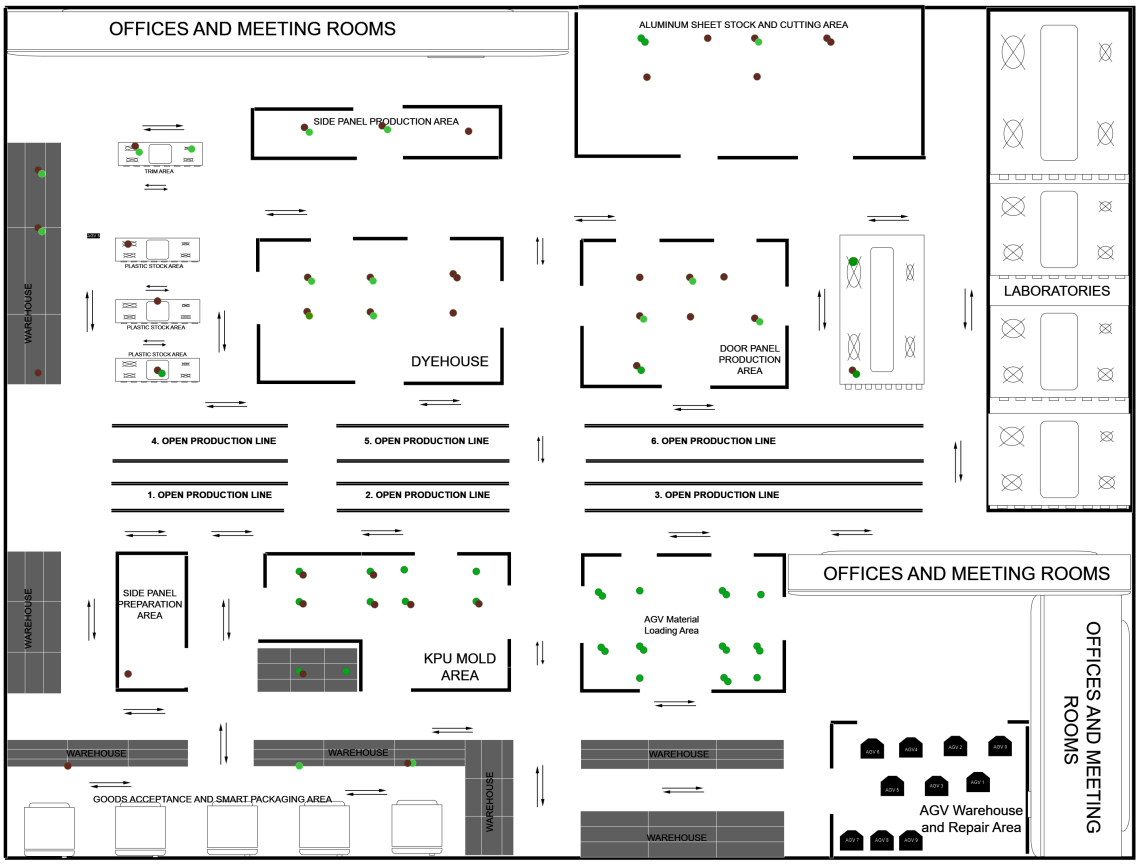

Figure 6: Layout of Vestel Electronics' Refrigerator Factory showing pickup and delivery nodes with green and red dots, respectively

The factory layout encompasses various sections essential to the production and logistics processes. The Side Panel Production Area specializes in manufacturing refrigerator side panels, while the Aluminum Sheet Stock and Cutting Area manages the storage and cutting of aluminum sheets used in production. The Dyehouse plays a critical role in painting and finishing components, whereas The Door Panel Production Area is dedicated to producing door panels. Additionally, the factory features six Open Production Lines utilized for assembling and producing various refrigerator components. The Side Panel Preparation Area prepares side panels for assembly, while the KPU Mold Area is involved in the molding process of specific parts. The Goods Acceptance and Smart Packaging Area is responsible for receiving goods and smart packaging for further processing. Multiple Warehouse areas are designated for

storing raw materials and finished products. The AGV Material Loading Area is set aside for loading materials onto AGVs, and the AGV Warehouse and Repair Area provides space for storing and repairing AGVs. The factory also houses Laboratories for testing and quality control, along with Offices and Meeting Rooms located at both the north and south ends of the facility for administrative and coordination purposes.

It is important to note that the AGV lanes within the factory are pre-determined. There are ten AGVs, and each can be equipped with up to three trolleys. For safety reasons, the use of four or more trolleys is not permitted. The trolley impact time is set to 30 seconds. The management sets the objective function coefficients for earliness and tardiness as 20, while the traveling cost's objective function coefficient is set as 1.

The MILP could not compute even a feasible solution within 24 hours. Hence, we only report the solutions of the ALNS algorithm and the RoT approach. The ALNS algorithm solves the case study in 1370.4 seconds, resulting in an objective function value of 5,942, whereas the RoT (i.e., the improved current solution method of Vestel Electronics) computed the objective function value as 286,044 in 192.4 seconds. The dominant component in this objective function value is due to tardiness, which constitutes 99.2% of the total cost. in 192.4 seconds. Even though the ALNS algorithm takes more time than the RoT, the use of additional time is worthwhile, as it significantly reduces the objective function value from 286,044 to 5,942.

Similarly, when we analyze the cost breakdown of the objective function value of 5,942, computed via the ALNS algorithm, we observe that the dominant component is again due to tardiness, with the distance cost being 1,902 and the tardiness penalty being 4,040. This indicates a significant opportunity for improvement by reducing tardiness. Management can address this issue by updating strategic and operational decisions, such as negotiating with production planners and schedulers about the latest delivery times or investing in additional AGVs.

The best routes computed by the ALNS algorithm, which yielded an objective function value of 5,942, are presented in Table 9. These routes reflect the algorithm's ability to effectively balance distance and tardiness costs, thereby optimizing the overall operation. On the other hand, the routes obtained using the RoT approach, which resulted in a substantially higher objective function value of 286,044, are shown in Table 10. This comparison highlights the efficiency of the ALNS algorithm in minimizing costs compared to the traditional RoT.

Table 9: Routes computed by ALNS algorithm for the Vestel Refrigerator Factory case study with objective function value of 5942

AGV ID	Route
0	D0 - C13 - C51 - C104 - C66 - C2 - C55 - C10 - C63 - C33 - C86 - D0
1	D0 - C48 - C53 - C106 - C35 - C88 - C20 - C73 - C101 - C43 - C96 - D0
2	D0 - C52 - C105 - C52 - C5 - C105 - C36 - C58 - C89 - C19 - C72 - D0
3	D0 - C4 - C23 - C57 - C76 - C39 - C92 - C6 - C59 - C25 - C78 - C8 - C61 - D0
4	D0 - C13 - C42 - C95 - C66 - C49 - C102 - C44 - C97 - C37 - C90 - D0
5	D0 - C7 - C60 - C4 - C57 - C49 - C102 - C32 - C85 - C24 - C77 - D0
6	D0 - C9 - C62 - C30 - C83 - C21 - C74 - C48 - C101 - C32 - C85 - C11 - C64 - C22 - C75 - D0
7	D0 - C12 - C17 - C47 - C100 - C70 - C28 - C81 - C50 - C103 - C65 - C29 - C82 - C16 - C69 - C1 - C54 - C40 - C93 - C46 - C99 - C18 - C71 - C41 - C94 - C27 - C80 - D0
8	D0 - C34 - C87 - C15 - C68 - C42 - C95 - C26 - C79 - C43 - C96 - C3 - C56 - D0
9	D0 - C45 - C14 - C67 - C98 - C38 - C91 - C22 - C75 - C31 - C84 - D0

We also note that the size of the neighborhood search employed by the destroy operators introduced in Section 4, determined by the predefined percentage of requests, is crucial for the performance of the ALNS algorithm. While solving the hypothetical instances, we use 100% of the total number of service nodes for the size of the

Table 10: Routes computed by RoT for the Vestel Refrigerator Factory case study with objective function value of 286044

AGV ID	**Route**
0	D0 - C7 - C60 - C51 - C104 - C2 - C55 - C27 - C41 - C94 - C80 - D0
1	D0 - C9 - C62 - C25 - C78 - C44 - C97 - C40 - C93 - C36 - C89 - C18 - C71 - C20 - C73 - D0
2	D0 - C14 - C39 - C92 - C67 - C13 - C66 - D0
3	D0 - C34 - C87 - C17 - C70 - C28 - C81 - C35 - C88 - C43 - C96 - C31 - C84 - D0
4	D0 - C15 - C45 - C68 - C98 - C38 - C91 - C16 - C69 - C33 - C86 - D0
5	D0 - C53 - C50 - C103 - C106 - C32 - C85 - C10 - C63 - C37 - C90 - C22 - C75 - D0
6	D0 - C4 - C57 - C49 - C102 - C6 - C59 - C8 - C61 - D0
7	D0 - C30 - C83 - C21 - C74 - C52 - C105 - C29 - C82 - D0
8	D0 - C23 - C76 - C48 - C101 - C3 - C12 - C56 - C26 - C79 - C65 - C11 - C64 - D0
9	D0 - C47 - C100 - C42 - C95 - C5 - C58 - C46 - C99 - C1 - C54 - C19 - C72 - C24 - C77 - D0

neighborhood search since the number of service nodes is less than 20. However, 30% is chosen for the real case study to avoid increasing the computation time while still resulting in a good ALNS algorithm performance.

Requests for pickups and deliveries in the Vestel refrigerator factory are received from various sections within the layout in Figure 6. These sections include the Side Panel Production Area, Aluminum Sheet Stock and Cutting Area, Dyehouse, Door Panel Production Area, and the Open Production Assembly. Additionally, requests originate from the Side Panel Preparation Area, KPU Mold Area, Goods Acceptance and Smart Packaging Area, and multiple Warehouse locations. These diverse request points highlight the comprehensive coverage of material flow within the factory, ensuring that different stages of production and storage are efficiently managed by the ALNS algorithm. In Figure 7, the layout displays the routes generated by the ALNS algorithm, with each AGV's route represented in a distinct color, clearly depicting the comprehensive network of pickups and deliveries. Additionally, the colors used to represent the discrete AGV routes overlap, indicating that the AGVs are making

complex movements in the warehouse.

Lastly, Table 11 displays the frequency of selected destroy and repair operators during the ALNS algorithm in the case study. Among the most commonly used destroy operators are the Worst Cost and Random Route, while the Greedy Insert operator is utilized more frequently than the random insert in this case study.

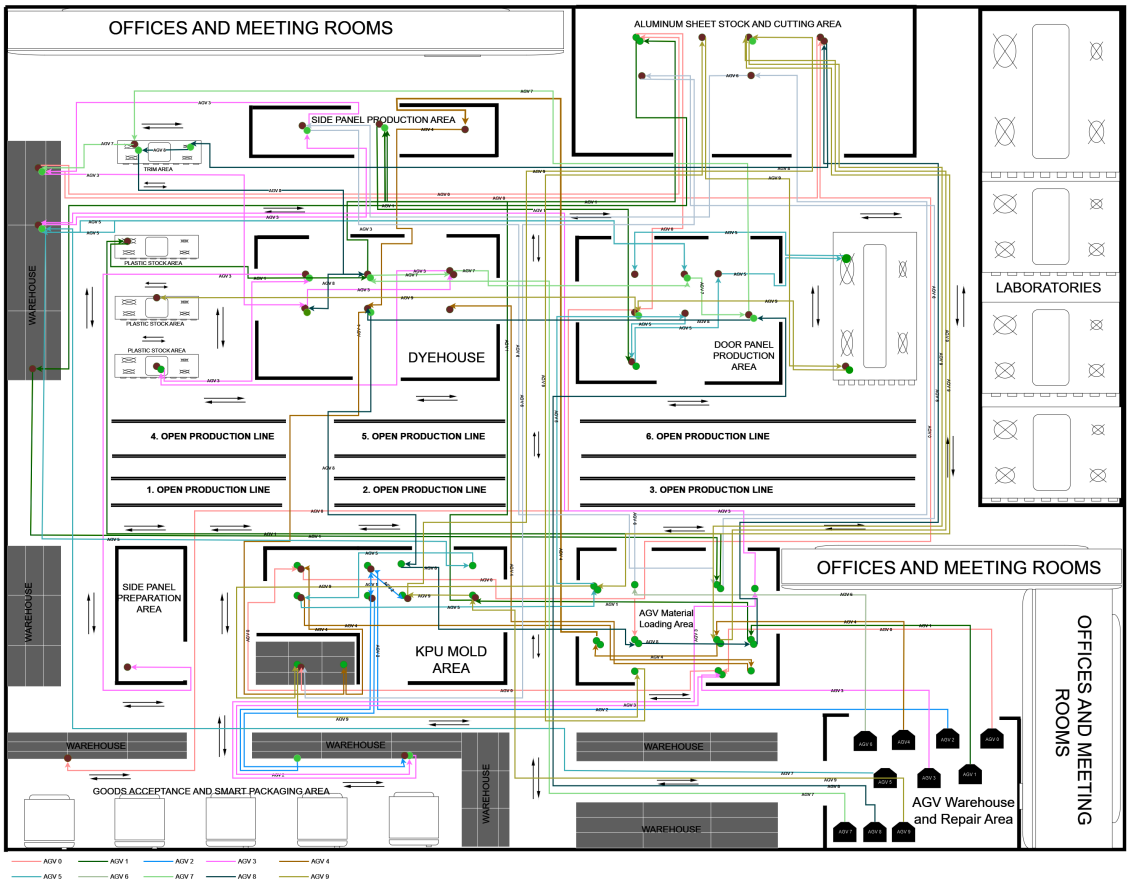

Figure 7: AGV routes computed by the ALNS algorithm for the Vestel Refrigerator Factory case study

Table 11: Frequency of selected destroy and repair operators during the ALNS algorithm in the Vestel Refrigerator Factory case study

Destroy Operator	**Repair Operator**	
	Greedy Insertion	**Random Insertion**
Random Request	3	1
Worst Cost	10	3
Worst Time	7	2
Demand Based Request	4	1
Worst Neighbourhood	1	0
Random Route	9	3
Time Based Request	8	4

The problem realized by Vestel Electronics is dynamic due to frequent changes in demand, production schedules, and potential disruptions. Therefore, the management requires a solution approach that is both robust and adaptable to these changes, while still maintaining high efficiency and reliability.

In summary, our numerical study demonstrates that the ALNS algorithm not only computes reliable and high-quality results in a short computation time but also offers the flexibility needed to handle dynamic situations. Given the dynamic nature of demands in the warehouse and production areas throughout the day, our proposed ALNS algorithm can be employed multiple times during the day to manage AGV routing and load handling with trolleys, effectively capturing and adapting to these changes.

CHAPTER VI

CONCLUSIONS

This study explores the effectiveness of various algorithms in solving the AGV routing problem, particularly within the context of Vestel Electronics' manufacturing operations. Through a comprehensive comparison of the traditional MILP approach with the Adaptive Large Neighborhood Search (ALNS) algorithm and the RoT approach, significant insights are gained into the performance differences across multiple scenarios and problem sizes.

The computational results of this research demonstrate that the ALNS algorithm consistently outperforms the MILP method in terms of both efficiency and solution quality. The ALNS algorithm shows markedly lower average CPU times and maintains stable performance across various instances, establishing it as a more suitable choice for large-scale and time-sensitive applications. In contrast, the MILP approach frequently fails to find feasible solutions within the allotted CPU time limit, particularly as the complexity of the problem increases.

Further analysis reveals the robustness of the ALNS algorithm considering varying conditions, such as the number of service points, trolleys, impact times, penalty values, and the number of AGVs. For instance, as the number of service points and trolleys increase, the performance of the MILP method deteriorates significantly, while the ALNS algorithm maintains low and stable gap values. This consistent pattern across different parameters, such as impact times and penalty values, underscores the reliability and efficiency of the ALNS algorithm, making it a robust tool for handling complex AGV routing problems.

When applied to the real-world data from Vestel Electronics' factory operations,

the ALNS algorithm achieves significantly lower total costs compared to the RoT approach. The detailed cost analysis highlights that while the distance cost is 1902 units, the tardiness penalty is 4040 units, emphasizing the substantial impact of scheduling penalties on the overall cost. These results highlight the importance of optimizing not only the routing but also the scheduling aspects of AGV operations to minimize penalties and improve efficiency.

In conclusion, this research contributes to the literature by providing a detailed comparison of the ALNS algorithm with traditional methods, demonstrating its superior performance in both computational efficiency and solution quality. The application of the ALNS algorithm to real-world data further underscores its practical utility, showcasing significant potential for cost reduction and operational efficiency. Future research could explore further enhancements to the ALNS algorithm, such as incorporating more sophisticated heuristics or hybrid approaches, to improve its performance even further in more challenging scenarios.

APPENDIX A

EXAMPLES OF HYPOTHETICAL INSTANCES

The tables presented in this appendix provide essential information about the service points involved in the Automated Guided Vehicle (AGV) routing problem. Each row corresponds to a single service point uniquely identified by a *PointID*. The *Type* column categorizes these points into depots (**d**), pickup points (**cp**), and delivery points (**cd**). Spatial positioning is detailed through the x and y columns, which specify the coordinates of each service point on a Cartesian plane, facilitating effective route planning and spatial analysis. The *Demand* column quantifies the load associated with each point, where positive values for pickup points indicate the amount to be collected and negative values for delivery points indicate the amount to be distributed. Temporal constraints are outlined by the *ReadyTime* and *DueDate* columns, defining the permissible time window for AGV arrivals; arrivals outside this window incur penalties, emphasizing the importance of timely operations. The *ServiceTime* column denotes the duration required to complete necessary tasks at each service point, such as loading or unloading. Lastly, the *PartnerID* column establishes the pairing between pickup and delivery points, ensuring that each collected load is correctly matched to its designated delivery destination, thereby maintaining a strict one-to-one correspondence. Collectively, these tables encapsulate the comprehensive spatial and temporal parameters crucial for optimizing the AGV routing process and achieving operational efficiency.

A.1 An example of hypothetical instance: 11 service points

Table 12: An example of hypothetical instance: 11 service points

PointID	Type	x	y	Demand	ReadyTime	DueDate	ServiceTime	PartnerID
D0	d	35.0	35.0	0.0	0.0	730.0	4.0	0
C1	cp	41.0	49.0	45.0	12.0	296.0	12.0	C6
C2	cp	55.0	22.0	55.0	14.0	291.0	12.0	C7
C3	cp	20.0	13.0	50.0	17.0	290.0	14.0	C8
C4	cp	40.0	20.0	35.0	19.0	299.0	12.0	C9
C5	cp	38.0	12.0	35.0	15.0	295.0	12.0	C10
C6	cd	17.0	10.0	-45.0	42.0	469.0	14.0	C1
C7	cd	25.0	19.0	-55.0	43.0	465.0	12.0	C2
C8	cd	12.0	27.0	-50.0	44.0	460.0	14.0	C3
C9	cd	40.0	14.0	-35.0	45.0	461.0	12.0	C4
C10	cd	36.0	10.0	-35.0	46.0	463.0	14.0	C5

A.2 An example of hypothetical instance: 15 service points

Table 13: An example of hypothetical instance: 15 service points

PointID	Type	x	y	Demand	ReadyTime	DueDate	ServiceTime	PartnerID
D0	d	35.0	35.0	0.0	0.0	850.0	4.0	0
C1	cp	30.0	40.0	25.0	35.0	322.0	15.0	C8
C2	cp	46.0	32.0	30.0	32.0	333.0	15.0	C9
C3	cp	33.0	24.0	35.0	37.0	325.0	15.0	C10
C4	cp	40.0	30.0	40.0	33.0	342.0	15.0	C11
C5	cp	40.0	43.0	45.0	35.0	339.0	15.0	C12
C6	cp	28.0	28.0	50.0	34.0	355.0	15.0	C13
C7	cp	28.0	42.0	30.0	36.0	352.0	15.0	C14
C8	cd	30.0	22.0	-25.0	52.0	560.0	15.0	C1
C9	cd	25.0	39.0	-30.0	55.0	565.0	16.0	C2
C10	cd	25.0	27.0	-35.0	53.0	563.0	15.0	C3
C11	cd	40.0	34.0	-40.0	58.0	561.0	17.0	C4
C12	cd	41.0	49.0	-45.0	54.0	563.0	15.0	C5
C13	cd	42.0	40.0	-50.0	55.0	562.0	14.0	C6
C14	cd	48.0	22.0	-30.0	56.0	561.0	16.0	C7

A.3 An example of hypothetical instance: 19 service points

Table 14: An example of hypothetical instance: 19 service points

PointID	Type	x	y	Demand	ReadyTime	DueDate	ServiceTime	PartnerID
D0	d	25.0	25.0	0.0	0.0	950.0	4.0	0
C1	cp	20.0	30.0	35.0	45.0	622.0	14.0	C10
C2	cp	41.0	27.0	35.0	42.0	633.0	15.0	C11
C3	cp	28.0	29.0	45.0	47.0	625.0	16.0	C12
C4	cp	35.0	24.0	50.0	43.0	642.0	13.0	C13
C5	cp	36.0	59.0	45.0	45.0	639.0	17.0	C14
C6	cp	21.0	23.0	50.0	44.0	655.0	15.0	C15
C7	cp	30.0	42.0	55.0	46.0	652.0	14.0	C16
C8	cp	23.0	25.0	55.0	42.0	660.0	16.0	C17
C9	cp	37.0	31.0	40.0	45.0	665.0	16.0	C18
C10	cd	44.0	51.0	-35.0	63.0	863.0	15.0	C1
C11	cd	40.0	34.0	-35.0	78.0	861.0	17.0	C2
C12	cd	41.0	49.0	-45.0	74.0	863.0	15.0	C3
C13	cd	42.0	40.0	-50.0	75.0	862.0	14.0	C4
C14	cd	38.0	22.0	-45.0	74.0	864.0	13.0	C5
C15	cd	14.0	31.0	-50.0	77.0	863.0	17.0	C6
C16	cd	49.0	41.0	-55.0	76.0	867.0	15.0	C7
C17	cd	33.0	17.0	-55.0	73.0	865.0	12.0	C8
C18	cd	26.0	22.0	-40.0	75.0	862.0	16.0	C9

REFERENCES

[1] Y. Cao, A. Yang, Y. Liu, Q. Zeng, and Q. Chen, "AGV dispatching and bidirectional conflict-free routing problem in automated container terminal," *Computers Industrial Engineering*, vol. 184, p. 109611, 2023.

[2] A. I. Corréa, A. Langevin, and L.-M. Rousseau, "Scheduling and routing of automated guided vehicles: A hybrid approach," *Computers Operations Research*, vol. 34, no. 6, pp. 1688–1707, 2007. Part Special Issue: Odysseus 2003 Second International Workshop on Freight Transportation Logistics.

[3] S.-Y. H. Ling Qiu, Wen-Jing Hsu and H. Wang, "Scheduling and routing algorithms for AGVs: A survey," *International Journal of Production Research*, vol. 40, no. 3, pp. 745–760, 2002.

[4] I. Vestel, "Corporate Information," 2024. Accessed: 2024-08-02.

[5] Buyfromturkey, "Vestel, the Largest Electronics Factory in Europe," 2024. Accessed: 2024-08-02.

[6] C. Liang, Y. Zhang, and L. Dong, "A Three Stage Optimal Scheduling Algorithm for AGV Route Planning Considering Collision Avoidance under Speed Control Strategy," *Mathematics*, vol. 11, no. 1, 2023.

[7] K. C. Vivaldini, L. F. Rocha, M. Becker, and A. P. Moreira, "Comprehensive Review of the Dispatching, Scheduling and Routing of AGVs," in *CONTROLO'2014 - Proc. of the 11th Port. Conf. on Autom. Control*, pp. 505–512, Springer International Publishing Switzerland, 2015.

[8] M. De Ryck, M. Versteyhe, and F. Debrouwere, "Automated guided vehicle systems, state-of-the-art control algorithms and techniques," *Journal of Manufacturing Systems*, vol. 54, pp. 152–173, 2020.

[9] H. Zou, W. Liu, and D. Zou, "The optimal investment strategies for university endowment funds based on principle component analysis," in *2017 29th Chinese Control And Decision Conference (CCDC)*, pp. 2755–2760, 2017.

[10] H. S. Hasan, M. S. Z. Abidin, M. Mahmud, and F. Mohd, "Automated guided vehicle routing: Static, dynamic and free range," *International Journal of Engineering and Advanced Technology*, vol. 8, no. 5C, pp. 1–7, 2019.

[11] J. Hu and Q. Ke, "Optimized control of geometric quantum discord," in *2015 IEEE International Conference on Information and Automation*, pp. 2379–2382, 2015.

[12] D. Zhang, C. Chen, and G. Zhang, "AGV Path Planning Based on Improved A-star Algorithm," in *2024 IEEE 7th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, vol. 7, pp. 1590–1595, 2024.

[13] N. Kokash, "An Introduction to Heuristic Algorithms," in *Proceedings of the University of Trento*, (Trento, Italy), Department of Informatics and Telecommunications, University of Trento, 2006.

[14] G. Ulusoy, F. Sivrikaya-Şerifoğlu, and Ümit Bilge, "A genetic algorithm approach to the simultaneous scheduling of machines and automated guided vehicles," *Computers Operations Research*, vol. 24, no. 4, pp. 335–351, 1997.

[15] I. A. Chaudhry, A. F. Rafique, I. A.-Q. Elbadawi, M. Aichouni, M. Usman, M. Boujelbene, and A. Boudjemline, "Integrated scheduling of machines and automated guided vehicles (AGVs) in flexible job shop environment using genetic algorithms," *International Journal of Industrial Engineering Computations*, vol. 13, pp. 343–362, 2022.

[16] S. M. Homayouni, S. H. Tang, N. Ismail, and M. K. A. Ariffin, "Using simulated annealing algorithm for optimization of quay cranes and automated guided vehicles scheduling," *International Journal of the Physical Sciences*, vol. 6, no. 27, pp. 6286–6294, 2011.

[17] X. Wang, H. Shi, and C. Zhang, "Path Planning for Intelligent Parking System Based on Improved Ant Colony Optimization," *IEEE Access*, vol. 8, pp. 65267–65273, 2020.

[18] C. Chen, E. Demir, and Y. Huang, "An Adaptive Large Neighborhood Search Heuristic for the Vehicle Routing Problem with Time Windows and Delivery Robots," *European Journal of Operational Research*, vol. 294, no. 3, pp. 1164–1180, 2021.

[19] S. N. Parragh, K. F. Doerner, and R. F. Hartl, "The vehicle routing problem with simultaneous pickup and delivery: A survey," *Transportation Science*, vol. 42, no. 2, pp. 189–231, 2008.

[20] I. Karaoglan, F. Altiparmak, I. Kara, and B. Dengiz, "A branch and cut algorithm for the location-routing problem with simultaneous pickup and delivery," *European Journal of Operational Research*, vol. 211, no. 3, pp. 318–332, 2011.

[21] H. Min, V. Jayaraman, and R. Srivastava, "A review of vehicle routing with simultaneous pickup and delivery," *Journal of Business Logistics*, vol. 20, no. 2, pp. 189–226, 1999.

[22] S. Ropke and D. Pisinger, "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows," *Transportation Science*, vol. 40, no. 4, pp. 455–472, 2006.

[23] S. Voigt, M. Frank, P. Fontaine, and H. Kuhn, "Hybrid adaptive large neighborhood search for vehicle routing problems with depot location decisions," *Computers & Operations Research*, vol. 146, p. 105856, 2022.

[24] S. Donmez, C. Koc, and F. Altiparmak, "The mixed fleet vehicle routing problem with partial recharging by multiple chargers: Mathematical model and adaptive large neighborhood search," *Transportation Research Part E: Logistics and Transportation Review*, vol. 167, p. 102917, 2022.

[25] X. Wang, Y. Chen, and M. Li, "An adaptive large neighborhood search for the multi-depot dynamic vehicle routing problem with time windows," *Computers & Operations Research*, vol. 125, p. 104364, 2024.

[26] T. Pichpibul, A. Udomsakdigool, and S. Boonkleaw, "A novel risk perspective on location-routing planning: An application in cash transportation," *Computers & Industrial Engineering*, vol. 174, p. 108745, 2024.

[27] N. Singh, Q.-V. Dang, A. Akcay, I. Adan, and T. Martagan, "A matheuristic for AGV scheduling with battery constraints," *European Journal of Operational Research*, vol. 298, no. 1, pp. 855–873, 2022.

[28] Y. Yang, M. Zhong, Y. Dessouky, and O. Postolache, "An integrated scheduling method for AGV routing in automated container terminals," *Computers & Industrial Engineering*, vol. 126, pp. 482–493, 2018.

[29] S. C. Leung, Z. Zhang, D. Zhang, X. Hua, and M. K. Lim, "A meta-heuristic algorithm for heterogeneous fleet vehicle routing problems with two-dimensional loading constraints," *European Journal of Operational Research*, vol. 225, no. 2, pp. 199–210, 2013.

[30] Y. Zhang, S. Wang, and G. Zhao, "Deep reinforcement learning for AGV path planning in complex environments," *Journal of Manufacturing Systems*, vol. 54, pp. 280–292, 2020.

[31] C. Liu, X. Xu, and Z. Li, "Dynamic routing for AGVs in a manufacturing environment using deep learning and genetic algorithms," *Journal of Intelligent Manufacturing*, vol. 30, no. 3, pp. 1101–1112, 2019.

[32] X. Wang, Y. Chen, and M. Li, "An adaptive large neighborhood search for the multi-depot dynamic vehicle routing problem with time windows," *Computers & Operations Research*, vol. 125, p. 104364, 2024.

[33] S. Allahyari, S. Yaghoubi, and T. Van Woensel, "A novel risk perspective on location-routing planning: An application in cash transportation," *Transportation Research Part E: Logistics and Transportation Review*, vol. 150, p. 102356, 2021.

[34] Q.-V. Dang, N. Singh, I. Adan, T. Martagan, and D. van de Sande, "Scheduling heterogeneous multi-load AGVs with battery constraints," *Computers & Operations Research*, vol. 136, p. 105517, 2021.

[35] D. Pisinger and S. Ropke, "A general heuristic for vehicle routing problems," *Computers & Operations Research*, vol. 34, no. 8, pp. 2403–2435, 2007.

[36] W.-Q. Zou, Q.-K. Pan, and M. F. Tasgetiren, "An effective iterated greedy algorithm for solving a multi-compartment AGV scheduling problem in a matrix manufacturing workshop," *Applied Soft Computing*, vol. 99, p. 106945, 2021.

VITA

Reşide Özkır graduated with distinction from Yeditepe University in January 2018, completing her degree in Industrial Engineering one semester ahead of schedule and ranking third in her department. After graduation, she worked for over three years in the field of intellectual property rights, where she gained significant experience and became a certified Turkish Patent Attorney in late 2019. She has been with Vestel Electronics for over three years, currently serving as a Senior Intellectual Property Specialist, while also pursuing a Master's degree in Industrial Engineering at Özyeğin University.