THE HONG KONG
POLYTECHNIC UNIVERSITY
香港理工大學
Pao Yue-kong Library
包玉剛圖書館

Copyright Undertaking

This thesis is protected by copyright, with all rights reserved.

By reading and using the thesis, the reader understands and agrees to the following terms:

1. The reader will abide by the rules and legal ordinances governing copyright regarding the use of the thesis.
2. The reader will use the thesis for the purpose of research or private study only and not for distribution or further reproduction or any other purpose.
3. The reader agrees to indemnify and hold the University harmless from and against any loss, damage, cost, liability or expenses arising from copyright infringement or unauthorized usage.

IMPORTANT

If you have reasons to believe that any materials in this thesis are deemed not suitable to be distributed in this form, or a copyright owner having difficulty with the material being included in our database, please contact lbsys@polyu.edu.hk providing details. The Library will look into your claim and consider taking remedial action upon receipt of the written requests.

Pao Yue-kong Library, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong

http://www.lib.polyu.edu.hk

ADVANCED OPTIMIZATION ALGORITHMS FOR SPLIT DELIVERY VEHICLE ROUTING PROBLEM WITH THREE-DIMENSIONAL LOADING CONSTRAINTS

ZHANG HAN

PhD

The Hong Kong Polytechnic University

2025

The Hong Kong Polytechnic University
Department of Computing

Advanced Optimization Algorithms for Split Delivery Vehicle Routing Problem with Three-Dimensional Loading Constraints

ZHANG Han

A thesis submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy
February 2025

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgment has been made in the text.

Signature: _____

Name of Student: ZHANG Han

Abstract

The Split Delivery Vehicle Routing Problem with Three-Dimensional Loading Constraints (3L-SDVRP) is a combination of the Split Delivery Vehicle Routing Problem (SDVRP) and the Three-Dimensional Packing Problem (3DPP), presenting significantly more challenges than the original two problems. There are two objectives: minimizing the number of vehicles used (or maximizing the average loading rate), and minimizing total travel distance. Solving the 3L-SDVRP is crucial for enhancing logistics and transportation efficiency across various industries, impacting both operational efficiency and cost-effectiveness. This thesis advances the state-of-the-art in solving the 3L-SDVRP through several key contributions.

First, we introduce novel and efficient search operators, specifically the Hierarchical Neighborhood Filtering (HNF) and Adaptive Knowledge-guided Search (AKS) operators. These operators enhance solution diversity and search efficiency in our evolutionary algorithm, significantly improving overall algorithm performance.

Second, we propose innovative methods to balance exploration and exploitation in meta-heuristic algorithms, achieving this balance in both global and local search-based algorithms.

Third, we develop a new multi-objective algorithm, the Pareto-based Evolutionary Algorithm with Concurrent Crossover and Hierarchical Neighborhood Filtering Mutation (PEAC-HNF). This algorithm effectively addresses the 3L-SDVRP under limited computational resources by optimizing multiple objectives simultaneously, providing

decision-makers with a diverse set of optimal solutions.

Fourth, we propose new local search-based algorithms that enhance the state-of-the-art SDVRLH2 algorithm, significantly reducing computational resource consumption while maintaining a high solution quality. These improvements are achieved through the integration of adaptive strategies and heuristic adjustments tailored to the specific characteristics of the 3L-SDVRP.

Fifth, we introduce an adaptive interactive routing-packing strategy, which combines the strengths of existing approaches to improve solution quality. This strategy adaptively adjusts packing patterns based on the vehicle's remaining space and the space requirements of different packing pattern at each node, ensuring efficient space utilization and reducing the number of vehicles required.

Sixth, comprehensive experimental studies demonstrate the superior performance of our proposed algorithms across various benchmark datasets. The results indicate that our methods provide higher quality solutions and, in many cases, outperform existing methods in terms of computational efficiency, especially for large-scale problems.

This thesis presents a suite of methodologies for addressing the 3L-SDVRP, each with its distinct advantages and applicability to specific industrial scenarios. For smaller-scale problems that necessitate a multi-objective approach to generate a diverse set of solutions with varying degrees of balance between the two objectives, the PEAC-HNF algorithm proposed in Chapter 3 is recommended. In contrast, for larger-scale problems where computational resources are scarce, the efficient local search algorithm presented in Chapter 4 is a more suitable choice. For larger-scale problems that require high-quality solutions, the AKS algorithm proposed in Chapter 6 is the preferred option. Furthermore, given the importance of the interaction between the packing and routing processes in solving the 3L-SDVRP, the adaptive routing-packing strategy proposed in Chapter 5 offers a flexible approach that can be broadly applied across various search algorithms, enhancing their overall effectiveness.

Overall, the methods proposed in this thesis, including the HNF and AKS operators and the balance of exploration and exploitation, are flexible and applicable to other combinatorial optimization problems. This thesis contributes to the field of combinatorial optimization by providing robust, efficient, and adaptive solutions to the complex 3L-SDVRP, with significant implications for industrial applications and future research directions.

Publications Arising from the Thesis

1. Han Zhang, Qing Li, and Xin Yao. (2024). "PEAC-HNF: A Novel Multi-Objective Evolutionary Algorithm for Split Delivery Vehicle Routing Problems with Three-Dimentional Loading Constraints", In *IEEE Transaction on Emerging Topics on Computational Intelligence (TETCI)*. Early Access. DOI: 10.1109/TETCI.2024.3499992
   Supports Chapter 3.

2. Han Zhang, Qing Li, and Xin Yao. (2024). "PEACH: A Multi-Objective Evolutionary Algorithm for Complex Vehicle Routing with Three-Dimensional Loading Constraints". In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO'24)*. pp.231-234. DOI:10.1145/3638530.3654333
   Supports Chapter 3.

3. Han Zhang, Qing Li, and Xin Yao. (2025). "An Efficient Local Search Algorithm for Split Delivery Vehicle Routing Problem with Three-Dimensional Loading". In *Memetic Computing*. DOI: 10.1007/s12293-025-00451-9
   Supports Chapter 4.

4. Han Zhang, Qing Li, and Xin Yao. (2024). "An Adaptive Interactive Routing-Packing Strategy for Split Delivery Vehicle Routing Problem with 3D Loading

Constraints". In *Proceeding of the Genetic and Evolutionary Computation Conference (GECCO'24)*. pp. 249-257. DOI: 10.1145/3638529.3653991
Supports Chapter 5.

5. Han Zhang, Qing Li, and Xin Yao. (2024). "Knowledge-Guided Optimization for Complex Vehicle Routing with Three-Dimentional Loading Constraints". In *the 18th International Conference on Parallel Problem Solving from Nature (PPSN'24)*. DOI: 10.1007/978-3-031-70055-2_9
Supports Chapter 6.

Acknowledgments

When I started my PhD, I envisioned the day I would graduate, and now, that moment is near, marking the end of my doctoral journey. In ancient China, scholars pursued rigorous studies with the hope of excelling in imperial examinations (科舉考試), seeking to earn official positions, bring honor to their families, or achieve great ambitions. This arduous journey was known as "ten years of hard study" (十年寒窗). Today, while the imperial examination system is long gone, diligent study remains the primary way for many to achieve their aspirations. Counting from primary school, I have been studying for over twenty years; from university, it has been a decade.

First and foremost, I express my heartfelt gratitude to my supervisors, Professor Xin Yao and Professor Qing Li. There is an old Chinese saying, "A day as a teacher, a lifetime as a father", signifying the immense gratitude owed to one's teacher. In modern times, as we encounter many teachers from primary school to university, the bond may not seem as profound. However, Professor Xin Yao stands as my lifelong mentor. He has always shown me immense understanding and respect. He guided my research with patience, imparting a high standard of scientific rigor and deep academic insights. His mentorship in reading literature, conducting research, designing algorithms, and writing papers has shaped my academic skills. Additionally, his remarkable personal qualities have profoundly influenced me, teaching me how to conduct myself and navigate life's challenges. He has always been my guiding light and role model. Without his guidance, I would not be who I am today. Professor

Qing Li provided significant support and guidance, with insightful and constructive advice crucial to my academic progress. His unfailing support at critical moments and constant encouragement were invaluable, serving as a great source of inspiration throughout my academic journey. I am forever grateful for their support.

Furthermore, the unwavering support from my family has been a driving force behind my perseverance. Coming from a rural background, my parents' hard work and sacrifices created a nurturing environment that supported my education from university to a doctorate. Their care and confidence in me have always been my source of strength. I am also deeply grateful to my extended family members, including my aunts, uncles, cousins, and grandparents, for their constant support and warmth.

I also extend my gratitude to the teachers, classmates, and friends who have accompanied and helped me throughout this journey. Special thanks to Dr. Changwu Huang and Dr. Liyan Song for their guidance and support during challenging times, and to Prof. Ke Tang and Dr. Wenqi Fan for their kind and insightful advice. I appreciate the career support from Dr. Bo Yuan, Dr. Shuyi Zhang, and Xialiang Tong, and the help from Chenran Zhang, Zhi Yang, Yijing Liu, Dr. Weijie Zheng, Dr. Zilu Wang, Yuannan Ji, Ruihang Hu, Yunfan Zhou, Qi Wang, Junhua Huang, Dr. Yufei Kuang, Xiaoyan Zhao, Yunce Zhao, Dr. Gan Ruan, Dr. Xinming Shi, Dr.Da Ren, Jiahao Wu, Zeyu Dai, and Dr. Borui Gong.

I would also like to acknowledge the support from several research funding programs, including the National Key R&D Program of China (Grant No. 2023YFE0106300), the National Natural Science Foundation of China (Grant No. 62250710682), the Guangdong Provincial Key Laboratory (Grant No. 2020B121201001), the Guangdong Major Project of Basic and Applied Basic Research (Grant No. 2023B0303000010), and the Program for Guangdong Introducing Innovative and Entrepreneurial Teams (Grant No. 2017ZT07X386). Their support has been essential to the completion of my research.

In ancient Chinese philosophy, the I Ching (Book of Changes, 易經) uses the principles of Yin (陰) and Yang (陽) and sixty-four hexagrams (六十四卦象) to describe the universe. The last hexagram is called "Fire Water Not Yet Complete" (火水未濟卦). Fire, characterized by its upward movement, over water, characterized by its downward movement, signifies a state where the two elements do not cross and interact. This symbolizes incompletion and difficulty in achieving goals, hence 'Not Yet Complete'. This hexagram suggests that things are not yet finished but are full of potential for development, as implied by the dynamic interaction of Yin and Yang. The philosophy within this final hexagram reflects profound wisdom, indicating that even at the end, there is boundless potential for growth and transformation.

"Life, you see, never is as good or as bad as one thinks. People are stronger or weaker than one believes. Sometimes a simple phrase can make us cry, and other times we discover that we have gone through incredible ordeals with courage." — Guy de Maupassant, *A Life*

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we begin with a basic introduction to the Split Delivery Vehicle Routing Problem with Three-Dimensional Loading Constraints (3L-SDVRP) in Section 1.1. Subsequently, in Section 1.2, we discuss the challenges faced in researching the 3L-SDVRP and the motivations behind our study. In Section 1.3, we summarize the contributions made by this thesis. Finally, Section 1.4 outlines the organization of the thesis.

1.1 The Split Delivery Vehicle Routing Problem with Three-Dimensional Loading Constraints

Vehicle Routing Problems (VRPs) constitute a critical category of combinatorial optimization problems (COPs), boasting a broad spectrum of practical applications, notably in logistics and supply chain management [3] [98]. These problems have attracted considerable attention in both academic and industrial circles over several decades. Although numerous efficient and effective optimization algorithms for VRP variants with simple constraints have been developed [29] [97], the varied and in-

tricate nature of constraints encountered in diverse real-world scenarios means that existing research does not entirely satisfy the complex demands of industrial applications. Encouraged by advancements in hardware and the expansion of computational capabilities, researchers are increasingly focusing on more complex VRPs, marked by more elaborate constraints and larger scales, thus moving closer to addressing the complexities of real-world industrial settings. As a result, the study of these complex VRPs continues to be a key area of focus in the field.

This thesis investigates the Split Delivery Vehicle Routing Problem with Three-dimensional Loading Constraints (3L-SDVRP), a complex combinatorial optimization challenge that integrates two NP-hard problems: the split delivery vehicle routing problem (SDVRP) and the three-dimensional packing/loading problem (3DPP). This integration not only amplifies the complexity but also broadens the practical relevance of 3L-SDVRP, making it significant for both theoretical exploration and real-world application [26] [72].

The 3L-SDVRP requires strategic vehicle routing and 3D box packing, with each vehicle starting at a starting point (depot), traveling to various customer nodes to load designated boxes, and then proceeding to an endpoint. Depending on different scenarios, the starting and ending points may or may not be the same. Vehicles and boxes are treated as 3D rectangles [13]. The boxes at each node vary in 3D size (length, width, and height) and must be packed into vehicles in a manner that optimizes space usage while adhering to 3D loading constraints. Distinct from simple VRP variants ([52] [54] [71]), the 3L-SDVRP is characterized by three principal features that increase its complexity and applicability to real scenarios:

- Limited capacity: Each vehicle has a finite capacity, and its 3D loading space is considered as a large 3D rectangular space.
- Split Delivery: In typical VRP settings, each node is serviced by only one vehicle, with all deliveries from that node loaded onto that vehicle. In real-world

applications, vehicles have finite capacities, and the quantity and 3D sizes of boxes at each node may vary greatly. Such differences can result in situations where the total load from a node exceeds a vehicle's maximum capacity, or when visiting a node, a vehicle's remaining capacity is insufficient for all boxes. Therefore, the 3L-SDVRP allows for split deliveries where multiple vehicles can service a single node.

- 3D Loading Constraints: Unlike traditional VRPs that consider only the volume and/or weight of boxes, 3L-SDVRP requires careful planning of the loading sequence and placement of cuboid-shaped boxes within each vehicle. This requires advanced approaches to optimize both the vehicle routing and the 3D packing of boxes, ensuring that each vehicle's load does not exceed its capacity while maximizing the use of available space.

The dual objectives of 3L-SDVRP are to minimize the number of vehicles used (or maximize the average vehicle loading rate) and the total travel distance, thereby reducing operational costs and enhancing delivery efficiency. This is particularly challenging due to the need to balance two conflicting objectives and ensure the load feasibility of multiple vehicles. The mathematical formulation of 3L-SDVRP is provided in Chapter 2.

1.2 Challenges and Motivations

Exact algorithms, such as linear and integer programming, have played a central role in the solution of many combinatorial optimization problems. However, the 3L-SDVRP problem combines two well-known NP-hard subproblems: the vehicle routing problem (VRP) and the three-dimensional loading problem (3DLP). As a result, it is characterized by highly nonlinear and nonconvex constraints, as well as an extremely large and complex solution space. While exact methods, such as linear programming,

integer programming, and branch-and-bound, are theoretically capable of finding the optimal solution, they are infeasible for such complex problems. It should be emphasized that the 3L-SDVRP features highly nonlinear and strongly coupled constraints between routing and three-dimensional loading. This intrinsic nonlinearity makes the problem intractable for exact optimization approaches, regardless of instance size. In most cases, it is not even possible to construct an exact mathematical model that faithfully represents all the practical constraints, let alone solve it optimally using standard methods such as linear or integer programming. Consequently, metaheuristic and other intelligent optimization algorithms are not merely preferred, but are essentially the only feasible approach for tackling such problems.

Attempting to relax 3L-SDVRP to be solvable by exact methods proves impractical for two main reasons: (1) reducing it to simpler VRP or TSP variants still remains NP-hard, where exact algorithms struggle with large-scale problems; (2) reducing its complexity by relaxing crucial constraints would lose essential features of the problem, differing significantly from the real-world scenario and potentially making solutions inapplicable or even infeasible. Consequently, intelligent optimization algorithms (e.g., meta-heuristic algorithms) stand out as effective and prevalent approaches for solving such complex problems. These methods are based on a generate-and-test iterative strategy [106], where each iteration involves the generation of a new set of potential solutions from the existing ones using various search operators, with the hope of finding improved solutions. This process is iteratively repeated to ultimately find an approximate optimal solution.

1.2.1 Global Search-Based Multi-Objective Optimization for 3L-SDVRP

In some studies, global search-based methods have been employed to solve the 3L-SDVRP, including MOEA/D [51], estimation of distribution algorithms [50], and

genetic algorithms [65], [70]. Global search-based algorithms offer the advantage of exploring the solution space more comprehensively, which increases the likelihood of finding near-optimal or optimal solutions. They are particularly effective in avoiding local optima, a common issue in complex optimization problems. However, these algorithms typically require significant computational resources and time due to their extensive search processes, especially when applied to large-scale problems. For complex combinatorial optimization problems like the 3L-SDVRP, global search-based algorithms are often inefficient and need improvement in solution quality. Moreover, their convergence rates can be slower compared to local search-based algorithms, making them less practical for problems requiring real-time or near real-time solutions.

Furthermore, the 3L-SDVRP has two objectives. Some studies attempt to solve it using multi-objective evolutionary algorithms to balance vehicle loading ratios (or the number of vehicles required) and travel cost. Moura [65] tackled the 3L-SDVRP using the Multi-Objective Genetic Algorithm (MOGA) but relaxed some vital constraints, potentially limiting the solution's relevance to real world scenarios. Recently, Liu et al. [51] introduced a Multi-objective Evolutionary Algorithm based on Decomposition by Offline Machine learning (MOEA/D-OM) to address the 3L-SDVRP. They utilized a machine learning model to predict the feasibility of packing arrangements for given routes, thus bypassing packing computations for solutions without feasible packing arrangements and reducing the algorithm' s runtime. However, the success of this strategy relies on the algorithm generating a substantial number of packing infeasible routes during its search process. Moreover, the offline-trained machine learning model encounters generalization challenges. Therefore, the use of multi-objective approaches in addressing the 3L-SDVRP is still relatively unexplored. Furthermore, the time-intensive 3D packing process in 3L-SDVRP demands substantial computational resources which are often limited in real-world scenarios [47].

At present, addressing the 3L-SDVRP as a multi-objective problem and improve the performance of global search-based algorithms within limited computational resources

remains a significant research challenge. We explore this topic in detail in Chapter 3, developing a novel multi-objective algorithm for 3L-SDVRP [119].

1.2.2 Local Search-Based Single-Objective Optimization for 3L-SDVRP

Local search-based algorithms are commonly employed to solve 3L-SDVRP, including conventional local search [10] [66], tabu search [14] [107], and simulated annealing [13]. These algorithms are highly effective due to their speed and ability to find high-quality solutions within a localized region of the solution space. They excel in fine-tuning solutions and are particularly efficient for problems with smaller search spaces or well-defined neighborhoods. However, their primary disadvantage is the tendency to get trapped in local optima, which limits their ability to comprehensively explore the global solution space. This limitation makes them less effective for problems with complex landscapes or multiple optimal solutions, where a broader perspective is necessary to avoid suboptimal results.

Additionally, in real-world scenarios, minimizing the number of vehicles is often more critical than reducing travel distances due to the significant higher costs associated with acquiring and maintaining more vehicles and hiring additional drivers compared to the expenses incurred from increased travel distance. Consequently, most research treats 3L-SDVRP as a single-objective problem [10] [13] [14] [50] [66] [76] [107].

Given these insights, we explored the local search-based approach to solve 3L-SDVRP. In alignment with existing research, we addressed 3L-SDVRP as a single-objective problem, prioritizing vehicle reduction as the primary objective and total travel distance as a secondary objective. We also employed widely used datasets to ensure our results are comparable with existing studies. In Chapter 4, we develop an effective local search-based method to solve 3L-SDVRP efficiently.

1.2.3 Interactive Routing-Packing Strategy for 3L-SDVRP

Fundamentally, any 3L-SDVRP solution must address both routing decisions (determining which nodes each vehicle should visit) and packing decisions (designing the 3D packing plan for each vehicle). These decisions substantially influence the final solution quality, making the interactions between routing and packing during the solution process crucial for algorithm performance. In our study, we refer to these interactions as "interactive routing-packing strategies". The term "interactive" denotes the mutual influence of routing and packing decisions: routing decisions dictate which nodes' boxes a vehicle will load, while packing decisions directly affect the number of boxes that can be loaded within the vehicle's limited capacity.

Prevailing interactive routing-packing strategies are mainly divided into two paradigms: routing first packing second (R1P2), and packing first routing second (P1R2). The R1P2 strategy [14] adapts packing decisions during the route search procedure, allowing them to change in response to routing decisions. On the contrary, the P1R2 strategy [10] implies making packing decisions before routing, thereby reducing the complexity of the routing phase to a direct SDVRP. Moreover, [10] introduced a two-customer segment pattern (2C-SP) into P1R2 strategy to further reduce the number of vehicles. The 2C-SP method involves combining two nodes after the packing decisions of each node have been made, and then repacking them. If creating a 2C-SP saves more loading space than packing each node individually, it is maintained.

In the R1P2 strategy, packing decisions are made dynamically during the route planning process. This approach offers considerable flexibility, as packing can be adjusted in response to routing changes. However, this flexibility comes at the cost of increased computational complexity, as each routing adjustment necessitates re-solving the packing problem. On the other hand, the P1R2 strategy determines packing decisions prior to route planning. This sequence requires solving the packing problem only once, which significantly speeds up the solution process. Nonetheless, the draw-

back of P1R2 is its rigidity: once packing decisions are made, they cannot adapt to subsequent routing changes. This lack of adaptability can hinder efforts to minimize the number of vehicles used.

Motivated by the analysis provided, we introduce an adaptive interactive routing-packing strategy [115] in Chapter 5.

1.2.4 Knowledge-Guided Optimization for 3L-SDVRP

As problems from real-world scenarios become increasingly complex and scale larger, enhancing the efficiency and performance of meta-heuristic algorithms becomes critically important. Traditional meta-heuristic methods solve problems without relying on domain-specific knowledge. These methods assume zero prior knowledge about the problem, necessitating a search from scratch [87]. However, as we delve deeper into solving a problem, our understanding improves, and we gather useful information. This accumulated knowledge can assist in solving related problems more effectively and efficiently, rather than starting the search anew each time. By integrating domain-specific knowledge into the optimization process, known as knowledge-guided optimization, the efficiency and effectiveness of search algorithms can be significantly enhanced. This concept leverages insights, patterns, and heuristics derived from expert knowledge, historical data, and problem-specific characteristics to guide the search process more intelligently. For instance, in dynamic and uncertain environments, knowledge extracted from previously solved problems can be transferred to solving newly emerged problems, aiding in obtaining better solutions [80] [81] [82].

Search operators are crucial in meta-heuristic algorithms as they determine the search direction and step size in the solution space. Many general search operators have been designed for solving vehicle routing problems (VRPs) and other combinatorial optimization problems (COPs), such as swap operators, shift operators, k-opt operators [35], etc. Traditional search operators, while commonly used, often lack specific

domain knowledge crucial for complex or large-scale problems, leading to poor efficiency and solution quality. Some research addresses this by integrating domain knowledge into search operators to enhance efficiency and solution quality. Examples include the merge-split (MS) operator for the capacitated arc routing problem (CARP) [89], and the region-focused operator for the multidepot multidisposal-facility multitrip capacitated vehicle routing problem (M3CVRP) [49]. Such domain-specific operators require a tailored approach, designing unique operators for different problems based on their specific characteristics.

Additionally, search step size, defining the extent of solution change by search operator per iteration, is crucial for algorithm performance. Methods for 3L-SDVRP can be categorized into local search-based methods [10] [13] [14] [66] [107] with smaller step sizes and global search-based methods [50] [51] [65] [70] with larger ones. Although local search is efficient, it risks falling into local optima, whereas global search tends to converge more slowly. Given the crucial role of search step size, researchers have proposed various methods to combine or balance both large and small step sizes. Yao [104] initially employed a large search step size in simulated annealing (SA) and theoretically demonstrated that SA with a large neighborhood size is more effective than with a small neighborhood size. He further extended this idea by developing the adaptive large neighborhood search method, validating its effectiveness on the Traveling Salesman Problem (TSP) [105]. Building on this foundation, further research has been conducted [63] [79] [86]. Additionally, the Memetic Algorithm [64] combines global and local search, effectively balancing step sizes and applying them across various combinatorial optimization problems (COPs) [17] [89]. However, these methods cannot be directly applied to 3L-SDVRP due to the problem's intricate constraints. Moreover, they lack the domain knowledge guidance which is important for effectively tackling such a complex problem. Thus, achieving a balanced trade-off between large and small step sizes in this context remains a significant challenge.

In Chapter 6, we apply domain knowledge to design an Adaptive Knowledge-guided

Insertion (AKI) operator, which we integrate into a local search framework to develop an Adaptive Knowledge-guided Search (AKS) algorithm [117]. The AKS algorithm leverages this domain knowledge to navigate the search process and balances large and small step sizes, thereby enhancing the quality of solutions.

1.3 Contributions

The main contributions of this thesis are as follows:

- **More Efficient Search Operators**: We introduced new, efficient search operators. Specifically, the Hierarchical Neighborhood Filtering (HNF) operator increases offspring diversity and enhances search efficiency. Additionally, we developed the Adaptive Knowledge-guided Search (AKS) operator, which integrates domain knowledge and features larger search step sizes. These new operators, when combined with search algorithms, significantly improve algorithm performance.

- **New Methods for Balancing Exploration and Exploitation**: We explored methods to balance exploration and exploitation in meta-heuristic algorithms, proposing new solutions for achieving this balance in both global and local search-based algorithms. In global search-based algorithms, we utilized crossover for exploration and the HNF operator for exploitation. In local search-based algorithms, traditional neighborhood operators were used for fine-grained local search (exploitation), and the AKI operator was employed for larger step size exploration. Our approach achieves a well-balanced exploration and exploitation, thereby enhancing the algorithms' search capabilities.

- **Novel Multi-Objective Algorithm**: We proposed a new multi-objective algorithm, the Pareto-based Evolutionary Algorithm with Concurrent crossover

and Hierarchical Neighborhood Filtering mutation (PEAC-HNF). This algorithm effectively solves the 3L-SDVRP under limited computational resources, demonstrating its ability to balance multi-objective optimization efficiently.

- **More Efficient Local Search Algorithms**: Building on the state-of-the-art algorithm SDVRLH2 [10], we developed new local search algorithms that significantly reduce computational resource consumption. Furthermore, the Adaptive Knowledge-guided Search (AKS) algorithm, which integrates the AKI operator within a local search framework, enables more efficient discovery of high-quality solutions.

- **New Interactive Routing-Packing Strategy**: We proposed a new interactive routing-packing strategy that builds upon existing strategies. This new approach combines the strengths of current strategies, significantly improving solution quality.

- **Extensive Experimental Studies**: Through extensive experimental validation and comparative analysis, we demonstrated the superior performance of our proposed algorithms across various widely used benchmark datasets, showing significant improvements in solution quality and/or computational efficiency.

Uniqueness and Significance of This Thesis What distinguishes this thesis from prior works is its comprehensive and integrated approach to the 3L-SDVRP—a problem that tightly couples the complexities of vehicle routing and three-dimensional packing. Rather than decoupling these aspects or addressing them in isolation, the proposed algorithms are specifically designed to capture and exploit their interaction, introducing adaptive routing-packing interactive strategy and knowledge-guided operators that leverage the structural properties of the problem.

A key feature of the proposed methods is the explicit balance between exploration and exploitation. The AKI operator and concurrent crossover–mutation schemes

are developed to broaden the search space (exploration), while the HNF mutation and multi-neighborhood operators intensify the search around promising solutions (exploitation). This balance results in both high-quality and robust solutions.

Experimental results (see Chapters 3–6) demonstrate that our methods consistently outperform state-of-the-art algorithms in both solution quality and computational efficiency across multiple standard benchmarks. For instance, the proposed PEAC-HNF algorithm achieves better results than state-of-the-art multi-objective algorithms on all EMO competition instances. On datasets B-Y, Sha, and SD, the AKS algorithm reduces the number of vehicles by over 58 and the total travel distance by 12.78% (7.74%) on small (large)-scale problem instances compared to SDVRLH2 [10]. These results exemplify the practical impact and superiority of the proposed innovations.

It is noteworthy that the methods proposed in this thesis, such as the HNF operator, the AKI operator, and the approach for balancing exploration and exploitation, are not limited to solving the 3L-SDVRP. They can be applied to other complex combinatorial optimization problems as well.

Overall, these advancements not only address fundamental challenges in combinatorial optimization, but also provide practical tools for real-world applications in logistics and supply chain optimization.

1.4 Thesis Organization

The rest of this thesis is organized as follows:

- In Chapter 2, we introduce the notations and mathematical formulation of the 3L-SDVRP in detail. We also review the literature, focusing on methodological classifications for solving 3L-SDVRP, interactive routing-packing strategies, and the diverse search operators and step size balancing techniques.

- In Chapter 3, we develop a Hierarchical Neighborhood Filtering (HNF) mutation operator, characterized by: (1) Using diverse neighborhood structures like swap, 2-opt, and 3-opt to create a wide range of offspring from a single parent, thus improving solution diversity and algorithm exploitation capability. (2) It adopts a hierarchical approach to mutation, prioritizing individuals with higher nondomination ranks for more focused search on promising candidates. (3) The offspring undergo a filtering process, where some individuals are removed if they meet specific criteria, enhancing search efficiency and reducing unnecessary fitness evaluations. By incorporating the HNF mutation into the Evolutionary Algorithm (EA) framework, we have developed a novel Pareto-based Evolutionary Algorithm with Concurrent crossover and Hierarchical Neighborhood Filtering mutation (PEAC-HNF) for 3L-SDVRP. The HNF mutation enhances PEAC-HNF's exploitation capabilities, complementing the EA's inherent exploration strength, thereby achieving a better balance between exploration and exploitation. Additionally, PEAC-HNF executes crossover and mutation processes concurrently, allowing an individual to undergo either or both processes in parallel, but not in a sequential manner. Our PEAC-HNF optimizes the algorithm's efficiency and effectiveness in navigating the problem space. PEAC-HNF was evaluated against baselines and state-of-the-art algorithm for multi-objective 3L-SDVRP, e.g., MOGA [65] and MOEA/D-OM [51], demonstrating its efficiency. Further experimental studies validated the crucial role of the HNF mutation on improving algorithmic performance.

- In Chapter 4, we introduce a new algorithm based on the state-of-the-art SDVRH2 algorithm [10] for solving 3L-SDVRP, which significantly enhances search efficiency and solution quality. In our proposed algorithm: (1) We improve the box loading, subspace generation, and 2C-SP construction methods in the loading procedure to enhance the loading performance and reduce the number of vehicles used. (2) We design three new search operators that leverage

the problem's characteristics as the heuristic information to improve search efficiency. (3) We propose an adaptive splitting strategy that dynamically determines whether to split a node's boxes based on the status of the vehicle and node, thereby further reducing computational resource consumption. (4) We develop a new post-optimization method that can further reduce the number of vehicles.

- In Chapter 5, we propose an adaptive interactive routing-packing strategy that combines the advanced features of existing interactive strategies. Our strategy introduces adaptability into the routing process, allowing for a choice between independent loading of a node (aligned with the P1R2 strategy) and joint loading of consecutive nodes (utilizing the idea of 2C-SP). This flexible approach, which permits loading adjustments in response to route modifications, effectively reflects the core principle of the R1P2 strategy. The effectiveness of our strategy has been rigorously validated through computational experiments. Empirical evidence indicates that our strategy yields solutions that are comparable to or significantly superior to existing strategies, particularly in terms of the number of vehicle used. Moreover, despite the crucial role of interactive routing-packing strategies in addressing complex 3L-SDVRP, there has been a lack of comprehensive investigation or comparison of these strategies. Our research addresses this gap by offering a detailed comparative analysis and evaluation of existing interactive routing-packing strategies, substantiated by extensive experimental validation.

- In Chapter 6, we incorporate domain knowledge into the optimization algorithm to direct the search process effectively. First, heuristics are extracted from domain knowledge. Specifically, based on the "giant tour" representation, we develop a hypothesis about "what constitutes a good giant tour" through observation. We use a node insertion approach to change the order of nodes in the current giant tour to improve its quality and propose two node inser-

tion rules. Then, an Adaptive Knowledge-guided Insertion (AKI) operator is developed which can adaptively select suitable node insertion rules based on node distribution. The proposed AKI operator utilizes domain knowledge and has a large search step size. The AKI operator is integrated into a local search framework to form the Adaptive Knowledge-guided Search (AKS) algorithm. In the AKS algorithm, traditional search operators conduct searches with small step size (exploitation), while our proposed AKI operator performs searches with larger step size (exploration), thereby enhancing the search capability of the algorithm. Comprehensive experimental results also demonstrate the effectiveness of the AKS algorithm.

- In Chapter 7, we conclude this thesis and discuss potential directions for future research.

Chapter 2

Background and Literature Review

In this chapter, we begin by detailing the notations and mathematical formulation of 3L-SDVRP. We then present a comprehensive literature review, investigating different search paradigms, interactive routing-packing strategies, and search operators. This thorough review has allowed us to grasp the current state of research, pinpoint shortcomings in existing studies, and clearly establish the motivation for our research.

2.1 Mathematical Formulation of 3L-SDVRP

The 3L-SDVRP includes starting and ending points, along with multiple customer nodes. Each node contains boxes of different weight, 3D sizes (length, width, height) and quantities. Vehicles, originating from the starting point and available in multiple types with different 3D capacities, visit these nodes to load boxes and then transport them to the endpoint. This requires decisions on the number of vehicles and the route for each vehicle. Additionally, a packing scheme must detail which boxes are assigned to which vehicles, their spatial arrangement, and the order of loading. Notably, the starting and ending points can be either the same [10] [13] [14] [50] [65] [66] [76] [107] or different [51] [70], depending on the specific industrial scenarios, and there is no

limit to the number of vehicles available.

The 3L-SDVRP has two objectives: minimizing the number of vehicles used or maximizing the average vehicle loading rate, and minimizing the total travel distance (ttd). Some studies treat 3L-SDVRP as a multi-objective problem, considering the practical need to balance both objectives in industrial applications [51], [65]. However, extensive research indicates a general preference for minimizing the number of vehicles over reducing travel distances [10] [13] [14] [66] [76] [107]. This preference is primarily due to the significantly higher costs associated with acquiring, maintaining, and staffing additional vehicles, which far exceed the expenses from extended travel distances. In such research, when evaluating two solutions, s_1 and s_2, s_1 is considered superior if it involves fewer vehicles than s_2, or, if the number of vehicles is the same, s_1 has a lower ttd than s_2.

For problem formulation, different studies may consider varying specific problem characteristics and constraints. Tab. 2.1 summarizes the problem properties considered in our study and other research on the 3L-SDVRP. It is evident that the problem exhibits varying characteristics in different scenarios, which must be taken into account when designing algorithms.

2.1.1 Notations

Assuming that there are N customer nodes/sites, a starting point (with index 0), and an ending point (with index $N + 1$). The distances between all pairs of nodes are known a priori. Additionally, N sets of boxes are provided, with all boxes at each node belonging to a single set. Each box is characterized by its weight and 3D dimensions (i.e., length, width, and height). The objective is to allocate vehicles to transport all boxes from the nodes to the ending point, with all vehicles departing from the starting point.

The notations used in the mathematical formulation are shown in Tab. 2.2. In 3L-

Table 2.1: 3L-SDVRP problem properties considered in different research.

Literature	BD	HV	TW	WL	LIFO	Re	Or	VS	FS	VN
Liu et al. [51]	✗	✓	✗	✓	✓	✗	✓	✓	✗	✗
Rajaei et al. [76]	✓	✓	✗	—	✓	✓	✓	✗	✓	✗
Pei et al. [70]	✗	✓	✗	✓	✓	✗	✓	✓	✗	✗
Bortfeldt and Yi [10]	✓	✓	✗	✓	✓	✗	✓	✓	✗	✗
Chen et al. [14]	✓	✗	✓	✓	✗	✗	✗	✗	✗	✓
Li et al. [50]	✓	✓	✗	✓	✓	✗	✓	✓	✗	✗
Yi and Bortfeldt [107]	✓	✗	✗	✗	✗	✗	✓	✓	✗	✗
Ceschia et al. [13]	✓	✓	✗	✓	✓	✓	✓	✗	✗	✓
Moura and oliveira [66]	✓	✗	✓	✗	✗	✗	✓	✗	✓	✓
Moura [65]	–	✗	✓	✗	✗	✗	✗	✗	✓	–
Ours	✓	✗	✗	✓	✓	✗	✓	✗	✓	✗

Note: BD = back to depot; HV = heterogeneous vehicles; TW = time window; WL = weight limit; LIFO = last-in-first-out; Re = reachability; Or = orientation; VS = vertical stability; RS = robust stability; FS = full support stability; SS = surrounded stability; LBS = load bearing strength; VN = vehicle number limit; The symbol "-" means the relative entry is not mentioned in the literature.

SDVRP, both the vehicle containers and the boxes are treated as rectangular solids. We use one vertex of the vehicle container as the coordinate origin for convenience, aligning the x-, y-, and z-axes with the container's length, width, and height, respectively. This configuration is depicted in Fig. 2.1, allowing the placement of each box within the vehicle to be uniquely determined by coordinates in this system. Decision variables include the number and type of vehicles utilized (n), each vehicle's route, and the packing arrangement, detailing which boxes are assigned to which vehicles, their 3D coordinates, and the sequence in which the boxes are loaded into each vehicle.

Table 2.2: Notations used in the mathematical formulation

Notations	Description
$G = (V, E)$	road network, a complete directed graph
$V = \{0, 1, ..., N, N+1\}$	the set of vertices; 0 is the starting point; $N+1$ is the ending point; $1 \sim N$ are customer nodes
$E = \{(i, j) \mid i, j \in V,\ i \neq j\ \}$	the set of edges
d_{ij}	non-negative travel distance
$O = \{K_i \mid i = 1, \ldots, M\}$	one delivery order/instance; $1 \leq M \leq N$
$K_i = \{1, \ldots, m_i\}$	the set of boxes that need to be transported in the node i; m_i: # boxes in the node i
$A_i^t \subset K_i$	the set of boxes that are loaded in vehicle t in the node i
l_k, w_k, h_k	the length, width, and height of the box k
q_k, v_k	the weight and volume of the box k
x_k, y_k, z_k	the coordinate of the center of the box k
$L_t, W_t, H_t,\ t = 1, \ldots, n$	the length, width, height of container vehicle t; n: total # vehicles used to transport boxes
$Q_t,\ t = 1, \ldots, n$	the weight capacity of container vehicle t
X_{ij}^t	$= 1$ if vehicle t travels from node i to j; $= 0$ otherwise; $(i, j = 0, 1, \cdots, N+1;\ t = 1, \cdots, n)$
x_{k1}, y_{k1}, z_{k1}	the smallest x, y, and z coordinates of the box k in the vehicle.
x_{k2}, y_{k2}, z_{k2}	the largest x, y, and z coordinates of the box k in the vehicle
x_{t1}, y_{t1}, z_{t1}	the smallest x, y, and z coordinates of the container of the vehicle t (the origin of the coordinate system)
x_{t2}, y_{t2}, z_{t2}	the largest x, y, and z coordinates of the container of the vehicle t

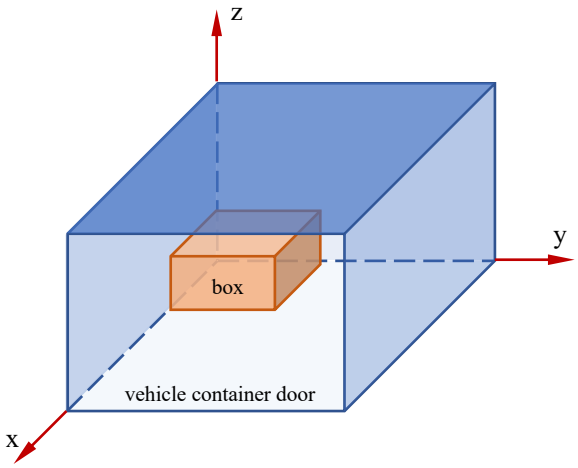


Figure 2.1: Illustration of vehicle container coordinate system.

2.1.2 Mathematical Formulation

The following mathematical formulation is constructed based on [41] and [66]. There are two objectives relevant to both routing and packing aspects of the problem.

The first objective:

$$min\ f_1 = 1 - \frac{\sum_{t=1}^{n} loading\_rate_t}{n} \text{ or } min\ f_1 = n \tag{2.1}$$

In Eq. (2.1):

$$loading\_rate_t = \max\left(v\_rate_t, w\_rate_t\right) \tag{2.1a}$$

$$v\_rate_t = \frac{\sum_{i=1}^{N} \sum_{k \in A_i^t} v_k}{V_t} \tag{2.1b}$$

$$w\_rate_t = \frac{\sum_{i=1}^{N} \sum_{k \in A_i^t} q_k}{Q_t} \tag{2.1c}$$

n : the number of vehicles used.

The second one:

$$min\ f_2 = \sum_{t=1}^{n} \sum_{i=0}^{N+1} \sum_{j=0}^{N+1} d_{ij} X_{ij}^t \tag{2.2}$$

Objective f_1 focuses on maximizing the average vehicle loading rate or minimizing the number of vehicle used, while objective f_2 aims to reduce the total travel distance (ttd) across all vehicles.

The constraints are as follows.

$$\sum_{j=1}^{N} X_{0j}^t = 1,\ t = 1, \ldots, n \tag{2.3}$$

$$\sum_{i=1}^{N} X_{i(N+1)}^t = 1,\ t = 1, \ldots, n \tag{2.4}$$

$$\sum_{i=0}^{N} X_{ik}^t = \sum_{j=1}^{N+1} X_{kj}^t, k = 1, \ldots, N;\ t = 1, \ldots, n \tag{2.5}$$

$$\sum_{i=0; i \neq j}^{N} X_{ij}^t \leq 1,\ j = 1, \ldots, N; t = 1, \ldots, n \tag{2.6}$$

$$\delta_1 \left(x_{k2} - x_{l1}\right) \left(x_{l2} - x_{k1}\right) \leq 0,\ k, l \in A_i^t;\ i = 1, \ldots, N;\ t = 1, \ldots, n \tag{2.7}$$

$$\delta_2 \left(y_{k2} - y_{l1}\right) \left(y_{l2} - y_{k1}\right) \leq 0,\ k, l \in A_i^t;\ i = 1, \ldots, N;\ t = 1, \ldots, n \tag{2.8}$$

$$\delta_3 \left(z_{k2} - z_{l1}\right) \left(z_{l2} - z_{k1}\right) \leq 0,\ k, l \in A_i^t;\ i = 1, \ldots, N;\ t = 1, \ldots, n \tag{2.9}$$

$$\delta_1, \delta_2, \delta_3 \in \{0, 1\},\ \delta_1 + \delta_2 + \delta_3 \geq 1 \tag{2.10}$$

$$\bigcup_{t=1,\ldots,n} A_i^t = K_i,\ i = 1, \ldots, N \tag{2.11}$$

$$\bigcap_{t=1,\ldots,n} A_i^t = \emptyset,\ i = 1, \ldots, N \tag{2.12}$$

$$\begin{aligned} &\sum_{b \in B_k} [\min\{x_{b2} - x_{k1}, x_{k2} - x_{b1}\} \cdot \min\{y_{b2} - y_{k1}, y_{k2} - y_{b1}\}] \geq p \cdot w_k l_k \\ &k \in A_i^t,\ t = 1, \ldots, n;\ i = 1, \ldots, N; \\ &B_k : \text{the set of boxes under and touching the box } k \text{ in vehicle } t \end{aligned} \tag{2.13}$$

$$\sum_{k \in A_i^t} q_k \leq Q_t,\ i = 1, \ldots N;\ t = 1, \ldots, n \tag{2.14}$$

$$x_{k1} \geq x_{t1},\ k \in A_i^t; i = 1, \ldots, N; t = 1, \ldots, n \tag{2.15}$$

$$y_{k1} \geq y_{t1},\ k \in A_i^t; i = 1, \ldots, N; t = 1, \ldots, n \tag{2.16}$$

$$z_{k1} \geq z_{t1},\ k \in A_i^t; i = 1, \ldots, N; t = 1, \ldots, n \tag{2.17}$$

$$x_{k2} \leq x_{t2},\ k \in A_i^t; i = 1, \ldots, N; t = 1, \ldots, n \tag{2.18}$$

$$y_{k2} \leq y_{t2},\ k \in A_i^t; i = 1, \ldots, N; t = 1, \ldots, n \tag{2.19}$$

$$z_{k2} \leq z_{t2},\ k \in A_i^t; i = 1, \ldots, N; t = 1, \ldots, n \tag{2.20}$$

$$I_k > I_l, \quad \text{if Eq. (2.22, 2.23, 2.25) or (2.23, 2.24, 2.26) hold simultaneously.} \tag{2.21}$$

In Eq. (2.21):

$$k \in A_i^t;\ l \in A_j^t;\ i, j = 1, \ldots, N;\ t = 1, \ldots, n \tag{2.21a}$$

$$I_k, I_l : \text{loading order of box } k \text{ and } l \text{ in vehicle } R_t \tag{2.21b}$$

$$(x_{k2} - x_{l1})\,(x_{l2} - x_{k1}) > 0 \tag{2.22}$$

$$(y_{k2} - y_{l1})\,(y_{l2} - y_{k1}) > 0 \tag{2.23}$$

$$(z_{k2} - z_{l1})\,(z_{l2} - z_{k1}) > 0 \tag{2.24}$$

$$z_{k1} \geq z_{l2} \tag{2.25}$$

$$x_{k1} \geq x_{l2} \tag{2.26}$$

The constraints of 3L-SDVRP primarily focus on two aspects:

- Vehicle Routing Constraints

- *Validity and Flow Conservation:* Equations (2.3)–(2.5) define valid route formation. Each route initiates at the starting node and terminates at the ending node. Additionally, flow conservation must be maintained, meaning that the number of vehicles entering and exiting a node must be equal.

- *Single-Visiting:* Equation (2.6) enforces a single-visit requirement, ensuring that each customer is visited by the same vehicle only once.

- 3D Loading Constraints

  - *Non-Overlapping:* Equations (2.7)–(2.10) specify that boxes within the same vehicle must not overlap in any dimension.

  - *Non-Splitting:* Equations (2.11)–(2.12) dictate that each box can be loaded into only one vehicle.

  - *Supporting Stability:* Equation (2.13) is the full supporting stability constraint in which B_k represents the set of all boxes that are under box k and in contact with the bottom surface of k. Thus, constraint (2.13) requires that the supporting area must be greater than or equal to the percentage p of the bottom area of the box ($p = 1.0$ in our research).

  - *Weight Capacity:* Equation (2.14) restricts the total weight of loaded boxes to the vehicle's weight capacity.

  - *Size Constraint:* Equations (2.15)–(2.20) ensure that boxes must be contained within the vehicle's container without exceeding the container walls or doors.

  - *Last-In-First-Out (LIFO):* Equations (2.21)–(2.26) specify that boxes loaded later must be unloaded first, once a vehicle departs from a node.

2.2 Solution Approaches to 3L-SDVRP

Given the high complexity of 3L-SDVRP, exact algorithms are ineffective, thus intelligent optimization methods, particularly meta-heuristic algorithms, emerge as more appropriate and effective methods for such intricate challenges. In studies of 3L-SDVRP, the emphasis is on route searching with a feasible 3D loading plan for each route as a fundamental constraint.

2.2.1 Local vs. Global Search Approaches

Based on the range and strategy of their exploration within the solution space, algorithms for the 3L-SDVRP can be categorized into global search-based methods and local search-based methods.

Local search approaches for solving 3L-SDVRP encompass classic local search ([10] [66]), tabu search ([14] [107]), and simulated annealing ([13]). For global search, MOEA/D ([51]), estimation of distribution algorithms ([50]), and genetic algorithms ([65] [70]) are employed. Memetic algorithms [52] [89], although not yet applied to solving the 3L-SDVRP, represent another class of meta-heuristic algorithms that incorporate local search strategies into global search algorithms.

Local search-based algorithms perform well in exploitation by intensively searching around known good solutions. However, they risk entrapment in local optima, particularly in complex solution spaces. Global search-based algorithms are adept at exploration, covering a wide area of the solution space and potentially avoiding local optima. However, they usually encounter challenges such as higher computational costs and slower convergence rates, especially in the context of complex and large-scale problems. Memetic algorithms are known for their effective balance between exploration and exploitation, which is effective for addressing a wide range of combinatorial optimization problems (COPs). However, these algorithms often demand

more computational resources than global search-based algorithms and require the design of tailored local search strategies.

2.2.2 Multi-Objective vs. Single-objective Approaches

Additionally, since 3L-SDVRP inherently involves two objectives, some research treats it as a multi-objective problem. Moural [65] tackled 3L-SDVRP from a multi-objective perspective using the Multi-Objective Genetic Algorihtm (MOGA) but relaxed some vital constraints, potentially limiting the solution's relevance to real world scenarios. Recently, Liu et al. [51] proposed the Multi-objective Evolutionary Algorithm based on Decomposition by Offline Machine learning (MOEA/D-OM), which is the latest method for solving the multi-objective 3L-SDVRP. It leverages a pre-trained machine learning model to assess packing feasibility for routes, discarding infeasible ones. This approach improve runtime efficiency by avoiding unnecessary 3D packing calculations on infeasible solutions. However, its success largely depends on generating many packing infeasible solutions during the search. The use of a pre-trained model for packing evaluation could also limit the algorithm's adaptability and performance across diverse problem instances. Moreover, the authors recognize that the method is limited in its ability to handle larger-scale problems effectively.

Although the 3L-SDVRP has two objectives, in practice, minimizing the number of vehicles is considered more crucial than reducing the total travel distance (ttd) [10]. This prioritization is due to the substantially greater expenses associated with obtaining and maintaining additional vehicles, as well as hiring more drivers, compared to the additional travel costs. Consequently, it is conventionally treated as a single-objective problem in most literature [10] [13] [14] [50] [66] [76] [107], where the primary objective is the optimization of the number of vehicles used, with the minimization of the ttd as a secondary objective. SDVRLH2 is the state-of-the-art single-objective method for 3L-SDVRP. It consists two stages: the determination of

the packing arrangement for each node and the implementation of local search for the routing process. SDVRLH2 is renowned for its exceptional performance across diverse problem instances, underscoring its leadership in the field. However, it has limitations. The packing algorithm, which packs two boxes at a time, generates only two sub-spaces, potentially wasting space and increasing vehicle usage. The routing component relies on traditional search operators without leveraging problem-specific heuristics, thus reducing search efficiency. Additionally, the outer loop, which varies the maximal admissible splitting costs, incurs significant computational cost. Finally, the post-optimization phase, which implements 3-opt, is computationally expensive.

2.3 Interactive Routing-Packing Strategies

Fundamentally, any 3L-SDVRP solution must address both routing decisions (determining which nodes each vehicle should visit) and packing decisions (designing the 3D packing plan for each vehicle). These decisions substantially influence the final solution quality, making the interactions between routing and packing during the solution process crucial for algorithm performance. We refer to these interactions as "interactive routing-packing strategy". The term "interactive" denotes the mutual influence of routing and packing decisions: routing decisions dictate which nodes' boxes a vehicle will load, while packing decisions directly affect the number of boxes that can be loaded within the vehicle's limited capacity. Prevailing interactive routing-packing strategies are mainly divided into two paradigms: routing-first-packing-second (R1P2), and packing-first-routing-second (P1R2).

2.3.1 Routing-First-Packing-Second (R1P2)

The R1P2 strategy is widely used to tackle VRPs with 3D loading constraints [21] [46] [47] [57] [59] [60]. The R1P2 strategy seamlessly incorporates the packing process into

the routing process. For each route, it generates a unique loading plan. The algorithm then iteratively optimizes these routes to enhance solution quality. Importantly, any changes made to a route will naturally modify the set of boxes to be loaded, demanding immediate adjustments in the corresponding loading plan. In brief, the R1P2 strategy solves multiple 3D packing problems as it optimizes routes. When obtaining loading arrangements, meta-heuristic algorithms [14] [21] [46] [47] [60] [77] [120], local search techniques [14] [65] [66], or a bundle of heuristic algorithms [8] [13] [42] [50] [57] [59] [69] [76] [91] are generally utilized.

2.3.2 Packing-First-Routing-Second (P1R2)

The P1R2 strategy, first introduced by [9], tackles packing and routing as two independent phases. During the packing stage, each node is evaluated as an isolated 3D packing problem (3DPP), with all packing decisions being made before the routing process. The loading plan remains transparent to the routing stage, which means the algorithm does not delve into the specifics of loading. Instead, it utilizes essential information such as the total space occupied by boxes at each node, which affects the routing decisions. This strategy effectively simplifies the problem, essentially transforming it into a VRP without the complexities of 3D loading.

A vertical layer approach are commonly adopted in P1R2. Introduced by [27] and later extended in the context of the 3DPP [7] [12] [25] [56], a vertical layer is a rectangular space occupied by multiple boxes. As illustrated in Fig. 2.2, each layer's width and height dimensions correspond to the vehicle's y and z axes, respectively. The depth of the layer, measured along the x-axis, is established by the first box loaded into the layer. In the P1R2 strategy, each node is treated as a separate 3DPP. Solving this problem results in boxes at a given node forming one or multiple layers with varying depths. During the routing phase, a layer is considered the smallest loading unit; its internal configuration of boxes remains invariant. Consequently,

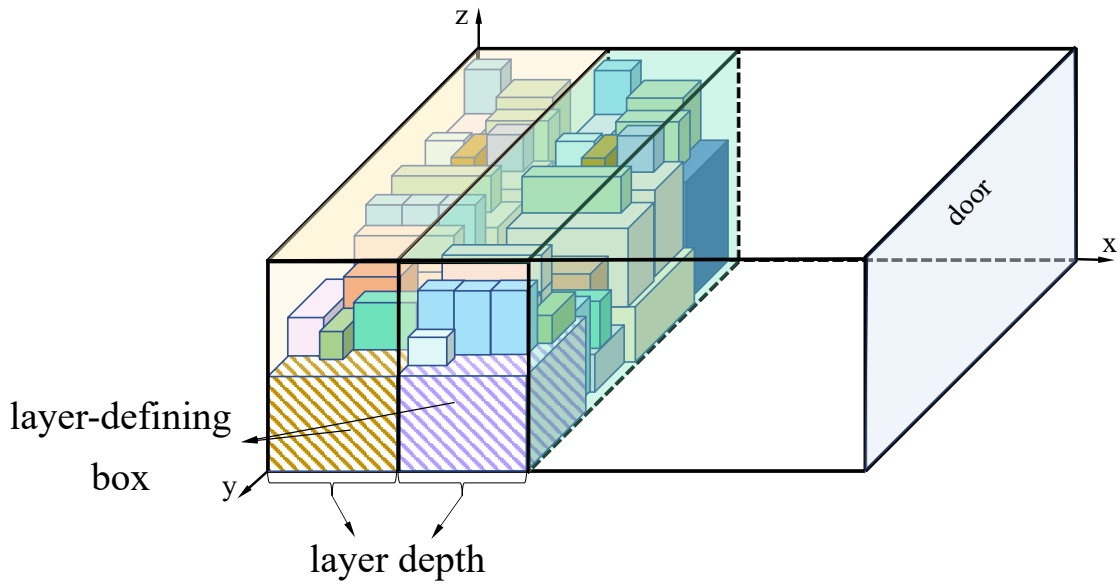


Figure 2.2: Vertical layers.

distinct layers from the same node can be loaded onto different vehicles.

2.3.3 P1R2 with 2C-SP

The SDVRLH2 algorithm, as proposed by [10], is currently the most advanced method for tackling 3L-SDVRP. Utilizing the P1R2 strategy, this algorithm introduces an inventive packing pattern termed the two-customer segment pattern (2C-SP) based on vertical layers.

In the SDVRLH2, each node, represented as n_1, seeks to form 2C-SPs with the closest 30% of its neighboring nodes in terms of physical distance. For any neighbouring node n_2, a "mixed layer" is formed by choosing one vertical layer from each node according to a set rule. Combined with the remaining vertical layers from n_1 and n_2, this mixed layer results in a unique 2C-SP for the node pair (n_1, n_2).

To quantify the efficiency of this pattern, a "saving value", represented as δ, is calculated. This value measures the space saved by a 2C-SP. Each node n_1 will attempt

to establish 2C-SPs with its neighbours through two different procedures and determine the corresponding δ values. The 2C-SP that results in the highest δ value—meaning it saves the most space—is selected as the preferred option. For a detailed introduction on 2C-SP, please refer to [10].

In the SDVRLH2 algorithm, vertical layers and 2C-SPs are established prior to routing, aligning with the P1R2 strategy. The goal of combining two nodes into a 2C-SP is to minimize the space taken up by boxes, which in turn reduces the number of vehicles needed.

2.4 Search Operators and Step Sizes

Meta-heuristic algorithms are a class of iterative search algorithms based on a generate-and-test strategy, typically involving a repetitive process of continuously generating, evaluating, and improving solutions until certain stopping criteria are met [106]. Search operators are one of the core components in these algorithms, determining how the algorithm explores the solution space and improves the current solution. In each iteration, search operators are used to generate new solutions, thereby exploring the solution space. Search operators determine the search direction and the search step size during the search process. Their selection and design process are not only related to the problem being solved but also to the representation, and they directly impact the performance of the optimization algorithm, thus playing a crucial role.

2.4.1 Commonly Used Search Operators for 3L-SDVRP

Tab. 2.3 shows the search operators commonly used in the existing literature for solving VRPs with 3D loading constraints. These operators can be divided into two types: intra-sequence and inter-sequence operators. Intra-sequence operators act on a single sequence of nodes, affecting changes within the same sequence, such as reordering,

inverting, or swapping nodes. Inter-sequence operators, on the other hand, act on two different sequences of nodes, typically involving operations such as transferring nodes from one sequence to another or swapping nodes between sequences.

- *Swap Operator:* Exchange the positions of two nodes.
- *Shift Operator:* Move a node to another position.
- *2-opt Operator:* Breaks two edges and reconnects two new edges to alter the node order.
- *3-opt Operator:* Breaks three edges in a node sequence and reconnects three new edges.
- *Move & Rotate Block:* This operator groups identical boxes into blocks for a block sequence representation. It selects and inserts blocks between sequences.
- *Split Operator:* Divides a node sequence into two non-empty sequences.
- *Best Cost Route Crossover:* Selects two node sequences, inserts two consecutive nodes from each into the other.
- *1-point Crossover:* Randomly divides two node sequences at a selected position and recombines them.
- *2-point Crossover:* Randomly selects two points in two node sequences and exchanges the segments.

There are also other search operators, such as the Partially Mapped Crossover (PMX) [38] [93] [99], Order Crossover (OX) [4] [74], Sequence Based Crossover (SBX) [73] [89], and Route Based Crossover (RBX) [73], etc. Although these operators have not been used in existing research to solve VRPs with 3D loading constraints, they serve as general search operators and can be used to solve various Combinatorial Optimization Problems (COPs).

Table 2.3: Commonly used search operators for VRPs with 3D loading constraints.

Search operators	Type		Literature
	Intra-sequence	Inter-sequence	
Swap	√	√	[10] Bortfeldt & Yi (2020), [14] Chen et al. (2020), [107] Yi & Bortfeldt (2018), [13] Ceschia et al. (2013), [47] Koch et al. (2020), [77] Reil et al. (2018), [120] Zhang et al. (2015), [103] Wei et al. (2014), [48] Lacomme et al. (2013), [110] Zachariadis et al. (2013), [6] Bortfeldt (2012), [121] Zhu et al. (2012), [23] Fuellerer et al. (2010), [102] Wang et al. (2010), [92] Tarantilis et al. (2009), [26] Gendreau et al. (2006)
Shift	√	√	[10] Bortfeldt & Yi (2020), [14] Chen et al. (2020), [107] Yi & Bortfeldt (2018), [100] Turky et al. (2017), [13] Ceschia et al. (2013), [42] Junqueira & Morabito (2015), [69] Pace et al. (2015), [120] Zhang et al. (2015), [103] Wei et al. (2014), [6] Bortfeldt (2012), [109] Zachariadis et al. (2012), [121] Zhu et al. (2012), [102] Wang et al. (2010), [92] Tarantilis et al. (2009), [26] Gendreau et al. (2006), [23] Fuellerer et al. (2010)
2-opt	√	√	[14] Chen et al. (2020), [66] Moura & Oliveira (2009), [77] Reil et al. (2018), [69] Pace et al. (2015), [120] Zhang et al. (2015), [103] Wei et al. (2014), [48] Lacomme et al. (2013), [121] Zhu et al. (2012), [102] Wang et al. (2010), [92] Tarantilis et al. (2009)
3-opt	√		[77] Reil et al. (2018), [9] Bortfeldt & Homberger (2013)
Move & rotate block		√	[13] Ceschia et al. (2013)
Best cost route crossover		√	[65] Moura (2008), [34] Hanshar & Ombuki (2007), [67] Ombuki et al. (2002), [68] Ombuki et al. (2006)
1-point crossover		√	[14] Chen et al. (2020), [120] Zhang et al. (2015), [103] Wei et al. (2014), [48] Lacomme et al. (2013), [109] Zachariadis et al. (2012), [102] Wang et al (2010), [92] Tarantilis et al. (2009)
2-point crossover		√	[84] Ruan et al. (2013), [62] Miao et al. (2012)

Note: Intra-sequence refers to an operator acting on a single sequence of nodes; inter-sequence refers to an operator acting on two sequences of nodes.

Although general operators are straightforward to comprehend, easy to implement, and applicable across various COPs, their lack of specialized domain knowledge guidance significantly limits their effectiveness in the search process. This limitation becomes especially evident in problems with complex constraints and large scales. Therefore, in the context of complex VRPs such as 3L-SDVRP discussed in this thesis, general operators are inadequate for addressing these challenges efficiently and effectively.

Some research has placed an emphasis on the integration of domain knowledge into search operators, aiming to enhance search efficiency and yield solutions of superior quality. An example of this advancement is the Merge-Split (MS) operator, developed by Tang et al. [89] for the Capacitated Arc Routing Problem (CARP). This operator comprises two key components: Merge and Split. The Merge component operates by randomly selecting a specified number of routes, denoted as p, and combining their tasks into an unordered list. Subsequently, the Split component employs a Path Scanning (PS) heuristic [28] along with five distinct rules to generate a series of ordered lists. These lists are then segmented into routes utilizing Ulusoy' s splitting procedure [101]. This operator effectively integrates domain knowledge via the PS heuristic and Ulusoy's method and embodies the concept of a large step size [104] in route optimization, through its innovative approach of merging and dividing routes. Lan et al. [49] introduced two distinct region-focused operators for the Multidepot Multidisposal-facility Multitrip Capacitated Vehicle Routing Problem (M3CVRP). These operators begin by randomly selecting a route, denoted as R_1, and a node in R_1, referred to as k. Within a defined radius ρ around node k, a node c not part of R_1 is identified. The subsequent operation can occur in one of two ways: either a Region-Focused Single-Point Swap (RFSPS), which involves exchanging node k with node c, or a Region-Focused Segment Swap (RFSS), which requires the substitution of the entire segment following nodes k and c.

Search operators that integrate domain knowledge, like these, are generally tailored to

specific problems or their unique characteristics. This specificity demands a case-by-case analysis, leading to the design of distinct search operators based on the domain knowledge relevant to each problem.

2.4.2 Search Step Sizes

In meta-heuristic algorithms, the search step size denotes the magnitude of changes applied to solutions in each iteration, influencing the algorithm's capacity for effective exploration and exploitation of the solution space. Larger step sizes enable more extensive exploration of the solution space, but risk missing the optimal solution, whereas smaller step sizes concentrate on thoroughly exploiting the best solution found within a specific region. Additionally, the search step size directly impacts the convergence speed of the algorithm.

When tackling 3L-SDVRP, search methods like classic local search [10] [66], tabu search[14] [107], and simulated annealing [13] offer efficiency with smaller step sizes, but may get stuck in local optimal solutions. In contrast, evolutionary algorithms [51] [65] [70] and estimation of distribution algorithms [50], using larger step sizes, explore more broadly but with higher computational costs and slower convergence.

Researchers have actively explored methods to combine or balance small and large search step sizes. For example, Yao employed a larger search step size in a local search-based algorithm—Simulated Annealing (SA), and theoretically showed that the SA with a larger neighborhood size are more effective than SA with a smaller one [104]. Shaw developed the Large Neighborhood Search (LNS) algorithm for simple VRPs, known for its substantial solution modifications (i.e., large search step size) [86]. This algorithm has been applied to various VRP variants [19] [85]. Subsequent advancements in local search-based methods have seen the introduction of dynamic adjustments in neighborhood sizes [63] [105], proving beneficial in a range of VRPs [11] [108]. Building upon the principles of LNS, Ropke and Pisinger developed an

adaptive large neighborhood search algorithm, initially for addressing the Pickup and Delivery Problem with Time Windows (PDPTW) [79]. This methodology has been further extended to apply to other VRP variations, such as VRP with Transshipment Facilities (VRPTF) [22], VRP with Multiple routes and Time Windows (VRPMTW) [2], and VRP with Cross-Docking (VRPCD) [32], illustrating its effectiveness in diverse routing challenges. The Memetic Algorithm (MA) [64], which effectively merges local and global search methods and incorporates both small and large step sizes, is recognized for its high time complexity and requires the design of tailored local search strategies. Nevertheless, the MA has been extensively applied to a diverse array of COPs, including the Capacitated Arc Routing Problem (CARP) [89], a variety of VRP variants [31] [58], the Job Scheduling Problem (JSP) [24] [30] [61], and the Graph Coloring Problem (GCP) [17] [55], among others. However, directly applying these methods to 3L-SDVRP is ineffective as they do not incorporate domain knowledge which is crucial for addressing such a complex problem. Thus, achieving a balanced trade-off between large and small step sizes remains a significant challenge.

2.5 Discussion

The high complexity of 3L-SDVRP presents significant challenges for solving it using exact algorithms. Consequently, intelligent optimization algorithms, particularly meta-heuristic algorithms, are recognized as effective and widely-used approaches for addressing complex problems like 3L-SDVRP. These methods operate on a generate-and-test iterative strategy [106], where each iteration generates a new set of potential solutions from the existing ones using various search operators, aiming to find improved solutions. This process is repeated iteratively to approximate an optimal solution. Several key factors influence the performance of these algorithms:

- A critical factor is the choice of search operators, such as crossover operators,

mutation operators, and neighborhood operators. These operators are core components of meta-heuristic algorithms, determining how the algorithm navigates the search space. Their importance lies in maintaining solution diversity, effectively balancing exploration and exploitation, and adapting to specific problem characteristics. By designing and utilizing effective search operators, meta-heuristic algorithms can more efficiently find high-quality solutions.

- The balance between exploration and exploitation is crucial to the success of these algorithms. Exploration requires conducting a broad search within the solution space to discover new potential solutions, thereby supporting innovation and diversity. Conversely, exploitation focuses on intensifying the search around already identified effective solutions, aiming to refine them further. The performance of different algorithms depends significantly on their emphasis on either exploration or exploitation.

- In meta-heuristic algorithms, the search step size refers to the extent of change in solutions during each search step. The selection and adjustment of search step size are crucial to the algorithm' s performance and efficiency. It directly impacts the algorithm' s ability to explore and exploit, its convergence speed, adaptability to different problem characteristics, and robustness. Effective step size selection and adjustment strategies can significantly enhance the performance of meta-heuristic algorithms, enabling them to more effectively find high-quality solutions.

In Section 2.2, we summarize the different solution approaches for solving 3L-SDVRP, focusing on local/global search-based algorithms and single/multi-objective algorithms. Current research indicates that the exploration of global search-based multi-objective optimization algorithms for solving 3L-SDVRP remains insufficient. Additionally, combining routing and packing—both NP-hard problems—3L-SDVRP's solution process is notably time-consuming [6] [23] [26] [46] [47] [50] [78]. In real-world scenarios,

computational resources are often limited, posing a significant challenge in designing effective optimization algorithms that improve solution quality under these strict limitations. In Chapter 3, we tackled 3L-SDVRP as a multi-objective problem, focusing on improving algorithmic performance through effective balance between exploration and exploitation, especially under tight computational resource limitations (i.e., the number of fitness evaluations).

Moreover, in real-world scenarios, the number of vehicles is often more critical than the travel distance. This is because the costs associated with maintaining additional vehicles and hiring extra drivers are typically higher than those incurred from an increase in travel distance. Given these insights, we explored the local search-based approach to solve 3L-SDVRP. In alignment with existing research [10] [76], we addressed 3L-SDVRP as a single-objective problem, prioritizing vehicle reduction as the primary objective and total travel distance as a secondary objective. Our research in Chapter 4 aimed at proposing a more efficient method to solve 3L-SDVRP based on the state-of-the-art algorithm SDVRLH2 [10].

In Section 2.3, we review the existing interactive routing-packing strategies. Each strategy has its pros and cons. R1P2 integrates packing into routing, allowing for adjustments of the packing plan as the route changes. While this offers more flexibility and may require fewer vehicles, it also makes computation more complex and time-consuming. On the other hand, the P1R2 approach treats each node as a separate packing problem and establishes the packing plan before the routing process, which avoids having to solve the packing problem during routing repeatedly. This speeds up the process but may limit flexibility in minimizing vehicle use. In contrast to the P1R2 strategy where each node is independently loaded, the 2C-SP involves loading two nodes together, potentially saving vehicle space. However, as noted in [10], in creating 2C-SP, a node only considers its closest 30% neighboring nodes. Furthermore, these 2C-SPs are fixed prior to routing and do not update during the routing process. Therefore, there may exist opportunities for more effective interactive routing-packing

strategies. Despite the importance of interactive routing-packing strategies in solving 3L-SDVRP, there's a lack of in-depth comparison and assessment of these various approaches. In Chapter 5, our research provides a detailed experimental analysis that compares the effectiveness of the P1R2 and R1P2 strategies and investigates the impact of using the 2C-SP pattern. Based on the insights gained from this analysis, we introduce an adaptive interactive routing-packing strategy which dynamically adjusts loading decisions during the routing process.

In Section 2.4, we review the search operators used for solving 3L-SDVRP and discuss the efforts made to adjust search step size within the algorithms. Our literature review reveals that existing operators are becoming increasingly inadequate for effectively and efficiently solving complex VRPs. In response to this, incorporating domain knowledge into optimization algorithms through search operators can significantly improve algorithm performance and applicability. This approach is particularly effective when it focuses on the specific characteristics of the problem at hand. Besides, these search operators play a pivotal role in determining the algorithm's search step size. In optimization search algorithms, this step size is a vital factor that influences the balance between exploration and exploitation. It also significantly affects the algorithm's convergence speed. This, in turn, influences both the computational resources required and the quality of the solutions produced. Therefore, the selection of an appropriate search step size is a matter of utmost importance. Chapter 6 describes our proposed Adaptive Knowledge-guided Insertion (AKI) operator and Adaptive Knowledge-guided Search (AKS) algorithm. Our AKI operator not only introduces domain knowledge but also has a larger search step size compared to general operators. In AKS, traditional operators conduct detailed searches with small step sizes, while the AKI operator explores new areas with a larger step size. This approach balances exploration and exploitation without increasing the number of fitness evaluations (FEs), thereby improving efficiency and solution quality.

Chapter 3

A Multi-Objective Approach to 3L-SDVRP*

This chapter addresses the 3L-SDVRP as a multi-objective problem and introduces a Hierarchical Neighborhood Filtering (HNF) mutation operator, along with a Pareto-based Evolutionary Algorithm with Concurrent crossover and Hierarchical Neighborhood Filtering mutation (PEAC-HNF).

We propose the HNF operator with three key features: leveraging multiple neighborhood structures to generate offspring, thereby enhancing solution diversity during the search process; operating hierarchically by prioritizing individuals with high nondomination ranks, ensuring that the search focuses on superior solutions; and employing a filtering process to eliminate unnecessary individuals, thus reducing redundant fitness evaluations. Our PEAC-HNF algorithm combines the local search capabilities of the HNF operator with the global search capabilities of the Evolutionary Algorithm (EA) framework, effectively balancing exploitation and exploration. The organization of

*This chapter is partially based on a paper published at the 2024 Genetic and Evolutionary Computation Conference (GECCO' 24) [119] and a paper published at IEEE Transaction on Emerging Topics on Computational Intelligence (TETCI) [118].

this chapter is as follows. Section 3.1 discusses the current research challenges and our motivations. Section 3.2 details our proposed HNF mutation and PEAC-HNF algorithm. Section 3.3 compares our algorithm with state-of-the-art methods and offers further analysis. Section 3.4 concludes the chapter.

3.1 Introduction

In Chapter 2, we identified that 3L-SDVRP has two primary objectives: maximizing the average vehicle loading rate (or minimizing the number of vehicles used) and minimizing the total travel distance. The two objectives are inherently in conflict with each other. On one hand, due to the varying 3D sizes of boxes both between and within nodes, a vehicle may not be able to load all the boxes at a particular node, yet it might still have enough capacity to load boxes from other nodes. In this situation, strictly pursuing a high load rate for each vehicle requires loading as many boxes as possible. This approach may lead to a vehicle visiting more nodes to fully utilize its capacity, thereby increasing the total travel distance. On the other hand, if the aim is to minimize the total travel distance, vehicles might refrain from loading additional boxes to avoid visiting more nodes, resulting in unused capacities and a lower load rate. In Section 3.3.2, we compare our proposed PEAC-HNF algorithm with two single-objective methods, each focusing on one of the two objectives. The experimental results in Figs. 8 and 9 indicate that solutions obtained by single-objective methods are highly imbalanced in terms of quality across the two objectives; they perform well in the objective they prioritize but poorly in the other, illustrating the conflicting nature of these objectives. This complex relationship presents a significant challenge for multi-objective optimization, as it requires achieving solutions that are well-balanced across both objectives.

Current research on utilizing multi-objective optimization algorithms to solve 3L-SDVRP remains insufficient. Liu et al. [51] introduced a Multi-objective Evolutionary

Algorithm based on Decomposition by Offline Machine learning (MOEA/D-OM) to address the multi-objective 3L-SDVRP. This state-of-the-art multi-objective method employs pre-trained machine learning models to predict the packing feasibility of each route, discarding those predicted as infeasible. Consequently, this approach reduces the algorithm's runtime by avoiding the 3D packing process for infeasible solutions, achieving computational savings. However, this comes at the cost of generating a large number of infeasible solutions during the search process, specifically packing infeasible routes. Moreover, the reliance on pre-trained machine learning models presents a generalization challenge when solving different problem instances. Additionally, their proposed method struggles with larger problem scales, and the solution quality remains inferior to that obtained by the SDVRLH2 algorithm proposed in [10]. Moura [65] developed a Multi-Objective Genetic Algorithm (MOGA) to solve 3L-SDVRP. However, their study relaxed some critical constraints of the problem, thereby reducing the problem-solving difficulty and its relevance to real-world scenarios, which diminishes the applicability of the obtained solutions in practical problems.

The current research faces the following challenges:

- The inherent complexity of 3L-SDVRP, which involves both routing and loading constraints, makes efficient searching more difficult and computationally expensive. This complexity poses a challenge to the real-world demand for high-quality solutions with minimal computational resources, especially in time-sensitive logistics and supply chain applications.

- The conflicting relationship between the two objectives of 3L-SDVRP makes it hard to achieve high-quality solutions for both objectives simultaneously. Overemphasizing one objective may lead to poor performance on the other (see experimental results of single-objective methods on Section 3.3.2), which is undesirable in real-world scenarios where both objectives are important.

To address these challenges, we develop a Hierarchical Neighborhood Filtering (HNF) mutation operator, characterized by:

- Using diverse neighborhood structures like swap, 2-opt, and 3-opt to create a wide range of offspring from a single parent, thus improving solution diversity and algorithm exploitation capability.

- It adopts a hierarchical approach to mutation, prioritizing individuals with higher nondomination ranks for more focused search on promising candidates.

- The offspring undergo a filtering process, where some individuals are removed if they meet specific criteria, enhancing search efficiency and reducing unnecessary fitness evaluations.

By incorporating the HNF mutation into the EA framework, we have developed a novel Pareto-based Evolutionary Algorithm with Concurrent crossover and Hierarchical Neighborhood Filtering mutation (PEAC-HNF) for 3L-SDVRP. The HNF mutation enhances PEAC-HNF's exploitation capabilities, complementing the EA's inherent exploration strength, thereby achieving a better balance between exploration and exploitation. Additionally, PEAC-HNF executes crossover and mutation processes concurrently, allowing an individual to undergo either or both processes in parallel, but not in a sequential manner. The proposed PEAC-HNF algorithm demonstrates a better balance between exploration and exploitation under constrained computational resources, significantly enhancing overall performance. Furthermore, it provides a robust framework for multi-objective optimization in the context of the 3L-SDVRP. PEAC-HNF has been evaluated against baselines and the state-of-the-art multi-objective algorithm for 3L-SDVRP, e.g., MOEA/D-OM [51] and MOGA [65], demonstrating its effectiveness. Further experimental studies have validated the crucial role of the HNF mutation on improving algorithmic performance.

3.2 The PEAC-HNF Algorithm

To provide a clearer description of our proposed method, this section first presents the solution representation in Section 3.2.1. Next, we provide a detailed description of the Hierarchical Neighborhood Filtering (HNF) mutation in 3.2.2. Following this, Section 3.2.3 delve into the framework of the Pareto-based Evolutionary Algorithm with Concurrent crossover and Hierarchical Neighborhood Filtering mutation (PEAC-HNF). Finally, additional details of PEAC-HNF are provided in Section 3.2.4.

3.2.1 Representation and Giant Tour Decoding

This thesis employs the giant tour representation, a concept first introduced by Beasley [4]. This representation has been utilized in numerous variants of the VRPs [1] [10] [18] [20] [38] [44] [48] [65] [66] [74] [88] [103]. For a comprehensive survey on the giant-tour representation of VRP, interested readers can refer to [75]. In essence, a giant tour is a permutation of all nodes, representing a sequence rather than a feasible solution. In the context of 3L-SDVRP, a feasible solution includes both a set of feasible routes and a feasible packing plan for each vehicle. Therefore, a giant tour decoding procedure is needed to decode the giant tour into several feasible routes. The complexity of developing feasible solutions lies in the need to create not only routes but also detailed loading plans for multiple vehicles, such as the loading sequence and spatial arrangement of each box.

Decoding a giant tour means converting it into a feasible solution. The giant tour decoding process follows the method outlined in [10]. In this process, for a given giant tour, a new empty vehicle departs from the starting point and visits each node in the order specified by the giant tour, packing boxes at each node. Specifically, if a vehicle cannot load all boxes at a node, it will load as many as it can without violating any constraints, then proceed directly to the destination instead of visiting

other nodes. Subsequently, another new empty vehicle starts from the starting point, first visiting the last node served by the previous vehicle (if any boxes remain at that node), and then visits the remaining nodes in the giant tour. Split delivery occurs when the boxes from a node are distributed among multiple vehicles. Thus, through the decoding, we derive a feasible solution comprising a set of routes and a feasible packing arrangement for each vehicle from a giant tour.

3.2.2 HNF Mutation

To augment the algorithm's exploitation potential, we designed a novel Hierarchical Neighborhood Filtering (HNF) mutation operator that performs fine-grained local searches around high-quality solutions.

As shown in Algorithm 1, before performing the Hierarchical Neighborhood Filtering (HNF) mutation, nondominated sorting [16] is conducted to assign a nondomination rank to individuals in the parent population POP. The algorithm then performs mutation on individuals with the first nondomination rank, followed by the second, and so on (Algorithm 1 Steps 3-8). This hierarchical approach continues until the number of final offspring obtained by HNF mutation (denoted as O_{HNF}) reaches $P \cdot p_m$ (Steps 2), where P is the population size and p_m is the proportion of fitness evaluations dedicated to O_{HNF}.

For each individual, the HNF mutation utilizes different neighborhood structures, including swap, 2-opt, and 3-opt operators, to generate multiple offspring from a single parent s. Specifically, the swap operator exchanges the positions of two nodes [10] [26]; the 2-opt operator breaks two edges and reconnects them to alter the node order [14] [35]; and the 3-opt operator breaks three edges and reconnects them to create a new sequence of nodes [35] [77]. Thus, for a given parent s, all possible individuals generated by these three neighborhood structures constitute its offspring, denoted as O_s (Step 10).

After obtaining O_s, the individuals within it undergo an immediate filtering process (Step 11). This means not all individuals from O_s will be added to the final mutation offspring O_{HNF}. Individuals in O_s that meet the following criteria will be sampled and added to O_{HNF}: (1) they are different from individuals in the parent population, (2) they are different from individuals sampled from the crossover offspring population, and (3) they are different from individuals already added to O_{HNF} in the current generation. If the number of qualified individuals in O_s exceeds the required amount, individuals will be randomly sampled (Step 15). This procedure repeats until the number of individuals in O_{HNF} reaches $P \cdot p_m$.

Compared to traditional mutations or applying a single mutation operator multiple times on an individual, our HNF mutation (1) creates a broader range of offspring for selection, increasing diversity, (2) focuses the search on superior individuals, thereby improving the algorithm's ability to exploit solutions, and (3) enhances search efficiency by reducing unnecessary fitness evaluations through its filtering process.

3.2.3 Framework of PEAC-HNF Algorithm

Incorporating the HNF mutation into the EA framework, this chapter proposed a new Pareto-based Evolutionary Algorithm with Concurrent execution for crossover and Hierarchical Neighborhood Filtering mutation (PEAC-HNF).

Algorithm 2 details the PEAC-HNF algorithm, and Fig. 3.1 illustrates its workflow, beginning with the creation of an initial population of size P (Step 1). The fitness values of each individual, defined by their total travel distance and average vehicle loading rate, are evaluated. Individuals are then ranked using nondominated sorting [16], assigning each a nondomination rank (Step 2). Considering the computational resource limits, the FEs per generation equals P. The algorithm allocates $p_m \cdot P$ (where $0 < p_m < 1$) FEs for mutations and the remaining FEs, $(1 - p_m) \cdot P$, are used for crossovers. Both crossovers and mutations in PEACH occur concurrently. If

Algorithm 1 HNF Mutation (Key innovations in red boxes.)

Input: POP: population with nondomination rank; p_m: proportion of fitness evaluations dedicated to the HNF mutation

Output: O_{HNF}: final offspring obtained by the HNF mutation

$O_{HNF} \leftarrow \emptyset; rank \leftarrow 1; P \leftarrow |POP|$
while $|O_{HNF}| \neq P \cdot p_m$ **do**
    $S_{rank} \leftarrow$ all individuals whose nondomination rank equals $rank$ in POP
    if S_{rank} is $\emptyset$ **then**
        $rank \leftarrow rank + 1$
        continue
    end if
    $s \leftarrow$ random select an individual from S_{rank}
    $O_s \leftarrow \emptyset$; $POP \leftarrow$ delete s from POP
    $O_s \leftarrow$ get all neighborhood individuals of s by swap, 2-opt, and 3-opt operators and add them to O_s
    $O_s \leftarrow$ filtering O_s and retaining only qualified individuals
    if $|O_s| \leq (P \cdot p_m - |O_{HNF}|)$ **then**
        $O_{HNF} \leftarrow O_{HNF} \cup O_s$
    else
        $O_{HNF} \leftarrow$ select $(P \cdot p_m - |O_{HNF}|)$ individuals from O_s randomly and add to O_{HNF}
    end if
end while
return O_{HNF}

the offspring exceed the FEs allocated to any process, random sampling is utilized to select the offspring for further evaluation. These offspring are then assessed and sorted by nondomination ranks (Step 9). Using a $(\mu+\lambda)$ survival strategy $(\mu = P, \lambda = P)$, the top-performing individuals form the new parent population for the next generation, and the cycle repeats until the termination conditions are met.

Algorithm 2 PEAC-HNF Algorithm (Key innovation in red box.)

Input: P: population size; p_m: proportion of fitness evaluations dedicated to the HNF mutation; G: maximal # of generations

Output: S: final nodominated solutions

1: $g \leftarrow 1$; $POP_g \leftarrow$ initialize population randomly
2: $POP_g \leftarrow$ evaluate POP_g and perform nondominated sorting
3: **while** $g \leq G$ **do**
4: $\quad O_x \leftarrow$ perform crossover on POP_g and get offspring
5: $\quad O_{HNF} \leftarrow$ perform HNF mutation (Algorithm 1) on POP_g and get offspring
6: $\quad O_g \leftarrow O_x \cup O_{HNF}$
7: $\quad O_g \leftarrow$ evaluate O_g
8: $\quad POP_{g+1} \leftarrow O_g \cup POP_g$
9: $\quad POP_{g+1} \leftarrow$ perform nondominated sorting and survival selection on POP_{g+1}
10: $\quad g \leftarrow g + 1$
11: **end while**
12: $S \leftarrow$ the nondominated solutions (i.e., individuals with first nondominition rank) in POP_g
13: **return** S

By integrating the HNF mutation within the EA framework, PEAC-HNF enhances its exploitation capabilities while preserving the natural exploration advantages of evolutionary algorithms, thereby establishing an effective balance between exploration and exploitation. Furthermore, the PEAC-HNF algorithm demonstrates efficacy in identifying Pareto-optimal solutions that effectively balance the trade-offs between

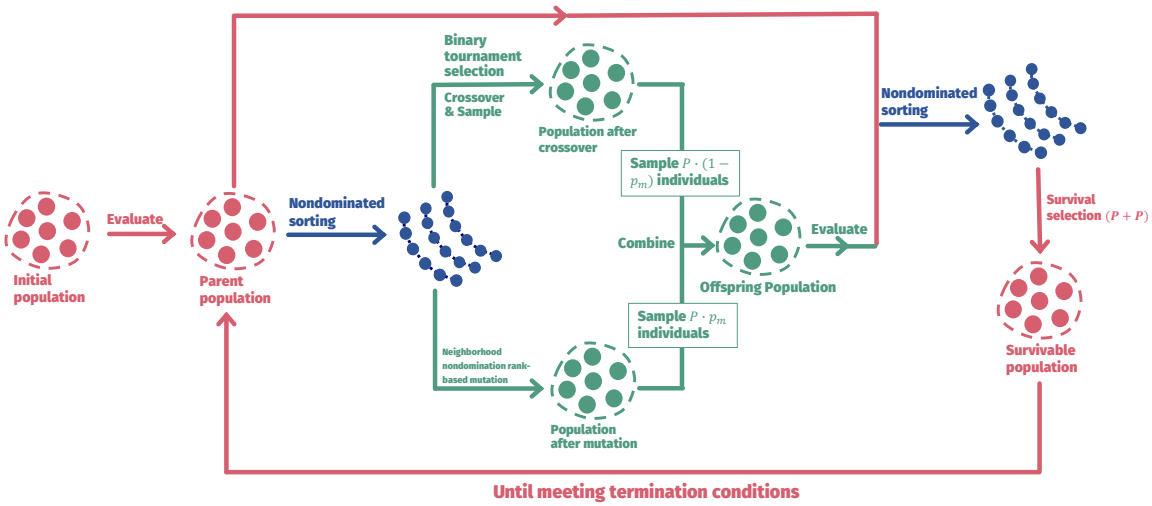


Figure 3.1: Pareto-based Evolutionary Algorithm with Concurrent execution for crossover and Hierarchical Neighborhood Filtering mutation (PEAC-HNF). P is population size. p_m is proportion of FEs allocated to offspring obtained by mutation.

conflicting objectives. This approach yields a diverse set of high-quality alternatives, thereby providing decision-makers with a comprehensive range of options for informed decision-making.

Concurrent Execution of Crossover and Mutation

The term "concurrent" refers to the parallel execution of crossover and mutation processes. This setup allows an individual to undergo either the crossover or mutation processes on their own, or both simultaneously, instead of one after the other.

In this study, executing crossover and mutation processes concurrently is found to be more effective than the traditional sequential method, where crossover is followed by mutation. Considering the constraint of limited FEs, our HNF mutation prefers individuals with higher nondomination ranks. To do this, it is necessary to first perform fitness evaluations and nondominated sorting to assign nondomination ranks. Fig. 3.2 contrasts the concurrent and serial sequence to crossover and mutation.

In our implementation, the # FEs per generation is strictly limited to P. We allocate a proportion p_m of these evaluations to mutation offspring, and the remaining proportion $1 - p_m$ to crossover offspring. Specifically, in each generation:

- $P \times (1 - p_m)$ FEs are used to evaluate offspring generated by crossover.
- $P \times p_m$ FEs are used to evaluate offspring generated by mutation.
- Both crossover and mutation are independently applied to the same parent population. If either operator generates more offspring than its allocated quota, random sampling is performed to ensure the correct ratio between the two types of offspring.

This design is fundamentally different from the conventional serial approach, where mutation is only performed on the offspring generated by crossover ($P \times (1 - p_m)$ individuals). In our concurrent approach, crossover and mutation are executed in parallel on the parent population (P individuals), so that the scope of mutation is extended and solution diversity is enhanced, without increasing the total # FEs per generation. Our experimental results in Section 3.3.3 validate the effectiveness of concurrent sequence.

3.2.4 Other Details of PEAC-HNF Algorithm

Fitness Evaluation

The evaluation method is illustrated in Algorithm 3. We have already known an individual is a giant tour. Through the giant tour decoding procedure, a feasible solution, which contains a set of routes and a feasible packing arrangement of each vehicle, will be obtained from an individual (Algorithm 3 Step 1). Then two objectives can be calculated (Algorithm 3 Step 2). After all individuals' f_1 and f_2 are obtained, the population is sorted based on the nondomination, and each individual is assigned

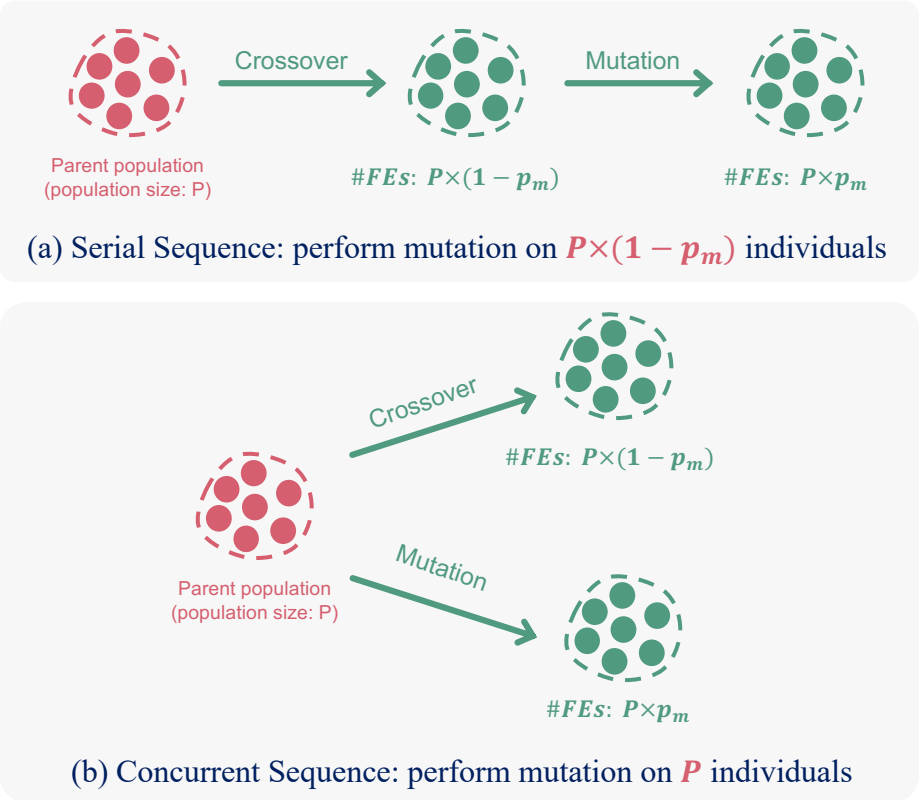


Figure 3.2: The difference between the serial and concurrent sequence of the crossover and mutation. P is $\#FEs$ allocated to each generation (also the population size). p_m is the proportion of FEs allocated to offspring obtained by mutation.

a nondomination rank [16]. The first nondomination rank is the best rank, the second nondomination rank is the next-best rank, and so on.

Algorithm 3 Fitness Evaluation

Input: g: a individual, i.e., a giant tour.

Output: f_1: the first objective value; f_2: the second objective value.

1: $s \leftarrow$ perform the giant tour decoding procedure to g and get the feasible solution;

2: $f_1, f_2 \leftarrow$ calculating fitness values based on s;

3: **return** f_1, f_2

Binary Tournament Selection

The binary tournament selection is to select two parents to do the crossover. Two individuals are picked randomly from the parent population. The better one will be the first parent by comparing their nondomination rank (if the two individuals have the same nondomination rank, continue to compare their first objective function value, i.e., f_1, then the second objective function value, i.e., f_2). To select the second parent, repeat the above procedure.

Crossover Procedure

Having the global search ability, crossover operators operate on two different individuals and exchange genetic information fragments of them. The crossover operators employed in our research include: *Partially Mapped Crossover (PMX)*, previously utilized in studies such as [38] [93] [99]. *Order Crossover (OX)*, used in works like [4] [74]. *Best Cost Route Crossover (BCRC)*, which has been applied in research including [34] [65] [67] [68].

All these crossover operators are adopted to perform the crossover. For two parents, six individuals will be generated (two for each crossover operator). The individuals

in the offspring population are not allowed to be the same as the ones in the parent population. Therefore, only the individuals that meet the above criteria will be sampled and added to the offspring population. The crossover will terminate when the number of individuals generated by crossover is equal to (or large than) $P \times (1 - p_m)$.

Survival Selection

The PEAC-HNF algorithm employs a $(\mu + \lambda)$ survival strategy, where μ and λ each equal the population size P. This strategy involves merging the parent and offspring populations and conducting a survival competition to select the next generation's parent population, as illustrated in Fig. 3.3. The combined population undergoes nondominated sorting, incorporating elitism by considering all individuals from both the parent and offspring populations.

Priority is given to individuals in the highest nondominated solution set (first rank), ensuring their survival. If the number of individuals in the first rank is less than μ, all these individuals are selected, and the remainder of the parent population for the next generation is filled by progressively selecting from the next highest nondominated solution sets (second rank, third rank, etc.). If the number of eligible individuals in any rank exceeds the available slots, selection from that rank is made randomly.

Termination Conditions

The algorithm terminates when all the allocated FEs have been utilized.

Packing Process

The flow chart in Fig. 3.4 outlines the packing process within the PEAC-HNF algorithm. The process begins by using a new vehicle to traverse nodes in the giant tour g sequentially. At each node, the vehicle attempts to pack as many boxes as

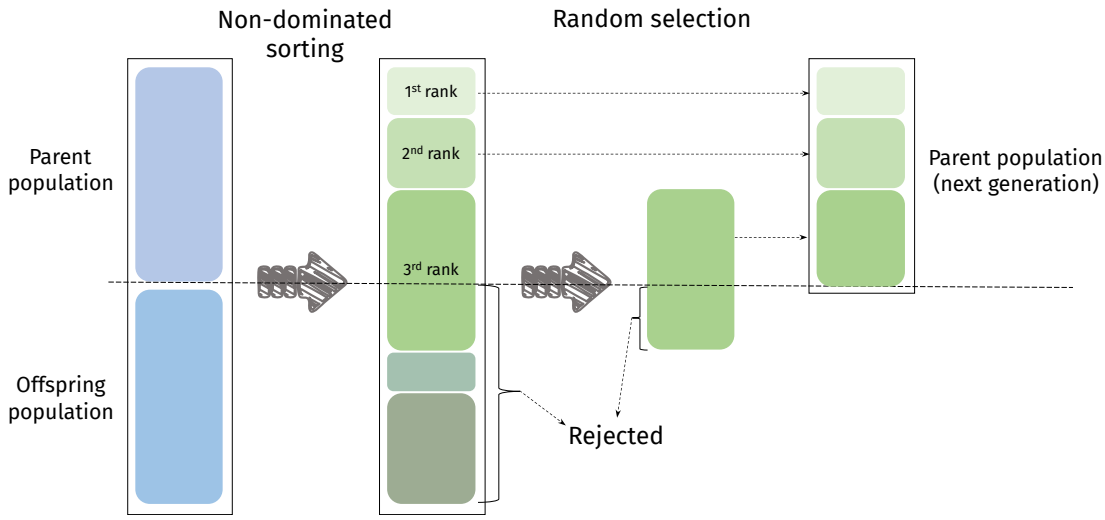


Figure 3.3: The $(\mu+\lambda)$ survival strategy, which aligns with the procedure in NSGA-II [16], except that the crowding distance sorting selection in NSGA-II is replaced by random selection to reduce computational cost.

possible. Once filled, a new vehicle is used to continue the tour. If a node's boxes cannot be completely loaded into the current vehicle, the remaining boxes are carried over to a new vehicle, a process termed as "splitting delivery". In the packing process, a "packing space" is defined as any unoccupied 3D rectangular area within a vehicle, with the coordinate system established using the vehicle's left-bottom-back corner as the origin. These spaces are sorted by the ascending order of their x-, y-, and z-coordinates. Simultaneously, boxes at each node are organized in descending order by their bottom area, volume, width, and length to optimize space utilization.

Besides, in our scenario, the vehicles are heterogeneous, meaning different vehicle type has distinct 3D dimensions, volumes, and weight capacities. Thus, selecting the appropriate type of vehicle—and the number of each type—is crucial for minimizing travel costs and enhancing the loading rate. In our approach, the vehicle type with the largest volume is initially used. Once all nodes have been serviced and all boxes packed, a replacement strategy is employed where smaller vehicles may substitute the larger ones used initially. This strategy aims to enhance the packing ratio by

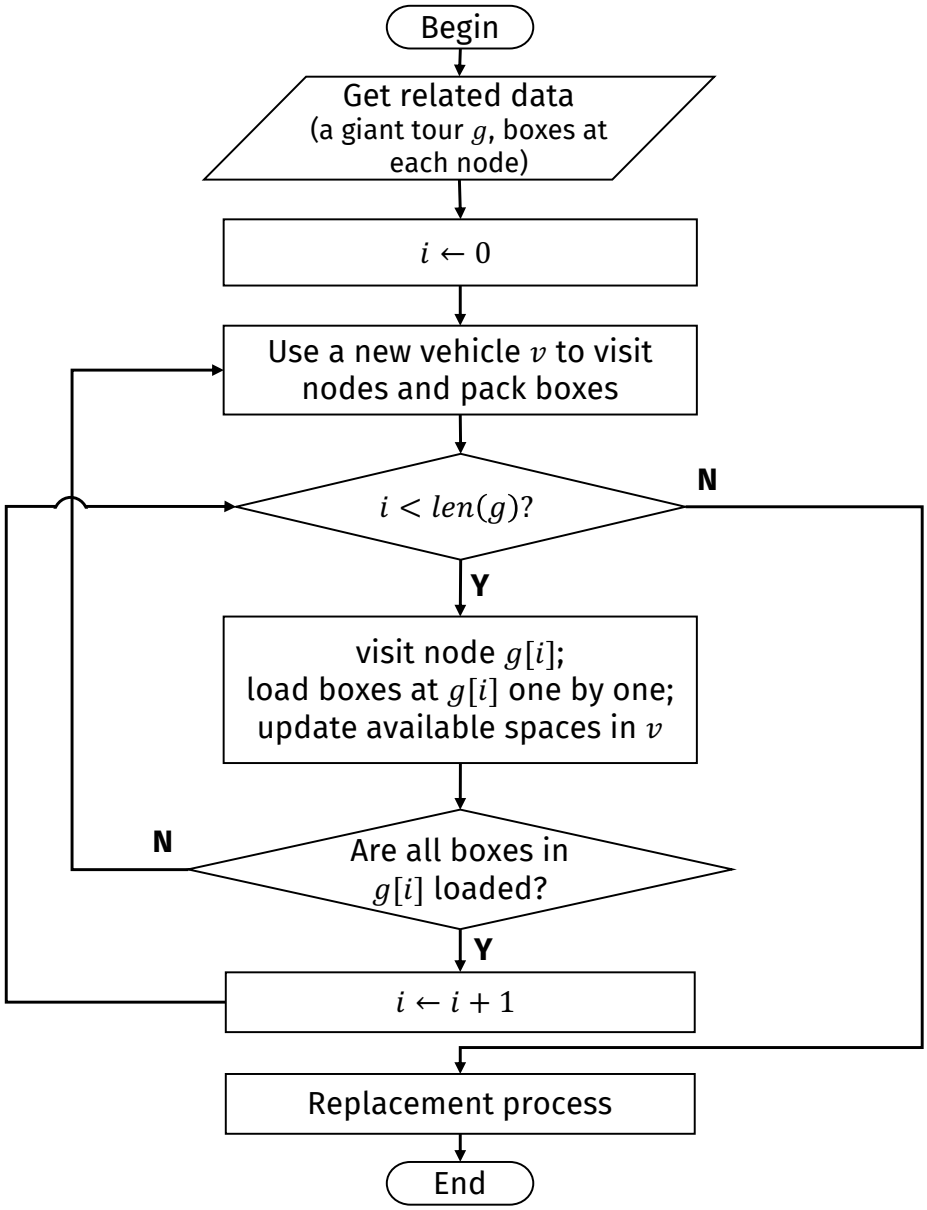


Figure 3.4: The flow chart of the packing process of a giant tour.

potentially replacing some larger vehicles with smaller ones, without discarding any boxes in the route.

3.3 Computational Studies

Our computational studies include comparisons of PEAC-HNF with baselines and state-of-the-art algorithms for solving multi-objective 3L-SDVRP on three datasets. To ascertain PEAC-HNF' s efficacy across objectives, it is also contrasted with single-objective algorithms. Further experimental studies underscore the benefits of HNF mutation and concurrent execution sequence of crossover and mutation. To ensure clarity, some key results are presented in the main text; more detailed results are available online in [113].

3.3.1 Experimental Setting

Problem Instances

This study uses three datasets (242 problem instances in total), as detailed in Tab. 3.1. The first dataset is from the 2021 Evolutionary Multi-Criterion Optimization Conference (EMO2021) logistics competition (referred to as EMO problem instances) [39], comprising 42 diverse problem instances. The second dataset, from [51], includes 100 problem instances. Additionally, we utilized 100 problem instances from Huawei Technologies Ltd. (referred to as HW problem instances), which are derived from real industrial scenarios. All problem instances are available online in [112].

Evaluation Metric

For algorithm performance evaluation, we use the hypervolume (HV) metric [122], which is a pivotal metric in multi-objective optimization for assessing both diversity and convergence. It is actually the area of the shaded polygon formulated by the reference point and non-dominated solutions. For f_1 and f_2, the smaller, the better. For HV, the larger, the better. Calculated by Eq. (3.1), HV_i refers to the HV of the

Table 3.1: Description of problem instances

42 problem instances from EMO2021 logistics competition (EMO problem instances)				
Entry	# nodes	# boxes	# boxes types	# vehicle types
Max	12	1011	297	3
Min	8	98	22	1
Avg (±SD)	9.21 (±1.37)	332.93 (±199.92)	101.74 (±53.58)	2.80 (± 0.44)
100 problem instances from [51]				
Entry	# nodes	# boxes	# box types	# vehicle types
Max	16	2275	392	11
Min	8	51	16	1
Avg (±SD)	9.60 (±1.66)	378.47 (±279.76)	118.38 (±59.58)	2.71 (±1.20)
100 problem instances from Huawei (HW problem instances)				
Entry	# nodes	# boxes	# box types	# vehicle types
Max	16	2275	392	11
Min	8	83	25	1
Avg (±SD)	9.70 (±1.7)	452.58 (±334.73)	129.2 (±63.04)	2.74 (±0.99)

solutions to the problem instance i [41] [51]:

$$HV_i\left(P_i, z^*\right) = vol\left(\cup_{x \in P_i}\left[\bar{f}_1(x), z_1^*\right] \times \left[\bar{f}_2(x), z_2^*\right]\right) \tag{3.1}$$

where

$$\overline{f_1}(x) = \frac{f_1(x) - f_1^{\min}}{f_1^{\max} - f_1^{\min}} \tag{3.1a}$$

$$\overline{f_2}(x) = \frac{f_2(x) - f_2^{\min}}{f_2^{\max} - f_2^{\min}} \tag{3.1b}$$

$$z^* = (1.2, 1.2)^T \tag{3.1c}$$

The function $vol(\cdot)$ refers to the Lebesgue measure. P_i refers to the non-dominated solution set of the instance i obtained by the algorithm. f_1 and f_2 represent two objectives values. z^* is the reference point, reflecting decision-makers' tolerance for the worst solution. If a solution is deemed acceptable, it must meet specific conditions: $\overline{f_1}(x)$, which is the ratio of the gap between its f_1 value and f_1^{min} to the gap between f_1^{max} and f_1^{min}, must be less than z^* (i.e., 1.2). The same condition applies to its f_2 value. For 42 EMO problem instances, the parameters $f_1^{\min}$, $f_1^{\max}$, $f_2^{\min}$,, $f_2^{\max}$, and z^* are officially provided by competition organizer [40] [41] and make up the boundary. For 100 problem instances from [51], we use the same $f_1^{\min}$, $f_1^{\max}$, $f_2^{\min}$, $f_2^{\max}$, and z^* as [51]. Different instances have different boundaries. The statistical test method used here is Mann-Whitney U test ($\alpha = 0.05$).

The hypervolume metric is widely recognized in the multi-objective optimization literature due to its ability to simultaneously assess both the convergence and diversity of a set of solutions. In the context of the 3L-SDVRP, these two aspects are crucial, as practical applications often require not only high-quality solutions but also a diverse set of trade-offs between the objectives, reflecting real-world operational needs. The use of hypervolume, along with the reference points adopted in this study, aligns with established practices in industrial applications and well-known competitions (e.g., EMO2021 [39]), making it highly suitable and relevant for evaluating algorithm performance on this problem.

However, it should be noted that while hypervolume provides a comprehensive assessment, it also has certain limitations. For example, it can be computationally expensive for higher-dimensional problems, and the choice of the reference point can influence the results. Additionally, hypervolume may not capture all aspects of solution set quality, such as the spread along the Pareto front in very specific regions of interest.

Apart from hypervolume, several other quality indicators are commonly used in multi-objective optimization, including inverted generational distance (IGD), generational

distance (GD), spacing, and coverage metrics (e.g., C-metric). Each of these indicators has its own advantages and can provide complementary insights into algorithm performance.

Therefore, it is meaningful to consider a broader set of evaluation criteria in future work. Combining multiple indicators can offer a more comprehensive and nuanced assessment of algorithm performance, ensuring that different facets of solution quality are thoroughly evaluated.

Parameters

PEAC-HNF' s parameters in Algorihtm 2 are configured as follows: G set to 20, p_m set to 0.5, with fitness evaluations (FEs) ranging from 1,000 to 12,000. The population size is derived from $\#FEs/\#generations$, and each experiment is replicated 30 times. It is worth noting that we intentionally avoided a detailed parameter tuning process to demonstrate that the strong performance of our algorithm is due to its innovative design rather than finely-tuned parameters.

Experimental Environments

The algorithms are implemented in Python 3.7, and experiments are run on Dell R370 server with 2x Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz CPU, 128G RAM, and CentOS 7.6 operating system.

3.3.2 Comparative Analysis

Comparison to Baselines

To validate the effectiveness of PEAC-HNF, it is compared against two baselines on 42 EMO problem instances (see Tab. 3.1): Baseline 1, inspired by SDVRLH2 [10], an

effective method for 3L-SDVRP, and Baseline 2, MOGA [65], a pioneering study in applying multi-objective algorithms to 3L-SDVRP. Experiments are conducted under varying FEs (1,000 to 12,000). Although problem constraints in the literature may vary slightly, such as the relaxation of the last-in-first-out constraint in [65], these methods still provide excellent baselines for comparison with our algorithm. Necessary adjustments and modifications to the algorithms ensure that these comparisons are meaningful and valid.

Tab. 3.2 displays HV values across multiple runs for each instance. PEAC-HNF matches or exceeds Baseline 1 on 39 out of 42 instances, significantly outperforming on 27. Against Baseline 2, PEAC-HNF is superior on 27 instances and comparable on 14. Fig. 3.5 illustrates the HV curves during search process, highlighting PEAC-HNF' s strengths in solution quality. Fig. 3.6 details performance variances at different FEs. Compared to Baseline 1 at 1,000 FEs, PEAC-HNF outperforms on only 5 instances. The advantage of PEAC-HNF becomes significant as the number of FEs increases, notably outperforming on 27 instances at 12,000 FEs. Against Baseline 2, PEAC-HNF consistently shows superior performance, more evident as FEs increase. Notably, PEAC-HNF's advantage over both baselines enhances with higher FEs, showcasing its scalability and effectiveness.

Comparison to MOEA/D-OM on EMO Problem Instances

PEAC-HNF is also compared against MOEA/D-OM [51], which is the state-of-the-art multi-objective algorithm for solving 3L-SDVRP and also the EMO2021 Logistics Competition's champion algorithm [40]. In our experiments, the default parameters of MOEA/D-OM are used. For a fair comparison, the number of fitness evaluations (FEs) for our PEAC-HNF is set to match MOEA/D-OM, i.e., 2000.

Tab. 3.3 demonstrates that PEAC-HNF significantly outperforms MOEA/D-OM on all 42 problem instances, with HV values at least doubling on these instances. As

Table 3.2: Hypervolume (*HV*) values (avg±std) of baseline 1, baseline 2, and PEAC-HNF (12000 FEs, 30 runs) on EMO problem instances.

Instances	Baseline 1	Baseline 2	PEAC-HNF	Instances	Baseline 1	Baseline 2	PEAC-HNF
E1594609968101	0.9017±0.0159 (-)	0.8936±0.019 (-)	**0.9151±0.012**	ECO2008190025	0.7122±0.0029 (-)	0.7177±0.0021	**0.7172±0.0026**
E1595638696418	0.7903±0.023 (-)	0.7967±0.0245 (-)	**0.7992±0.0284**	ECO2008220028	**0.5466±0.0032 (+)**	0.5104±0.0237 (-)	0.5452±0.0075
E1596676780873	1.4053±0.0178 (-)	1.4105±0.0063 (-)	**1.4156±0.0052**	ECO2008250134	**1.1921±0.0085**	1.1877±0.0265	1.1902±0.0116
E1596943422130	0.5198±0.0359 (-)	0.5364±0.0025	**0.5377±0.0009**	ECO2008250157	**1.2746±0.0157**	1.2268±0.0244 (-)	1.2674±0.0204
E1597112047246	**1.0128±0.0291 (+)**	0.9022±0.0586 (-)	0.9942±0.0347	ECO2008260084	0.3022±0.0013 (-)	0.2912±0.0242 (-)	**0.3036±0.0**
E1597284418604	1.0865±0.0272 (-)	**1.1115±0.022 (+)**	1.102±0.0186	ECO2008260104	1.0461±0.0113	1.0237±0.0176 (-)	**1.0481±0.0079**
E1597802825734	**1.882±0.0342 (+)**	1.8154±0.0388 (-)	1.8607±0.0339	EF1594632252863	0.7207±0.024	0.7034±0.0246 (-)	**0.7277±0.0231**
E1597809878442	0.5942±0.034 (-)	0.6046±0.029	**0.6195±0.0102**	EF1594805700879	0.7646±0.0105 (-)	0.7678±0.0017 (-)	**0.7691±0.0012**
ECO2007210113	0.8424±0.0013 (-)	0.8441±0.0013	**0.8445±0.0005**	EF1594955642012	0.2455±0.0065 (-)	0.2466±0.0057 (-)	**0.2518±0.0067**
ECO2007240027	0.8746±0.0103 (-)	**0.8969±0.0392**	0.8877±0.0213	EF1595231952604	1.3721±0.0105 (-)	1.3788±0.0064	**1.3797±0.0035**
ECO2007250011	0.9053±0.0461 (-)	0.9266±0.0229 (-)	**0.9338±0.0198**	EF1595295309474	0.642±0.0125 (-)	**0.667±0.0059**	0.6669±0.0064
ECO2007270107	1.0917±0.0388 (-)	**1.2359±0.002**	1.2309±0.028	EF1595295895624	0.4501±0.3842 (-)	0.6694±0.3767 (-)	**0.8765±0.3483**
ECO2007290014	1.1147±0.0152 (-)	1.1221±0.011 (-)	**1.1273±0.0087**	EF1595381687057	0.8853±0.0015 (-)	0.8801±0.0104 (-)	**0.887±0.0022**
ECO2007300009	0.3322±0.1926 (-)	0.6153±0.1786 (-)	**0.8243±0.0625**	EF1595474862428	**1.2907±0.0494**	1.1553±0.0411 (-)	1.2769±0.0517
ECO2008110020	**0.6736±0.0304**	0.6116±0.044 (-)	0.6703±0.0393	EF1595650233198	**0.2382±0.0**	0.2183±0.0508 (-)	**0.2382±0.0**
ECO2008110027	0.9109±0.0211	0.848±0.0243 (-)	**0.917±0.0284**	EF1595817642706	**1.0452±0.0013**	1.035±0.0099 (-)	1.0431±0.0086
ECO2008120023	0.513±0.0321 (-)	**0.5438±0.0**	**0.5438±0.0**	EF1596011846243	1.1537±0.0031 (-)	**1.1589±0.0068**	1.1583±0.0059
ECO2008120038	0.8118±0.1441	0.7654±0.104 (-)	**0.8609±0.0858**	EF1596102693660	0.2308±0.1528 (-)	**0.682±0.0027**	0.6815±0.0036
ECO2008130066	**1.0502±0.0256**	1.0309±0.019 (-)	1.0459±0.0176	EF1596466497583	0.8605±0.0153 (-)	0.8791±0.019 (-)	**0.889±0.0156**
ECO2008130135	0.6234±0.2677 (-)	0.6734±0.203 (-)	**0.7385±0.2353**	EF1596525590000	0.2341±0.0042 (-)	**0.2392±0.0022**	**0.2392±0.0025**
ECO2008160002	**0.7874±0.076**	0.7516±0.0487 (-)	0.7678±0.0451	EF1596590704042	1.4513±0.5333 (-)	**1.5932±0.5284**	1.4566±0.5355

Baseline 1 VS PEAC-HNF (+/-/=)	Baseline 2 VS PEAC-HNF (+/-/=)
3/27/12	1/27/14

Note: The symbol "+" /"-" indicates the related algorithm is significantly better/worse than PEAC-HNF according to the statistical test. For *HV*, the larger, the better.

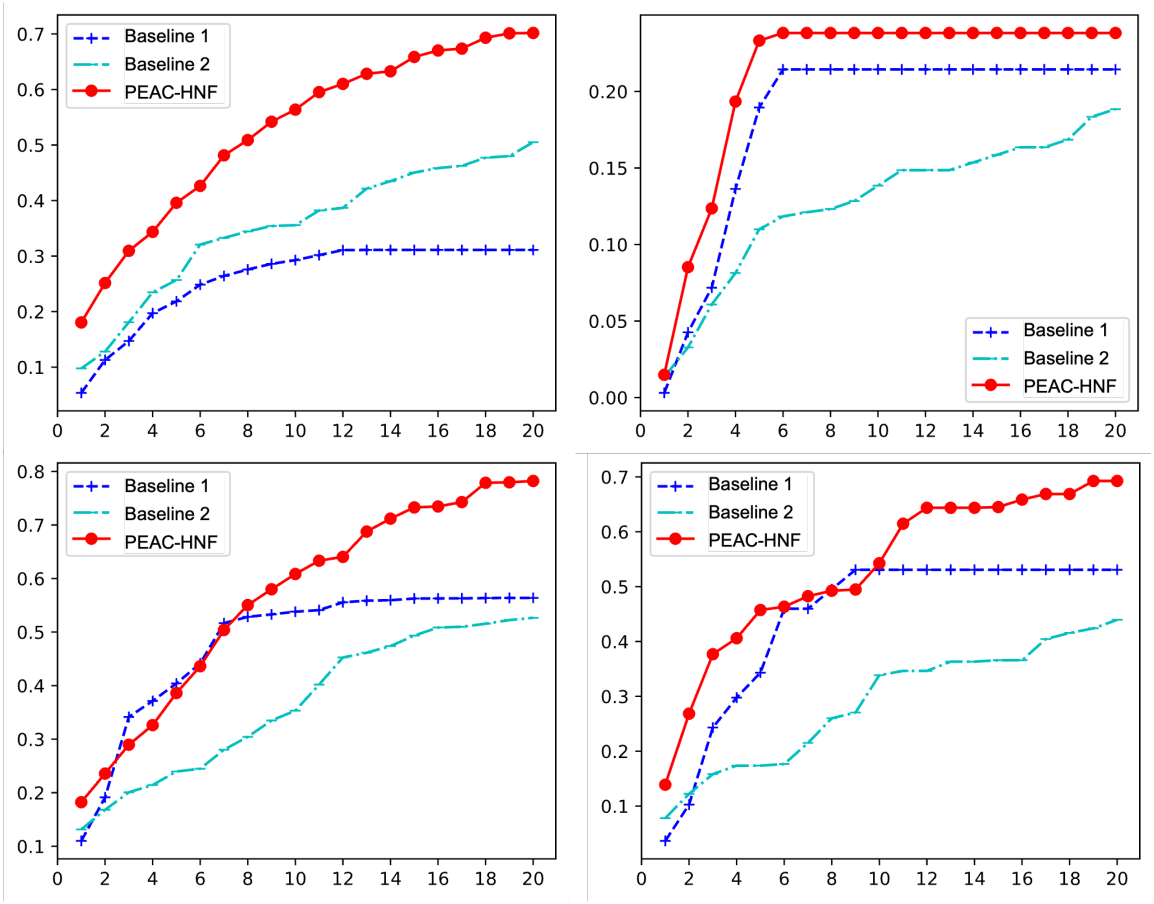


Figure 3.5: Evolutionary curves of PEAC-HNF and baselines on some instances (8000 FEs, 30 runs). The x-coordinate represents the number of generations. The y-coordinate represents the average HV of 30 independent runs.

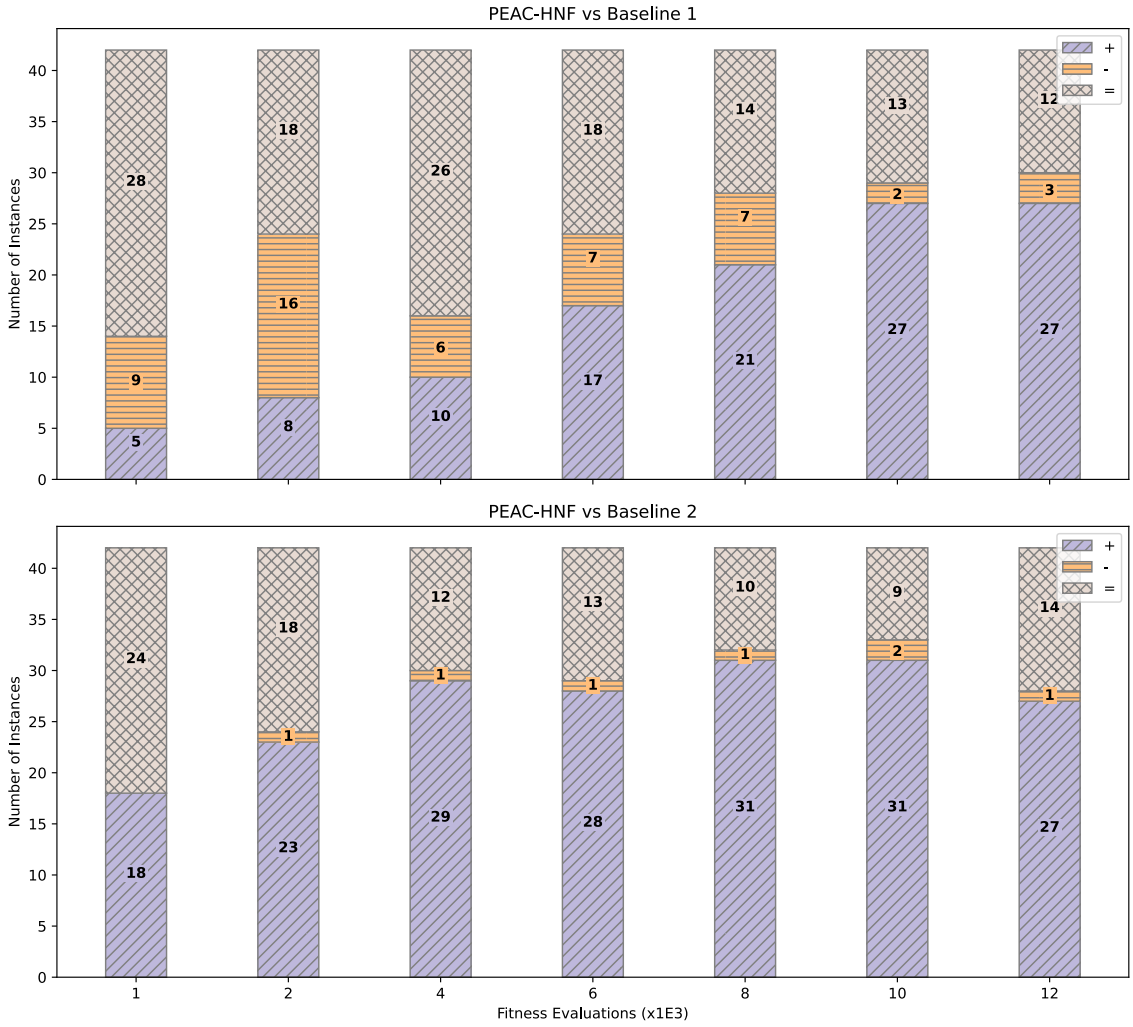


Figure 3.6: Comparison of PEAC-HNF with baseline methods under different fitness evaluations (FEs) on EMO instances. The symbols "+/-/=" respectively indicate the number of problem instances where PEAC-HNF significantly outperforms/underperforms/has no significant difference compared to the corresponding baseline method.

shown in the table, MOEA/D-OM achieves a hypervolume (HV) of zero for some problem instances, indicating that its solutions are worse than the acceptable worst solution and are therefore discarded. Moreover, our empirical findings suggest that the PEAC-HNF algorithm is computationally intensive, requiring substantial CPU time. To further investigate this observation, we performed an in-depth analysis of the PEAC-HNF algorithm, as detailed in Section 3.3.3 (Time Analysis part). The experimental results reveal that the process of determining the packing plan constitutes approximately 98% of the total computational runtime. This can be attributed to the fact that each individual (i.e., the giant tour) requires a complete recalculation of the packing arrangement from the ground up. Consequently, this process is computationally demanding and contributes significantly to the overall runtime of the algorithm.

Additionally, our results show that MOEA/D-OM's performance significantly declines with previously unseen problems or new scenarios, underscoring challenges in its generalization capability [53].

Comparison to MOEA/D-OM on Problem Instances from [51]

The study by [51] examined 100 problem instances, detailed in Tab. 3.1. Our proposed PEAC-HNF was compared against MOEA/D-OM across these instances, with results summarized in Tab. 3.4. In terms of total hypervolume, which represents the sum of average HVs across all instances, PEAC-HNF significantly outperformed MOEA/D-OM for minimum, maximum, and average total hypervolume values. Specifically, PEAC-HNF achieved superior results on 92 out of 100 instances and matched MOEA/D-OM on 5 instances, underscoring its enhanced performance. While our algorithm yields superior solutions, it also incurs increased computational overhead. As demonstrated in our experimental analysis in Section 3.3.3 (Time Analysis part), this is primarily due to the requirement for a complete recalculation of the

Table 3.3: Hypervolume (HV) values of MOEA/D-OM and our PEAC-HNF (2000 FEs, 10 runs) on EMO problem instances. The symbol "+/-" indicates the related algorithm is significantly better/worse than the other one according to the statistical test. For ***HV***, the larger, the better

Instances	MOEA/D-OM	PEAC-HNF	Instances	MOEA/D-OM	PEAC-HNF
	HV (avg±std)	HV (avg±std)		HV (avg±std)	HV (avg±std)
E1594609968101	0.3277±0.0352 (-)	**0.845±0.0336** (+)	ECO2008190025	0.0±0.0 (-)	**0.7092±0.0035** (+)
E1595638696418	0.0±0.0 (-)	**0.7152±0.0863** (+)	ECO2008220028	0.0586±0.0688 (-)	**0.4789±0.0357** (+)
E1596676780873	0.0±0.0 (-)	**1.3807±0.0233** (+)	ECO2008250134	0.0±0.0 (-)	**1.1393±0.0311** (+)
E1596943422130	0.0122±0.0365 (-)	**0.5213±0.0308** (+)	ECO2008250157	0.0±0.0 (-)	**1.1579±0.0424** (+)
E1597112047246	0.0±0.0 (-)	**0.8095±0.1153** (+)	ECO2008260084	0.0±0.0 (-)	**0.2284±0.0794** (+)
E1597284418604	0.3033±0.0292 (-)	**1.0549±0.0206** (+)	ECO2008260104	0.2247±0.0184 (-)	**0.9773±0.0413** (+)
E1597802825734	0.0±0.0 (-)	**1.6222±0.1726** (+)	EF1594632252863	0.0±0.0 (-)	**0.6114±0.0619** (+)
E1597809878442	0.0±0.0 (-)	**0.5178±0.0638** (+)	EF1594805700879	0.0±0.0 (-)	**0.7458±0.0146** (+)
ECO2007210113	0.3214±0.0612 (-)	**0.8396±0.0067** (+)	EF1594955642012	0.0±0.0 (-)	**0.2317±0.0087** (+)
ECO2007240027	0.1873±0.2533 (-)	**0.805±0.1057**	EF1595231952604	0.0±0.0 (-)	**1.357±0.0376** (+)
ECO2007250011	0.0±0.0 (-)	**0.8285±0.0878**	EF1595295309474	0.0±0.0 (-)	**0.6482±0.0089** (+)
ECO2007270107	0.4775±0.1462 (-)	**1.1482±0.0903** (+)	EF1595295895624	0.0±0.0 (-)	**0.4006±0.466** (+)
ECO2007290014	0.2789±0.3957 (-)	**1.0714±0.0206** (+)	EF1595381687057	0.4416±0.098 (-)	**0.8706±0.0212** (+)
ECO2007300009	0.0±0.0 (-)	**0.3868±0.2756** (+)	EF1595474862428	0.0±0.0 (-)	**1.083±0.0896** (+)
ECO2008110020	0.0±0.0 (-)	**0.5229±0.0461** (+)	EF1595650233198	0.0±0.0 (-)	**0.2283±0.0373** (+)
ECO2008110027	0.0±0.0 (-)	**0.7943±0.0409** (+)	EF1595817642706	0.0813±0.058 (-)	**0.9858±0.0339** (+)
ECO2008120023	0.0±0.0 (-)	**0.523±0.0263** (+)	EF1596011846243	0.0±0.0 (-)	**1.135±0.0181** (+)
ECO2008120038	0.0±0.0 (-)	**0.5564±0.1498** (+)	EF1596102693660	0.0±0.0 (-)	**0.3053±0.21** (+)
ECO2008130066	0.3639±0.0193 (-)	**0.9894±0.032** (+)	EF1596466497583	0.0±0.0 (-)	**0.8141±0.0377** (+)
ECO2008130135	0.1742±0.0192 (-)	**0.3371±0.2177** (+)	EF1596525590000	0.2151±0.0206 (-)	**0.2358±0.0045** (+)
ECO2008160002	0.377±0.0693 (-)	**0.702±0.0196** (+)	EF1596590704042	0.0±0.0 (-)	**1.3416±0.4699** (+)

Summary	PEAC-HNF VS MOEA/D-OM (+/-/=)	Avg CPU Time (s)	
		MOEA/D-OM	PEAC-HNF
	42/0/0	60	1170

packing arrangement from scratch of the giant tour. The resulting computational effort required for these packing calculations substantially contributes to the elevated CPU time.

In [51], the runtime of MOEA/D-OM was limited to 60 seconds. To investigate if extended runtime would enhance MOEA/D-OM's performance, we increased its runtime to approximately 600 seconds (MOEA/D-OM with extended time), aligning it with our proposed PEAC-HNF. The results, detailed in Tab. 3.4, reveal that even with a tenfold increase in runtime, MOEA/D-OM's performance improvement was not significant. The total hypervolume increased by only 1.21 compared to the original 60-second setting. In contrast, PEAC-HNF significantly outperformed MOEA/D-OM with extended time, achieving superior minimum, maximum, and average hypervolume values. Specifically, PEAC-HNF obtained better results on 89 out of 100 instances and matched MOEA/D-OM on 8 instances.

Comparison to MOEA/D-OM on HW Problem Instances

To further validate our algorithm's performance, we conducted experiments on the HW problem instances. The results, as shown in Tab. 3.5, demonstrate that PEAC-HNF significantly outperforms MOEA/D-OM in terms of total hypervolume (i.e., the sum of the average hypervolume across 100 instances). Specifically, PEAC-HNF achieves superior minimum, maximum, and average values. On a per-instance basis, PEAC-HNF produces significantly better results than MOEA/D-OM in 97 out of the 100 problem instances.

Comparison to Single-Objective Methods

This problem has two objectives, which leads to an important consideration: can single-objective methods effectively address both objectives? If single-objective approaches are sufficient, then the design of multi-objective algorithms might not be

Table 3.4: Total hypervolume (i.e., the sum of hypervolume values across all 100 problem instances from [51]) of MOEA/D-OM and our PEAC-HNF (2000 FEs, 10 runs). T is the CPU time. For MOEA/D-OM with extended time, the maximum running time is extended to approximately 600 seconds. The statistical test method used here is the Mann-Whitney U test ($\alpha = 0.05$). The symbol "+/-" indicates our PEAC-HNF is significantly better/worse than MOEA/D-OM according to the statistical test. For ***HV***, the larger, the better

MOEA/D-OM				MOEA/D-OM with extended time				PEAC-HNF			
Total hypervolume			T	Total hypervolume			T	Total hypervolume			T
Min	Max	Avg+std		Min	Max	Avg+std		Min	Max	Avg+std	
107.22	108.66	108.17±0.35	60	108.58	110.00	109.38±0.38	608	139.75	140.70	140.39±0.37	606
PEAC-HNF VS MOEA/D-OM (+/-/=)				PEAC-HNF VS MOEA/D-OM with extended time (+/-/=)							
92/3/5				89/3/8							

Table 3.5: Total hypervolume (i.e., the sum of hypervolume values across all 100 HW problem instances) of MOEA/D-OM and our PEAC-HNF (2000 FEs, 10 runs) on HW problem instances. T is the CPU time. The statistical test method used here is Mann-Whitney U test ($\alpha = 0.05$). The symbol "+/-" indicates our PEAC-HNF is significantly better/worse than MOEA/D-OM according to the statistical test. For HV, the larger, the better

MOEA/D-OM				PEAC-HNF			
Total hypervolume			T	Total hypervolume			T
Min	Max	Avg+std		Min	Max	Avg+std	
107.21	109.04	107.91±0.53	60	142.01	142.96	142.32±0.19	781
PEAC-HNF VS MOEA/D-OM (+/-/=)							
97/3/0							

necessary. To explore this, PEAC-HNF is compared against single-objective methods. These methods evaluate individuals based solely on two objectives, f_1 and f_2. The first method prioritizes minimizing f_1, then f_2 (f_1-first-f_2-second), while the second method reverses this priority (f_2-first-f_1-second).

Figs. 3.7 and 3.8 display the distribution of solutions in the objective space for multiple problem instances, comparing the results obtained by the PEAC-HNF method and single-objective methods. Each figure shows the final nondominated population (obtained by PEAC-HNF) and the final best solution (obtained by the single-objective method) for 30 independent runs. Fig. 3.7 focuses on the comparison between PEAC-HNF and the f1-first-f2-second method, while Fig. 3.8 compares PEAC-HNF with the f2-first-f1-second method. Evidently, PEAC-HNF not only outperforms these methods in solution quality by performing well in both objectives but also showcases a significantly greater diversity in its solutions. This implies that PEAC-HNF is

capable of offering a range of trade-offs for decision-makers.

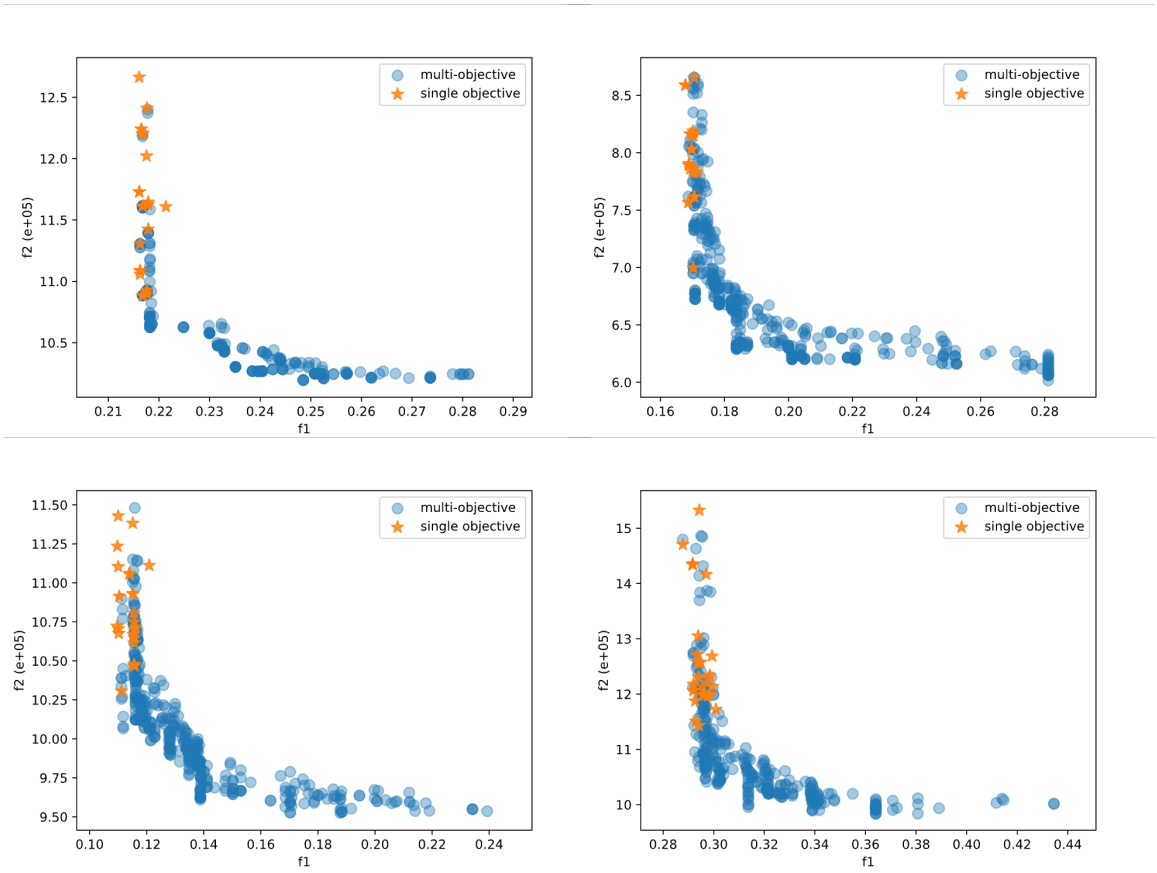


Figure 3.7: The final nondominated population of some instances for 30 independent runs.Use f_1 to refer to vehicle loading rate, $f2$ to represent total travel distance. In the figure, "single-objective" refers to the f_1-first-f_2-second single-objective method. "multi-objective" refers to the PEAC-HNF.

3.3.3 Further Analysis

Further experimental studies were conducted to validate the impact of our proposed HNF mutation and the concurrent execution of crossover and mutation. Our PEAC-HNF was compared to its variants (PEA-C-E, PEA-S-HNF, and PEA-S-E) on 42 EMO problem instances. Here, "C" stands for concurrent execution of crossover

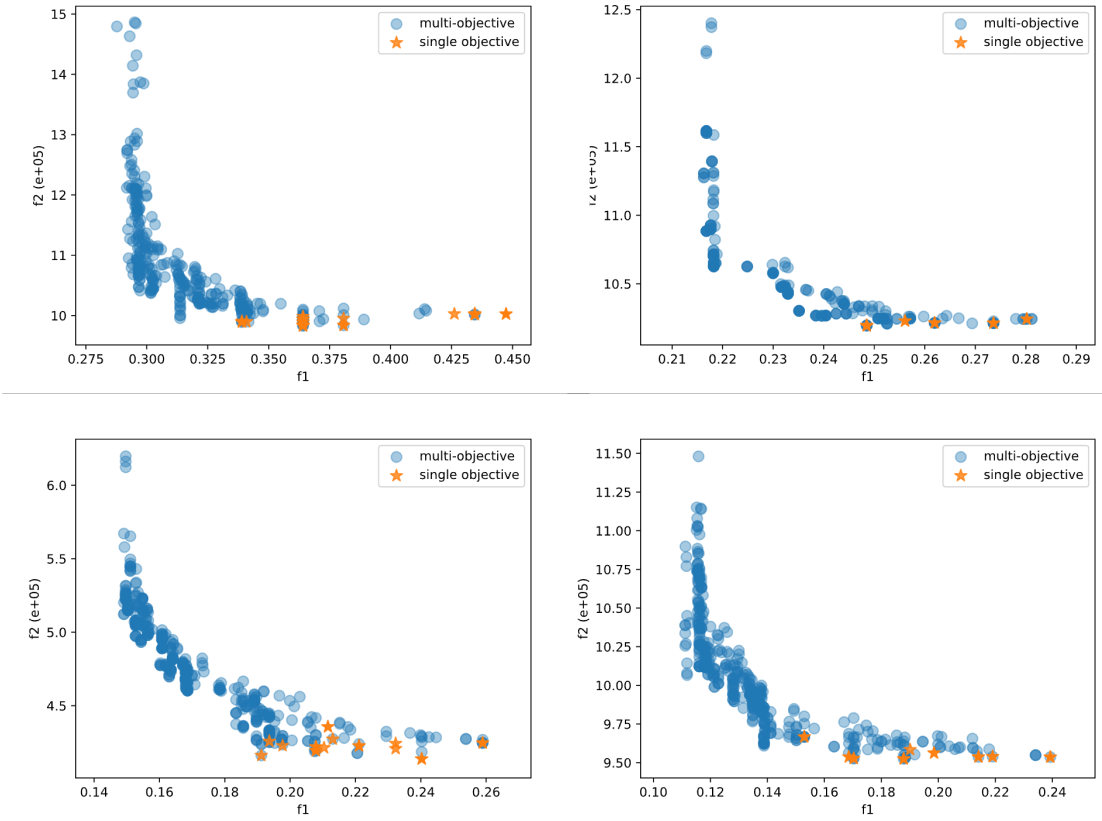


Figure 3.8: The final nondominated population of some instances for 30 independent runs. Use f_1 to refer to vehicle loading rate, $f2$ to represent total travel distance. In the figure, "single-objective" refers to the f_2-first-f_1-second single-objective method. "multi-objective" refers to the PEAC-HNF.

and mutation, "S" for serial execution, "HNF" for hierarchical neighborhood filtering mutation, and "E" for equal mutation probability across individuals. Fig. 3.9 shows the HV curves of PEAC-HNF and its variants. Fig. 3.10 provides detailed comparative results between PEAC-HNF and its variants across different number of FEs. More detailed experimental results are available online in [113].

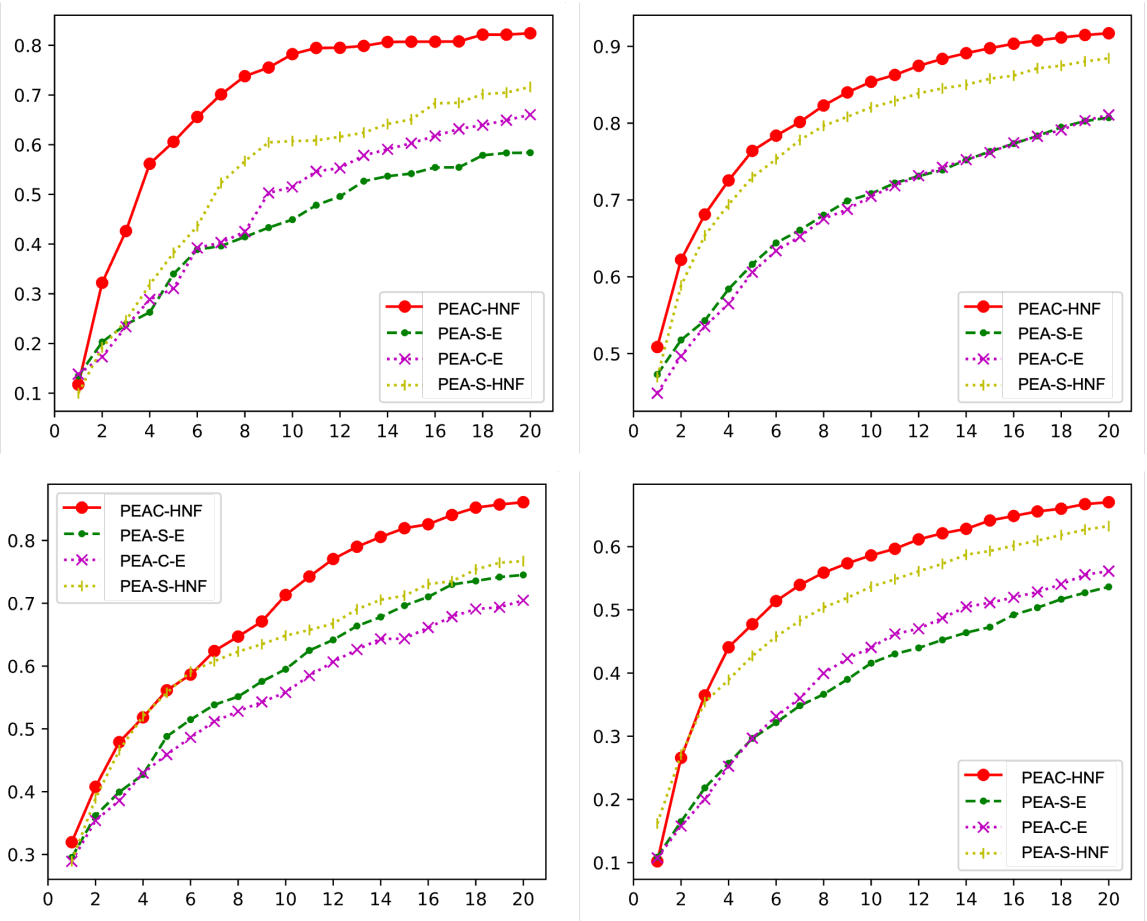


Figure 3.9: HV curves of PEAC-HNF and its variants on some instances (12000 FEs, 30 runs). The x-axis represents the number of generations. The y-axis represents the average HV of 30 independent runs.

Impact of HNF Mutation

PEAC-HNF consistently outperformed PEA-C-E across different numbers of fitness evaluations (FEs). For instance, at 1000 FEs, PEAC-HNF outperformed PEA-C-E on 24 instances and matched its performance on 18; at 8000 FEs, it was significantly better on 35 instances and equal on six. These results demonstrates that our HNF mutation indeed enhances algorithmic performance.

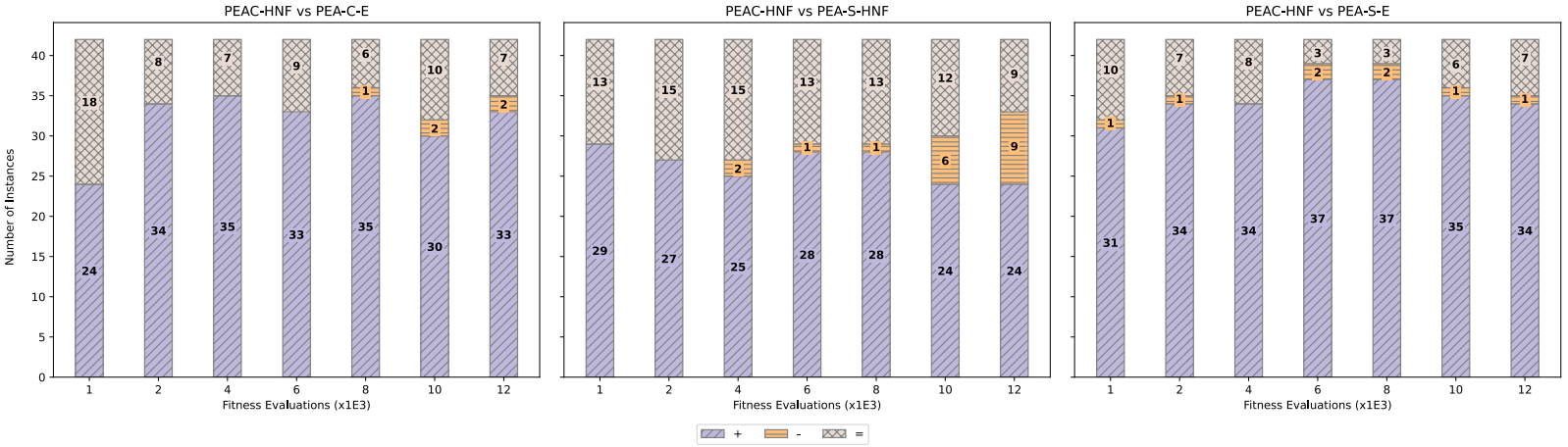


Figure 3.10: Comparison of PEAC-HNF with its variants under different fitness evaluations (FEs) on EMO problem instances. "C" stands for concurrent execution of crossover and mutation, "S" for serial execution, "HNF" for hierarchical neighborhood filtering mutation, and "E" for equal mutation probability across individuals. The symbols "+/-/=" respectively indicate the number of problem instances where PEAC-HNF significantly outperforms/underperforms/has no significant difference compared to the corresponding method.

Impact of Concurrent Crossover and Mutation

Compared to PEA-S-HNF, PEAC-HNF also demonstrated superior performance. At 2000 FEs, it was significantly better on 27 instances and equal on five; at 8000 FEs, it outperformed PEA-S-HNF on 28 instances and was equal on 13. This highlights the advantages of concurrently executing crossover and mutation.

Synergistic Impact of HNF Mutation and Concurrent Crossover and Mutation

Against PEA-S-E, PEAC-HNF showed significant improvements at various FE counts. At 1000 FEs, it surpassed PEA-S-E on 31 instances and matched on ten; at 8000 FEs, it outperformed on 37 instances and was equivalent on three. These findings further confirm the enhanced solution quality achieved through the synergistic effects of the HNF mutation and the concurrent execution of crossover and mutation.

Impact of the Number of FEs

Moreover, experimental results suggest that the optimal performance was at around 8000 FEs. Increases beyond this threshold do not always improve outcomes, underscoring the importance of efficiently utilizing limited computational resource for effective search.

Impact of Different Parent Selection Methods

In this chapter, we adopt the binary tournament (BT) selection as the parent selection strategy (see Section 3.2.4), owing to its simplicity and effectiveness. To investigate the impact of different parent selection methods, we further implemented more sophisticated approaches, including crowded tournament (CT) selection and rank-based roulette wheel (RRW) selection. These selection methods were integrated into our

algorithm, and a series of experiments were conducted on the EMO problem instances to compare their performance.

The results are summarized in Tables 3.6 and 3.7. When the number of function evaluations (# FE) was set to 2000 (see Table 3.6), BT performed comparably to CT and RRW. Specifically, BT and CT exhibited no significant difference on 40 out of 42 instances, while BT was not inferior to RRW on 41 out of 42 instances. Similarly, when # FE was set to 8000 (Table 3.7), the performance of BT remained statistically indistinguishable from CT and RRW.

These findings can be attributed to the inherent complexity of the 3L-SDVRP, which integrates both the vehicle routing problem (VRP) and the three-dimensional loading problem (3DLP)—both of which are NP-hard. The resulting search space is extremely complex, and thus, existing parent selection methods do not yield significant differences in performance for this problem. To further improve solution quality, it is necessary to gain deeper insights into the problem characteristics and leverage domain knowledge to design more powerful parent selection mechanisms and search operators. Such directions have been explored in Chapter 6.

Time Analysis

From the aforementioned experimental results, we observed that the proposed PEAC-HNF algorithm requires more CPU time compared to MOEA/D-OM. Although the two algorithms were implemented in different programming languages (PEAC-HNF in Python and MOEA/D-OM in Java), we still sought to investigate the factors contributing to PEAC-HNF' s higher time consumption. As theoretical analysis of computational time complexity is only feasible for simple metaheuristic algorithms on artificial problems, we conducted experiments to perform an empirical time analysis.

Table 3.8 presents the CPU time statistics for solving the 3D packing component within the PEAC-HNF algorithm. The metrics include the total packing time (Total

Table 3.6: Hypervolume (HV) values of PEAC-HNF with different parent selection methods (2000 FEs, 10runs) on EMO problem instances. CT = Crowded Tournament. RRW = Rank-based Roulette Wheel. BT = Binary Tournament.

Instances	CT	RRW	BT (Ours)	Instances	CT	RRW	BT (Ours)
E1594609968101	0.8389±0.0336	0.8484±0.0282	0.8275±0.044	ECO2008130135	0.3377±0.2555	0.7087±0.0033	1.1488±0.0423
E1595638696418	0.7±0.1112	0.7341±0.0383	0.6426±0.1115	ECO2008130066	0.9754±0.0204(-)	0.4513±0.037	0.459±0.0424
E1596676780873	1.3843±0.013	1.3893±0.0092 (+)	1.3713±0.018	ECO2008260084	0.2273±0.0725	1.148±0.0324	0.7907±0.0516
E1596943422130	0.5342±0.0022	0.532±0.0026	0.5327±0.0037	ECO2008220028	0.4638±0.0482	1.1439±0.0331	0.7314±0.0801
E1597112047246	0.729±0.1264	0.767±0.1391	0.697±0.1542	ECO2008190025	0.7102±0.0036	0.2604±0.0699	1.0104±0.0239
E1597284418604	1.0387±0.0306	1.0475±0.0115	1.0561±0.0285	ECO2008260104	0.961±0.0312 (-)	0.9574±0.0444	0.9846±0.024
E1597802825734	1.6187±0.1713	1.7233±0.0673	1.7017±0.1091	EF1594632252863	0.6037±0.0679	0.6322±0.0858	0.6204±0.0583
ECO2007210113	0.8359±0.0099	0.4952±0.0616	0.4954±0.0707	EF1594805700879	0.7576±0.0075	0.7475±0.0189	0.7451±0.0176
E1597809878442	0.4986±0.037	0.8416±0.0017	0.8418±0.0013	EF1594955642012	0.2377±0.0082	0.2338±0.007	0.2321±0.0047
ECO2007240027	0.7377±0.1576	0.788±0.0535	0.8034±0.0889	EF1595231952604	1.367±0.0122	1.3552±0.0255	1.3564±0.0135
ECO2007250011	0.8603±0.0644	0.7741±0.0921	0.7529±0.1706	EF1595295309474	0.6473±0.0092	0.6513±0.0088	0.3141±0.3923
ECO2007270107	1.1365±0.0964	1.1848±0.0703	1.1305±0.1002	EF1595295895624	0.218±0.2893	0.3617±0.3865	0.6464±0.0072
ECO2007290014	1.0718±0.0132	1.068±0.0118	1.0664±0.0248	EF1595381687057	0.8616±0.0387	0.8778±0.0076	0.8638±0.0207
ECO2007300009	0.5637±0.2632	0.5086±0.2585	0.4429±0.2677	EF1595817642706	0.989±0.0331	1.0575±0.0731	0.2382±0.0
ECO2008110027	0.8014±0.0389	0.5473±0.081	0.2092±0.0836	EF1595474862428	1.0978±0.0709	0.2382±0.0	1.0853±0.0792
ECO2008120038	0.5214±0.1167	0.7807±0.0391	0.4932±0.0694	EF1595650233198	0.2382±0.0	0.9964±0.0265	1.138±0.0148
ECO2008250134	1.1548±0.023	0.5281±0.0238	0.5176±0.026	EF1596011846243	1.1361±0.0181	1.1295±0.0219	1.0018±0.0277
ECO2008250157	1.1519±0.0241	0.4785±0.0905	0.3174±0.2294	EF1596102693660	0.2832±0.197	0.1799±0.0554	0.3287±0.2256
ECO2008110020	0.5315±0.0514	0.9804±0.0267 (-)	1.1537±0.0318	EF1596466497583	0.7883±0.0497	0.7967±0.0292	0.8156±0.0343
ECO2008160002	0.7132±0.0194	0.4776±0.2682	0.7097±0.0031	EF1596525590000	0.2357±0.0036	0.2345±0.0045	0.2334±0.0045
ECO2008120023	0.5124±0.0255	0.6895±0.0193	0.6311±0.1489	EF1596590704042	1.1729±0.3485	1.057±0.008	1.1711±0.3456

CT VS BT (+/-/=)	RRW VS BT (+/-/=)
0/2/40	1/1/40

Note: The symbol "+/-" indicates the related method is significantly better/worse than the binary tournament method according to the statistical test. For HV, the larger, the better.

Table 3.7: Hypervolume (HV) values of PEAC-HNF with different parent selection methods (8000 FEs, 10runs) on EMO problem instances. CT = Crowded Tournament. RRW = Rank-based Roulette Wheel. BT = Binary Tournament.

Instances	CT	RRW	BT (Ours)	Instances	CT	RRW	BT (Ours)
E1594609968101	0.8957±0.0141	0.8973±0.0136	0.9071±0.0141	ECO2008130135	0.7146±0.0281	0.5438±0.0	0.7096±0.0301
E1595638696418	0.7961±0.029	0.8053±0.0107	0.8033±0.0103	ECO2008130066	1.0409±0.0108	0.7092±0.0217	1.0436±0.0084
E1596676780873	1.4066±0.0081	1.4138±0.0054	1.4119±0.0083	ECO2008260084	1.1793±0.0123	1.2476±0.0231	1.18±0.0125
E1596943422130	0.5365±0.0021	0.9296±0.0734	0.5371±0.0016	ECO2008220028	0.7798±0.0717	0.6556±0.2279 (-)	0.7655±0.066
E1597112047246	0.9748±0.0237 (+)	0.5362±0.0016	0.9291±0.0566	ECO2008190025	0.3026±0.0013	0.717±0.0028	0.2858±0.0535
E1597284418604	1.0967±0.0214	1.0869±0.0304	1.0983±0.0238	ECO2008260104	0.8445±0.101 (+)	1.1873±0.0134	0.7243±0.1203
E1597802825734	0.5863±0.0365	0.8444±0.0006	0.599±0.025	EF1594632252863	1.04±0.02	0.5268±0.0219	1.0257±0.0284
ECO2007210113	1.8095±0.0895	1.848±0.0303	1.8565±0.0261	EF1594805700879	0.5382±0.0084	0.2864±0.0512	0.5306±0.0107
E1597809878442	0.8437±0.0014	0.6012±0.0207	0.8439±0.0009	EF1594955642012	1.2445±0.0188	0.2488±0.0067	1.2501±0.0182
ECO2007240027	0.9232±0.0242	0.8817±0.02	0.934±0.0198	EF1595231952604	0.661±0.0084	1.3786±0.0033	0.662±0.0047
ECO2007250011	0.894±0.0253	1.1233±0.0101	0.8838±0.0172	EF1595295309474	1.3793±0.0033	0.6656±0.0083	1.3804±0.0037
ECO2007270107	1.1137±0.015	0.9267±0.029	1.1181±0.0121	EF1595295895624	0.6913±0.4229	1.2611±0.0412	0.4748±0.4119
ECO2007290014	1.2188±0.0489	1.2356±0.002	1.203±0.0634	EF1595381687057	0.8865±0.0013	0.5765±0.4527	0.8867±0.0011
ECO2007300009	0.6883±0.2167	0.7975±0.0961	0.6729±0.2626	EF1595817642706	1.2602±0.0628	0.8866±0.0017	1.2697±0.0341
ECO2008110027	0.6429±0.0165	0.6063±0.275	0.6575±0.0411	EF1595474862428	0.2382±0.0	1.0374±0.0113	0.2382±0.0
ECO2008120038	0.5438±0.0	0.8861±0.0293	0.5438±0.0	EF1595650233198	1.0398±0.0092	0.2382±0.0	1.0368±0.0141
ECO2008250134	0.7605±0.19	1.0428±0.0173	0.8168±0.1978	EF1596011846243	1.154±0.0034	0.5829±0.1876	1.1578±0.0064
ECO2008250157	0.7175±0.0022	0.6361±0.036	0.7156±0.0028	EF1596102693660	0.6424±0.0967	1.1582±0.0067	0.5553±0.1946
ECO2008110020	0.7688±0.0007	0.8479±0.0942 (+)	0.7678±0.0028	EF1596466497583	0.8705±0.0154	0.8719±0.0159	0.8756±0.0179
ECO2008160002	0.8826±0.0318	0.7686±0.001	0.8832±0.0161	EF1596525590000	0.2374±0.0039	1.8564±0.5126	0.2376±0.0036
ECO2008120023	0.2452±0.0045 (-)	1.0357±0.0214	0.2518±0.0065	EF1596590704042	1.3016±0.4501	0.2362±0.0039	1.4118±0.5113
CT VS BT (+/-/=)				RRW VS BT (+/-/=)			
2/1/39				1/1/40			

Note: The symbol "+/-" indicates the related method is significantly better/worse than the binary tournament method according to the statistical test. For HV, the larger, the better.

PT) spent solving the packing component across all instances, the average packing time (Avg PT) per instance, and the standard deviation of the packing time (Std PT). Additionally, the table shows the average total CPU time (Avg RT) for running the entire PEAC-HNF algorithm and the proportion of time dedicated to solving the packing component relative to the total runtime (Proportion = Avg PT / Avg RT). The experimental results reveal that the time consumed in solving the 3D packing component accounts for nearly 98% of the algorithm' s total runtime. This can be attributed to the requirement for recalculating the packing arrangement from scratch for each individual (i.e., giant tour). This process is computationally intensive and significantly contributes to the overall runtime.

Table 3.8: CPU time (in seconds) analysis for the 3D packing component of the PEAC-HNF algorithm (2000 FEs, 10 runs). Total PT is the total packing time across all instances; Avg PT is the average packing time across all instances; Std PT is the standard deviation of the packing time; Avg RT is the average total CPU time required for the PEAC-HNF algorithm; Proportion is the ratio of the average CPU time for solving the packing component to the average total CPU time for the PEAC-HNF algorithm (Avg PT / Avg RT).

Datasets	Total PT	Avg PT	Std PT	Avg RT	Proportion
EMO problem instances	169446.728	760.691	496.850	771.371	0.986
Problem instances from [51]	328207.837	631.629	388.435	645.032	0.979
HW problem instances	533484.962	1175.943	707.583	1198.698	0.981

3.3.4 Discussion

In Section 3.3.2, we conducted a comprehensive comparison of our proposed PEAC-HNF method with various state-of-the-art approaches from the literature across multiple datasets, including a total of 242 problem instances. The results reveal the

following key insights:

- PEAC-HNF demonstrates a robust search capability, effectively identifying solutions that optimize both objectives. Additionally, its effectiveness becomes increasingly pronounced as the number of fitness evaluations (FEs) increases, underscoring the method's ability to navigate complex search spaces.
- Compared to the state-of-the-art multi-objective method for solving 3L-SDVRP, MOEA/D-OM [51], our PEAC-HNF yields higher-quality solutions for the majority of problem instances, showcasing its competitive performance.
- Unlike MOEA/D-OM [51], which relies on offline-trained machine learning models that can degrade in performance when faced with instances differing from the training data, our PEAC-HNF exhibits consistently strong performance across diverse problem instances, highlighting its adaptability and robustness.

Section 3.3.3 presents an in-depth examination to investigate the underlying reasons for the superior performance of our proposed PEAC-HNF method compared to other algorithms. To gain deeper insights, we conduct a comparative analysis of PEAC-HNF with its variants, including PEA-C-E, PEA-S-HNF, and PEA-S-E. The results reveal that the incorporation of the novel HNF mutation operator and the concurrent crossover and mutation design in the PEAC-HNF algorithm both contribute significantly to the enhancement of solution quality. Moreover, when these elements are combined, they synergistically improve the overall performance of the algorithm, leading to even better results.

Our experimental results provide the evidences for the effectiveness of PEAC-HNF. However, a limitation of this method is its high computational resource consumption, arising from the need to recalculate the 3D packing arrangement from scratch for each individual (i.e., a giant tour), significantly increasing the overall runtime. This highlights an important direction for future research: developing more efficient

algorithms or heuristics to mitigate this computational overhead.

3.4 Conclusion

This chapter addresses the multi-objective 3L-SDVRP problem and introduces a Hierarchical Neighborhood Filtering (HNF) mutation, characterized by several innovative features: (1) Diverse Offspring Production: It uses multiple neighborhood structures to generate a variety of offspring from a single parent, enhancing solution diversity and exploitation capabiltiy. (2) Hierarchical Mechanism: Unlike traditional random mutations, the HNF mutation prioritizes individuals with higher nondomination ranks, supporting more focused search around higher-performing individuals. (3) Offspring Filtering: This process filters out less promising offspring early, thereby increasing search efficiency and reducing unnecessary fitness evaluations.

We integrated the HNF mutation into the evolutionary algorithm framework to develop the PEAC-HNF algorithm. This algorithm employs various crossover operators for global search and the HNF mutation for intensive exploitation of high-performing individuals, achieving an effective balance between exploration and exploitation. Concurrent execution of crossover and mutation in PEAC-HNF further enhances performance, as confirmed by our extensive experimental evaluations.

Our proposed algorithm has been evaluated and compared to existing algorithms on 242 different problem instances, including real-world instances. While the experimental results confirmed the superiority of our algorithm over others, we note that these 242 commonly used problem instances are relatively small or medium sized. It would be interesting in the future to evaluate our algorithm's performance on large scale problem instances. We also note that our algorithm consumed more CPU time given the same number of FEs, implying some expensive operations (e.g., the packing process) in our algorithm, which needs to be improved in the future.

Chapter 4

An Efficient Local Search Algorithm for 3L-SDVRP*

In the previous chapter, we introduced a global search-based multi-objective evolutionary algorithm for solving the 3L-SDVRP. However, the problem scales addressed were relatively small. Compared to local search-based algorithms, global search-based algorithms typically require more computational resources, particularly when dealing with highly complex combinatorial optimization problems like the 3L-SDVRP on a larger scale. These methods often face challenges in convergence speed and solution quality.

In this chapter, we propose an efficient local search algorithm, building on the state-of-the-art SDVRLH2 [10], to solve large 3L-SDVRPs, and validate its effectiveness through experiments. Section 4.1 presents the relevant background and our research motivations. In Section 4.2, we introduce our algorithm, comprehensively describing its overall framework and key features. In Section 4.3, we present experimental studies, including comparisons with the state-of-the-art algorithm and in-depth analysis of our method. Finally, we conclude this chapter in Section 4.4.

*This chapter is partially based on a paper published at *Memetic Computing* [116].

4.1 Introduction

In Chapter 3, we proposed the PEAC-HNF algorithm and demonstrated through experiments that it outperforms state-of-the-art multi-objective algorithms for solving the 3L-SDVRP. However, our research also revealed several key insights:

- High Computational Resources: Due to the high complexity of the 3L-SDVRP, multi-objective optimization methods relying on global search still consume considerable computational resources, making them unsuitable for large-scale problems.
- Vehicle Count vs. Travel Distance: In real-world scenarios, the number of vehicles is often prioritized over travel distance due to the significantly higher costs associated with maintaining additional vehicles and hiring extra drivers compared to the costs incurred from increased travel distance. As a result, many studies treat the 3L-SDVRP as a single-objective problem, focusing on minimizing the number of vehicles used as the primary objective, with the minimization of total travel distance (ttd) as a secondary objective [10] [13] [14] [50] [66] [76] [107].
- Datasets and Comparability: Although the problem instances used in Chapter 3 are derived from real industrial scenarios, they are relatively small in scale, averaging approximately 9 nodes and 401 boxes. In contrast, several widely used 3L-SDVRP datasets in the literature include the B-Y instances (average of 114 nodes and 5985 boxes) [10], Shanghai instances (average of 40 nodes and 955 boxes) [10], and SD instances (average of 57 nodes and 2371 boxes) [13].

Therefore, our subsequent research focuses on local search-based methods, which can solve large scale problem instances. To ensure better comparability with existing literature, we treat the 3L-SDVRP problem as a single-objective problem and use datasets with large scale instances. SDVRLH2 [10] (depicted in Algorithm 4) is a local

search-based algorithm that maintains the best records for most problem instances in terms of solution quality, establishing it as the state-of-the-art local search-based method for solving the 3L-SDVRP. It consists of two stages: the determination of the packing arrangement for each node and the implementation of local search for the routing component.

However, it has the following weakness:

- The packing algorithm employs a strategy of packing two boxes at a time, which generates only two sub-spaces, potentially leading to wasted space and an increased number of utilized vehicles.
- The routing component of the algorithm utilizes traditional search operators, such as swap, 2-opt, and 3-opt, but fails to exploit problem properties as heuristic information, thus reducing the search efficiency.
- The outer loop, designed to vary the maximal admissible splitting costs, incurs a substantial computational overhead.
- Our analysis indicates that the post-optimization phase, which implements 3-opt, is computationally expensive and not very effective.

Therefore, our research aimed at refining and enhancing the method to solve 3L-SDVRP. Based on SDVRLH2 [10], this chapter proposes a more efficient local search algorithm that incorporates several innovations:

- We improve the box loading and sub-space generation process by two distinct approaches: (a) packing two boxes at once, generating five sub-spaces, and (b) sorting boxes and packing one box at a time, generating three sub-spaces. Both methods generate packing solutions, and a heuristic rule is employed to determine the optimal packing solution. Additionally, the 2C-SP construction method is also adjusted.

Algorithm 4 SDVRLH2 [10]

Input: problem data, parameters;

Output: best solution s_{best}

// N refers to the number of nodes

generate packing solution (1C-FLP, 1C-SP, 2C-SP) for each node
generate initial solution s_{init} and set $s_{best} \leftarrow s_{init}$
for $ips \leftarrow 1$ to nps **do**
    for $inbh \leftarrow 1$ to 2 **do**
        $max\_split\_costs \leftarrow Max\_split\_costs[ips]$
        if $inbh = 1$ **then**
            $nbh\_size = MAX(N/4, 3)$
        else
            $nbu\_size = MAX(N, 3)$
        end if
        $s_{curr} \leftarrow s_{best}$
        for $iter \leftarrow 1$ to $niter$ **do**
            $s_{iter\_best} \leftarrow$ determine best neighbour of s_{curr} by swap operators with range nbh_size
            post-optimize s_{iter_best} by 2-opt
            if $ips = 1$ and $N \leq 100$ **then**
                post-optimize $siter\_best$ by 3-opt
            end if
            update best solution s_{best} by s_{iter_best} where necessary
            $s_{curr} \leftarrow s_{iter\_best}$
        end for
    end for
end for
post-optimize s_{best} by 3-opt
return s_{best}

- For routing, we design three novel search operators that exploit problem properties as heuristic information, resulting in improved efficiency and effectiveness.
- Instead of utilizing an outer loop to control splitting deliveries in SDVRLH2, we propose an adaptive splitting strategy that evaluates the feasibility of splitting at each node based on the packing situation of the vehicle and node, reducing computational overhead.
- Our new post-optimization approach includes two stages: reassigning vehicle loads to minimize the number of utilized vehicles, and independently optimizing the travel distance of each vehicle. This further enhances the quality of the final solution.

4.2 Algorithm Description

In Section 4.1, we introduce the SDVRLH2 algorithm [10] and analyze its weaknesses. In this chapter, building on SDVRLH2 , we propose a new algorithm that incorporates several innovations to address these limitations. This section first gives an overview of our algorithm's framework, and then provides an in-depth discussion of each innovation introduced.

4.2.1 Overview of Our Algorithm

Following SDVRLH2 [10], our algorithm employs a "packing first routing second" strategy. As illustrated in Algorithm 5, this approach first utilizes packing heuristics to generate feasible packing arrangements (Steps 1-2), followed by a local search-based approach for routing optimization. In the routing process, a "giant tour" (also termed as an "individual" in Algorithm 5) acts as the solution's representation. It is a sequence of all nodes but, by itself, does not constitute a feasible solution due

to the impracticality of traversing all nodes in a single route by one vehicle without violating constraints. Therefore, a decoding process is essential to transform a giant tour into a viable routing and packing plan for evaluation. Detailed insights into the giant tour decoding are provided in Section 4.2.6.

Referencing Algorithm 5, initially, we generate a random initial individual as the best individual discovered so far (Step 3). We then conduct iterative exploration based on the current best individual, s_{best} (Steps 4-20), involving multiple iterative searches (Steps 6-19) that are each centered on the current individual, s_{curr}. In each iteration, we employ three proposed search operators: LKH mutation, route-pair swap, and multi-pair elitist recombination, which are described in detail in Section 4.2.3. Specifically, for s_{curr}, we first employ the LKH mutation operator to identify an improved individual, s_{iter_best} (Step 7). Next, we utilize the route-pair swap operator to obtain the neighborhood, N_{swap}, of s_{iter_best} (Step 8). After merging N_{swap} with s_{iter_best} (Step 9), we decode individuals in N and select the best individual among them as the new s_{iter_best} (Step 10). We then apply the multi-pair elitist recombination operator to acquire the corresponding neighborhood, N_{MPER} (Step 11), for superior individuals in N_{swap} (i.e., individuals that outperform s_{iter_best} prior to the route-pair swap operation). We choose the best individual from N_{MPER} and s_{iter_best} as the new s_{iter_best} (Steps 12-13). Following the update of s_{best} and s_{curr} (Step 14), we terminate the current loop if s_{curr} has not been updated for n_{no_imp} consecutive iterations (Step 15). Once the search is completed, the algorithm implements the new post-optimization method on the best individual, s_{best}, for further optimization (Steps 18-20).

Overall, Algorithm 5 extends the hill-climbing approach by conducting exhaustive local searches with various moves from the current best solution, s_{best}. Despite its effectiveness, the algorithm may prematurely converge to local optima. To address this issue, a possible strategy is to accept worsening solutions under certain conditions —akin to those used in simulated annealing [45]. This modification has the potential

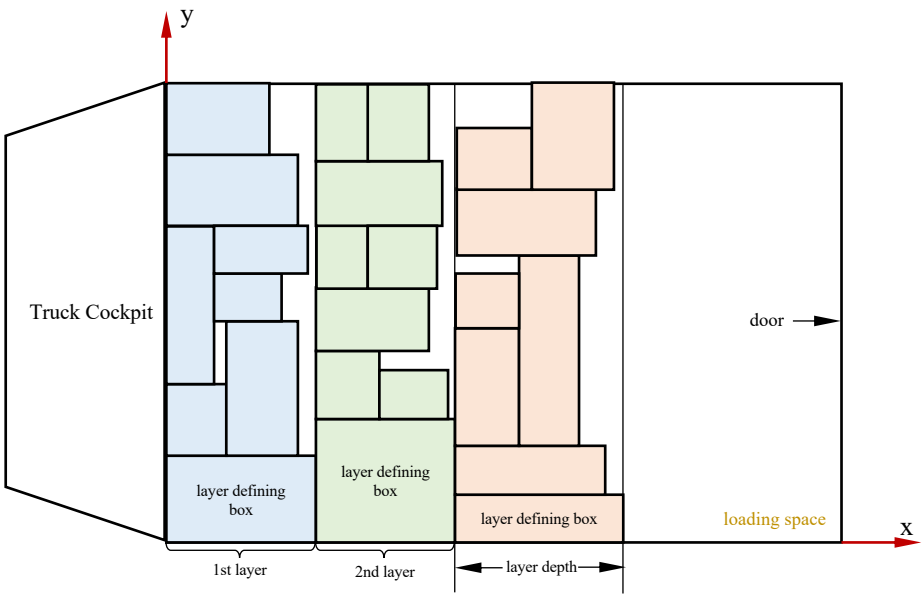


Figure 4.1: Packing plan with three basic layers (top-down view).

not only to diversify the search landscape but also to significantly improve the algorithm' s capability to escape local optima, marking a promising avenue for future research. In the following subsections, we will describe our innovative contributions in detail.

4.2.2 Improved Packing Method

Our layer-based packing strategy, inspired by methods from [7] and [10], organizes boxes at each node into cuboid basic layers as shown in Fig. 4.1. By checking the layer depth against the vehicle's x-axis length, this approach simplifies the solving process. It eliminates the need for frequent 3D packing problem resolutions during routing, thus reducing computational requirements.

Building on the basic layer concept, [10] introduced three packing patterns: One Customer Full Load Pattern (1C-FLP), One Customer Segment Pattern (1C-SP), and Two Customer Segment Pattern (2C-SP). 1C-FLP groups basic layers from a

Algorithm 5 Overview of our method (Key innovations in red boxes.)

Input: problem data, parameters

Output: best solution s_{best}

// Get packing arrangement

1: p_1, p_2 ← generate basic layer, then formulate 1C-FLP and 1C-SP packing pattern and construct 2C-SP (Algorithm 7) by using PM1 and PM2 respectively

2: p ← select packing solution from p_1 and p_2 for routing (Algorithm 6)

// Perform routing process with p

3: generate initial individual s_{init} and set $s_{best} \leftarrow s_{init}$

4: **for** $t \leftarrow 1$ to n_{out} **do**

5: $s_{curr} \leftarrow s_{best}$

6: **for** $iter \leftarrow 1$ to n_{iter} **do**

7: s_{iter_best} ← perform LKH mutation on s_{curr} and get updated solution

8: N_{swap} ← get neighborhood of s_{iter_best} by route-pair swap (Algorithm 8)

9: $N \leftarrow N_{swap} \cup \{s_{iter\_best}\}$

10: s_{iter_best} ← decoding and evaluate individuals in N (including determining each vehicle's packing schedule, splitting arrangement (Algorithm 10), and route), then determine best individual

11: N_{MPER} ← get neighbourhood based on good individuals in N_{swap} by multi-pair elitist recombination (Algorithm 9)

12: $N \leftarrow N_{MPER} \cup \{s_{iter\_best}\}$

13: s_{iter_best} ← decoding and evaluate individuals in N (including determining each vehicle's packing schedule, splitting arrangement (Algorithm 10), and route), then determine best individual

14: update s_{best} by s_{iter_best} where necessary; $s_{curr} \leftarrow s_{iter\_best}$

15: **if** s_{curr} has not been improved for n_{no_imp} consecutive iterations **then** break **endif**

16: **end for**

17: **end for**

// Post-optimization

18: **if** # node ≥ 100 **then**

19: s_{best} ← perform reassignment (Algorithm 11) and single route optimization to s_{best}
else s_{best} ← perform single route optimization to s_{best}

20: **end if**

21: **return** s_{best}

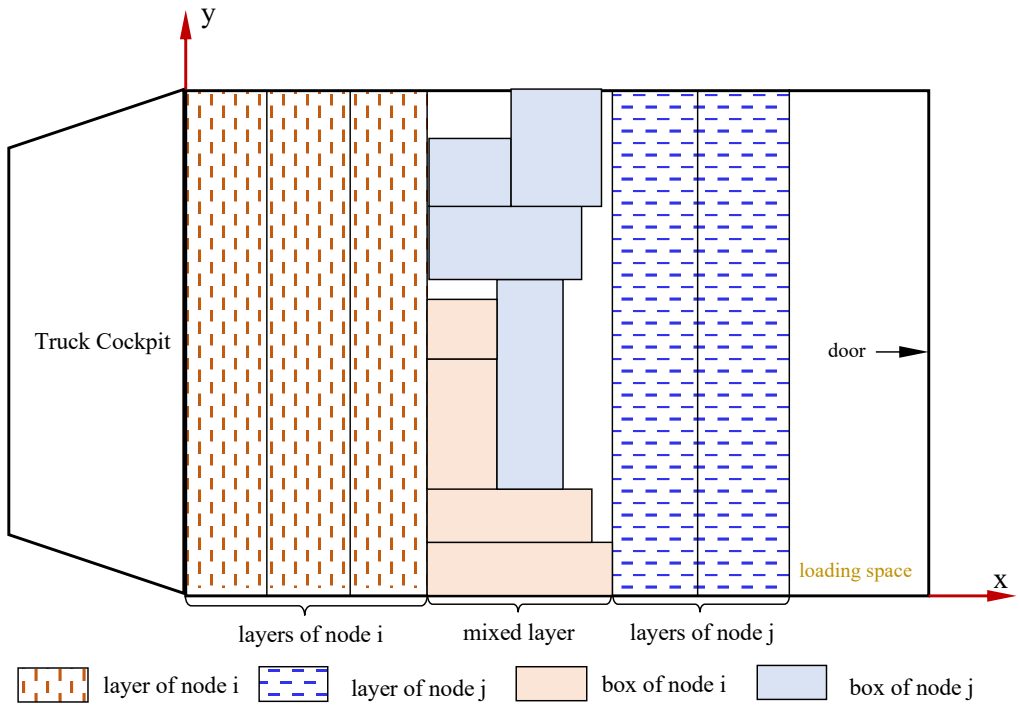


Figure 4.2: Illustration of two customers segment pattern (2C-SP). The mixed layer contains boxes of both nodes.

single node into a complete vehicle load for direct delivery. When a node's layers surpass vehicle capacity, indicating not all layers fit, one or more 1C-FLPs are formed. The remaining basic layers of the node, which do not constitute any 1C-FLP, form a 1C-SP. The 2C-SP integrates 1C-SPs from two distinct nodes to formulate a mixed layer containing boxes from both nodes (as illustrated in Fig. 4.2). This approach strategically selects combinations that maximize space utilization, thereby minimizing the overall volume occupied by the boxes. By merging layers from different nodes, the 2C-SP not only optimizes packing efficiency but also potentially reduces the number of vehicles required for transportation, leading to improved operational efficiency and cost-effectiveness in logistics operations. For comprehensive definitions of these patterns, readers are referred to Section 4.2.6 and [10].

Our enhancements to the packing method focus on two main aspects: refining the

box-packing and space-generation strategy, and advancing the construction technique of the 2C-SP.

Improved Box-Placement and Space-Generation Strategy

As previously mentioned, the basic layer is the smallest vehicle loading unit in our method. Comprehensive details on constructing a basic layer are provided in the Section 4.2.6. In the packing phase, the term "space" denotes a vacant cuboidal area available for box placement. Loading a box into a space means aligning the box's rear-bottom-left vertex with the corresponding corner of the space, fundamentally altering the original space to create new sub-spaces for further loading. This operation is crucial to the packing algorithm, significantly influencing its performance.

In [10], the approach involves loading two boxes (i.e., a box pair) at a time: first, load one box into current space s_{curr}, generate three subspaces, then load the second box into one of the three subspaces, and add the remaining two subspaces to the set of available spaces S_{avail} (note that the second box does not generate new subspaces). If no suitable box pair is found in the set of boxes B_{free}, only one box that can be loaded into s_{curr} will be selected. Through analysis, we believe that the method of loading box pairs has two disadvantages: it generates fewer subspaces, as the second box does not generate subspaces, potentially leading to space wastage; and the selected box pair may not maximize space utilization.

Consequently, we improved the box-packing and space-generation strategy in two directions, resulting in two modified strategies. One enhanced method (denoted as PM1) increases the number of subspaces generated based on [10]. In the original method, two boxes are loaded at a time, with the second box not generating subspaces. Our improved strategy generates subspaces along the axes after loading both boxes, resulting in five subspaces, which enhances space utilization. Another improved method (denoted as PM2) abandons the strategy of loading two boxes at

once, opting instead to load only one box at a time and generating three subspaces along the x, y, and z axes.

The two directions of improvement produce two modified packing methods, each employed to obtain a packing solution (p_1 and p_2). When entering the routing stage, a packing solution is selected based on the criteria we developed and introduced. As shown in Algorithm 6, for each packing solution, first calculate the sum of the depths of all packing patterns (including 1C-FLP, 1C-SP, and 2C-SP) as l_1 and l_2 (Step 1), then divide l_1 and l_2 by the vehicle length l_v to obtain v_1 and v_2 (Step 2). Let $n_{2C-SP}^{(1)}$ and $n_{2C-SP}^{(2)}$ represent the number of 2C-SPs in p_1 and p_2, respectively (Step 3). When $v_2 - v_1 \leq \alpha_1$ or $n_{2C-SP}^{(2)} - n_{2C-SP}^{(1)} \leq \alpha_2$, p_2 is selected as the final packing solution; otherwise, p_1 is chosen (Steps 4-8).

This approach tries to select the most suitable packing solution, taking into account both space utilization and the number of 2C-SPs in each solution. By implementing the two modified strategies, our method effectively improves the overall packing performance while addressing the limitations of the box-pair loading method used in [10].

Multi-stage 2C-SP Construction

As shown in Fig. 4.2, 2C-SPs include box layers from two different nodes and a mixed layer that contains boxes from both nodes. We extend the method for constructing 2C-SPs in SDVRLH2. The original approach allows a node to attempt 2C-SP construction with only some adjacent nodes. After construction, many nodes that could form 2C-SPs may remain. To address this issue, we propose a multi-stage method for 2C-SP construction, as delineated in Algorithm 7. This method considers N as a set of all nodes that have not yet constructed a 2C-SP. It applies the method from [10] to construct 2C-SP for all nodes in N (Step 4). Following a construction cycle, N is updated to only include nodes that have not yet constructed a 2C-SP (Step 7), and

the procedure is reiterated until no further 2C-SPs can be established (Steps 3-7). Compared to a single construction cycle, this method promises a more exhaustive formation of 2C-SPs, thereby further facilitating the reduction in the vehicle count.

Algorithm 6 Select a packing solution (Key innovation in red box.)

Input: p_1, p_2: two packing solutions obtained by the two improved methods PM1 and PM2; l_v: length of vehicle

Output: p_{final}: selected packing solution

1: $l_1, l_2 \leftarrow$ the sum of depth of all packing patterns (1C-FLP, 1C-SP, 2C-SP) in p_1, p_2

2: $v_1 \leftarrow l_1 \ / \ l_v; v_2 \leftarrow l_2 \ / \ l_v$

3: $n^{(1)}_{2C-SP}, n^{(2)}_{2C-SP} \leftarrow$ the number of 2C-SP in p_1, p_2

4: **if** $v_2 - v_1 \leq \alpha_1$ or $n^{(2)}_{2C-SP} - n^{(1)}_{2C-SP} \leq \alpha_2$ **then**

5: $\quad p_{final} \leftarrow p_2$

6: **else**

7: $\quad p_{final} \leftarrow p_1$

8: **end if**

9: **return** p_{final}

4.2.3 New Search Operators

This study highlights the importance of employing superior neighbourhood operators capable of leveraging domain knowledge in local search processes. Commonly used operators in current research, such as the swap operator ([120]), shift operator ([100]), 2-opt ([77]), and 3-opt ([77]), are generally applicable to all VRPs without exploiting the specific characteristics of the problem being studied. Specifically, the swap operator exchanges the positions of two nodes; the shift operator moves a node to another position; the 2-opt operator breaks two edges and reconnects them to alter

Algorithm 7 Multi-stage 2C-SP construction (Key innovation in red box.)

Input: N: the set of all nodes that have not constructed any 2C-SPs

Output: P_{final}: all constructed 2C-SPs

1: $P_{final} \leftarrow \emptyset$

2: **while** True **do**

3: $n_1 \leftarrow |P_{final}|$

4: $P' \leftarrow$ get 2C-SPs from N by the method in [10]

5: $P_{final} \leftarrow P_{final} \cup P'$

6: $n_2 \leftarrow |P_{final}|$

7: **if** $n_2 == n_1$ **then** break **else** update N **end if**

8: **end while**

9: **return** P_{final}

the node order; and the 3-opt operator breaks three edges and reconnects them to create a new sequence of nodes. While these general operators are applicable to a variety of combinatorial optimization problems, they lack the ability to effectively leverage problem-specific characteristics, leading to suboptimal search performance when tackling problems with complex constraints or large-scale instances.

In this thesis, we use the giant tour as the solution representation, which is then decoded into multiple routes and feasible 3D packing plans for each route (i.e., each vehicle). The aforementioned general operators operate on the entire giant tour but typically have small search step sizes and overlook that a giant tour inherently consists of multiple distinct routes. To conduct more effective searches, we leveraged this characteristic and designed three novel search operators: the Lin-Kernighan-Helsgaun (LKH) mutation, the route-pair swap (RPS), and the multi-pair elitist recombination (MPER). The LKH mutation operator optimizes each route obtained after decoding the giant tour. The RPS operator selects two routes from the decoded giant tour and performs node swaps between them. The MPER operator aggregates the effec-

tive swap operations identified during the route-pair swap process to further improve solution quality. The synergistic use of these three operators enables a more efficient search, enhancing both algorithm performance and solution quality. Detailed descriptions of these three operators are as follows.

Lin-Kernighan-Helsgaun (LKH) mutation: Following the decoding and evaluation of the parent giant tour (i.e., parent individual), a set of vehicle routes are obtained, with each route that can be regarded as an independent TSP problem. The TSP is a classical combinatorial optimization problem, and numerous research advancements have led to increasingly mature TSP-solving algorithms. The Lin-Kernighan-Helsgaun (LKH) algorithm ([94]) is among the most effective and cutting-edge heuristic algorithms for solving TSP. The LKH mutation process independently optimizes each route derived from a giant tour using LKH algorithm and subsequently generates an optimized route set.

Route-pair swap (RPS): Algorithm 8 demonstrates the route-pair swap process, which applies a swap operation to nodes of each route in R (Steps 2-10). Specifically, for each route r_1 in R, a random route r_2 is chosen from the remaining routes (Step 3). Then, for each node in r_1, a random node n_2 is selected from r_2 to swap, yielding two new routes, r_1' and r_2' (Steps 5-6). A giant tour g is created as a neighbor based on r_1', r_2', and the remaining routes (Steps 7-8). Finally, a neighbourhood N_{swap} is obtained.

Multi-pair elitist recombination (MPER): This operator is designed to be used in conjunction with RPS operator and is applied after it. As shown in Algorithm 9, after evaluating individuals in N_{swap}, those that outperform the parent individual (i.e., the individual before RPS operator) are selected to form a set S_{good} (Step 2). Subsequently, the algorithm selects as many mutually "compatible" individuals from S_{good} as possible to create S_{compat} (Step 4). The swaps in the S_{compat} individuals are then combined to form a new giant tour, g, as an individual in N_{MPER} (Steps 8-9), and the individuals in S_{compat} are removed from S_{good} (Step 10). The resulting

Algorithm 8 Route-pair swap

Input: a set of routes R

Output: neighbourhood N_{swap}

$N_{swap} \leftarrow \emptyset$
for r_1 in R **do**
    $r_2 \leftarrow$ random select one route from the remaining routes
    for node n_1 in r_1 **do**
        $n_2 \leftarrow$ random select a node from r_2
        $r_1', r_2' \leftarrow$ swap n_1 and n_2 and get two new routes
        $g \leftarrow$ get a giant tour based on r_1', r_2' and remaining routes in R
        $N_{swap} \leftarrow N_{swap} \cup \{g\}$
    end for
end for
return N_{swap}

neighbourhood for this operator is N_{MPER}.

It is important to define the concept of "compatible" individuals. In essence, compatibility between individuals is based on whether their swaps do not interfere with each other, i.e., the nodes involved in different swaps are not adjacent. This is because it is undesirable for the new edges created after a swap to be disrupted by another swap operation. As illustrated in Fig. 4.3, the two swaps in (a) and (b) are compatible, whereas those in (c) are incompatible. This is because when the swaps in (c) are combined, nodes 3 and 1 become adjacent, indicating that one new edge has been disrupted.

Additionally, while RPS and MPER operators alter node positions across routes, there's no need to assess the impact on each route's packing feasibility. This is because the adjusted routes are ultimately merged back into a giant tour, with a feasible solution obtained through giant tour decoding in the subsequent search process.

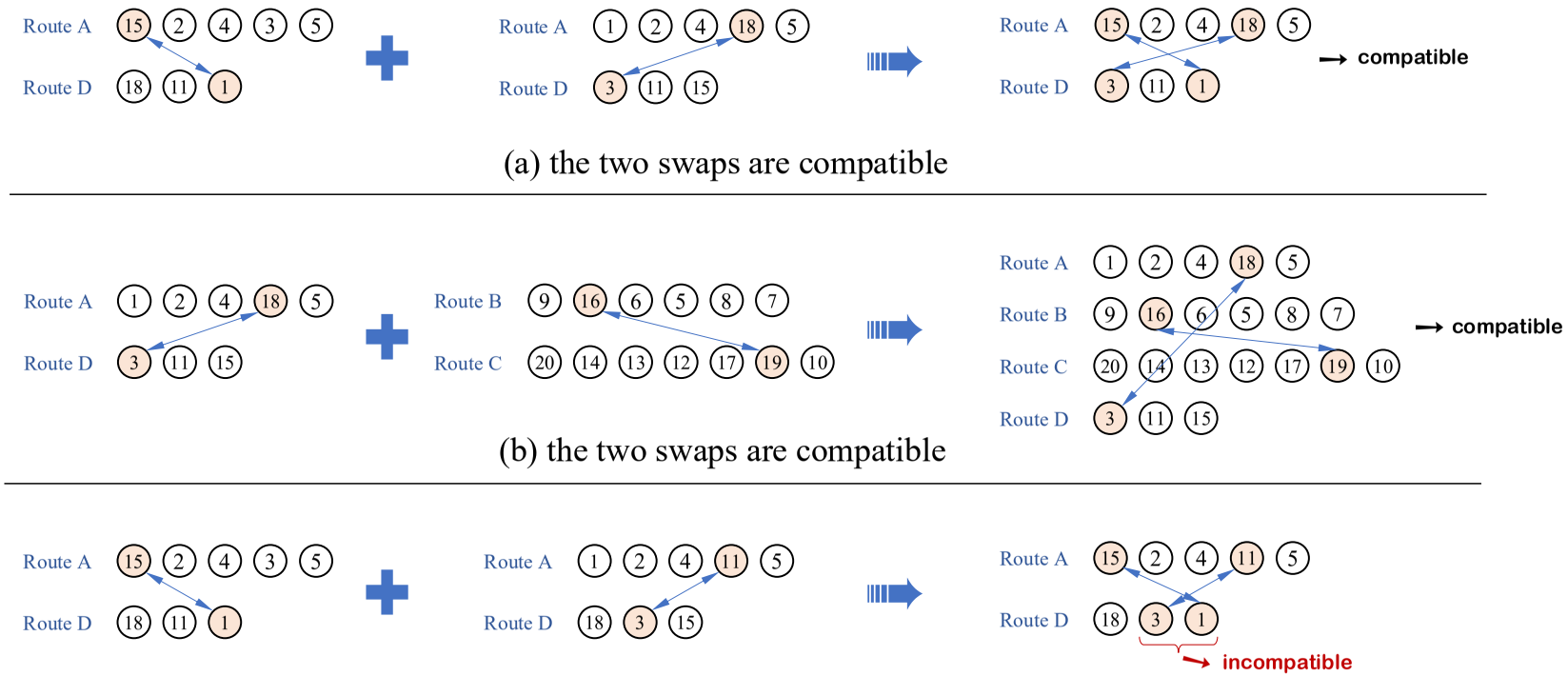


Figure 4.3: Compatible and incompatible swaps.

Algorithm 9 Multi-pair elitist recombination

Input: neighbourhood obtained by route-swap operator N_{swap}

Output: neighbourhood N_{MPER}

1: $N_{MPER} \leftarrow \emptyset$
2: $S_{good} \leftarrow$ select individuals better than the parent individual from N_{swap}
3: **while** $S_{good} \neq \emptyset$ **do**
4: $\quad S_{compat} \leftarrow$ select individuals that are compatible with each other from S_{good}
5: $\quad$ **if** $S_{compat} = \emptyset$ **then**
6: $\quad\quad$ break
7: $\quad$ **end if**
8: $\quad g \leftarrow$ obtain a giant tour by combining the swap of each individual in S_{compat}
9: $\quad N_{MPER} \leftarrow N_{MPER} \cup \{g\}$
10: $\quad S_{good} \leftarrow S_{good} \setminus S_{compat}$
11: **end while**
12: **return** N_{MPER}

4.2.4 Adaptive Splitting Strategy

The 3L-SDVRP incorporates split delivery, a crucial feature that allows boxes from a single node to be loaded onto multiple vehicles. This flexibility significantly influences solution quality, as split decisions directly impact vehicle routing and box loading arrangements.

In [10], the authors used a method where split delivery decisions are integrated into the giant tour decoding process. In this approach, vehicles sequentially visit nodes in the giant tour from front to back. Let n_{curr} represent the current visiting node, n_{next} the subsequent node, and 0 denote the depot. The distance between nodes i and j is given by $d_{i,j}$. When a vehicle cannot accommodate all box layers at n_{next}, it will still visit n_{next} and load as many layers as possible if the following condition is met: $(d_{n_{curr},n_{next}} + d_{n_{next},0})/d_{n_{curr},0} \leq max\_split\_costs$. Here, max_split_costs is

a predefined hyperparameter. This method sets multiple max_split_costs values as the outer loop and performs local searches for each value. However, this approach has two significant limitations:

- It fails to consider the remaining capacity of the vehicle and the box layer configuration at n_{next}. For instance, if the vehicle remaining capacity can only accommodate a minimal number of box layers, visiting n_{next} may be inefficient, as the incurred costs could outweigh the potential benefits.
- Using multiple max_split_costs values requires separate local searches for each value, increasing computational time and complexity.

To address these shortcomings, we propose an adaptive splitting strategy. As illustrated in Algorithm 10, splitting delivery at node i is executed only when the ratio of l_i^v to l_i exceeds a specific threshold p (Steps 5-7). This strategy adaptively determines whether to perform splitting delivery based on the information of the vehicle and node. Additionally, it eliminates the need to traverse Max_split_costs, thereby conserving computational resources and enhancing efficiency. Therefore, our proposed approach:

- Dynamically determines whether to visit the next node based on the vehicle remaining capacity and the box layer configuration at that node.
- Eliminates the need for multiple local searches, thereby reducing computational overhead.

4.2.5 New Post-Optimization Approach

In SDVRLH2, a 3-opt operator is employed for post-optimization. As a commonly used search operator in VRPs, the number of neighbourhood solutions generated by

Algorithm 10 Adaptive splitting strategy

Input: vehicle v, node i

Output: $b_{splitting}$ (True: splitting; False: not splitting)

1: $l_v \leftarrow$ the residual length of v

2: $l_i \leftarrow$ the sum of depths of all layers in i

3: $L_i^v \leftarrow$ select as many layers loaded into v as possible from i based on l_v

4: $l_i^v \leftarrow$ the sum of depths of layers in L_i^v

5: **if** $l_i^v \ / \ l_i < p$ **then** $b_{splitting} \leftarrow False$

6: **else** $b_{splitting} \leftarrow True$

7: **end if**

8: **return** $b_{splitting}$

the 3-opt increases exponentially with the problem size, resulting in high computational complexity. Moreover, due to the lack of heuristic information utilization, the effectiveness of employing 3-opt for post-optimization in this problem is not ideal. Therefore, this chapter designs a new post-optimization method to further improve solution quality.

The proposed post-optimization method consists of two main components: reassignment and routing optimization. The former aims to reassign all layers loaded on a specific vehicle to other vehicles to reduce the vehicle count, while the latter employs our LKH mutation operator to independently optimize each route to minimize the total travel distance.

Reassignment. The overall process of reassignment is demonstrated in Algorithm 11. All routes in the set R are sorted in ascending order based on the sum of their loaded layers' depths (Step 1). The reassignment is prioritized for routes with smaller loadings (Steps 2-12) since they have a higher likelihood of successful reassignment. Specifically, after determining the route r_1 that requires reassignment, the remaining routes in R are sorted in descending order according to the residual length of the

vehicles (Steps 3-4). Then, all layers in r_1 are attempted to be reassigned to the routes in R (Step 5), with the detailed reassignment method explained in Algorithm 12. If all layers in r_1 are successfully reassigned, R is updated and its routes are re-sorted based on the sum of their loaded layers' depths (Steps 7-8), followed by the next iteration; otherwise, the reassignment process stops, and the algorithm ends (Step 10).

Algorithm 11 Framework of Reassignment

Input: a set of routes R with loaded layers

Output: R after reassign

1: $R \leftarrow$ sort R in ascending order based on the sum of the depths of the layers each route carries
2: **for** route $r_1 \in R$ **do**
3: $R \leftarrow R \setminus \{r_1\}$
4: $R \leftarrow$ sort R in descending order based on the residual length of each vehicle
5: $b_{succ}, R_{reassign} \leftarrow$ reassign layers in r_1 to routes in R by Algorithm 12
6: **if** b_{succ} is True **then**
7: $R \leftarrow R_{reassign}$
8: $R \leftarrow$ sort R in ascending order based on the sum of the depths of the layers each route carries
9: **else**
10: break
11: **end if**
12: **end for**
13: **return** R

After identifying route r_a for reassignment, the algorithm reallocates its layers to other vehicles in R. This process involves calculating the maximum available space (rl_{max}) for each route (Algorithm 12 Step 1) and sorting r_a's layers by depth in L_{r_a} (Step 2). The algorithm tries to reassign L_{r_a}'s layers to suitable vehicles in R, marking success

if all layers are reassigned (Steps 3-25). For each layer ℓ in L_{r_a}, it seeks a new route r, assessing the needed additional space (l_{need}, Step 7). If $l_{need} \leq 0$, the layer fits without adjustments. If $l_{need} > rl_{max}$, ℓ cannot be reassigned to r. For $0 \leq l_{need} \leq rl_{max}$, the algorithm might transfer layers from r to accommodate ℓ, selecting the option that minimizes additional required space l_w (Steps 17-18). Failure to reassign any layer from L_{r_a} indicates a reassignment failure, and the original $\boldsymbol{R}$ is retained; otherwise, the successful reassignment potentially reduces the number of needed vehicles.

Single route optimization. After the reassignment process, some layers on certain routes are redistributed to other routes, making it necessary to optimize the routes post-reassignment. In 3L-SDVRP, each customer node can only be visited once by the same vehicle. Consequently, each vehicle departs from the depot, visits a subset of nodes, and returns to the depot. Given this, at the post-optimization phase, where the set of nodes visited by each vehicle is fixed, every vehicle route can be considered an independent TSP problem. Therefore, in this step, we employ our proposed LKH mutation operator to optimize the routes after reassignment, aiming to reduce the overall travel distance.

4.2.6 Other Details

Giant Tour Decoding and Evaluation

In this thesis, we utilize the concept of giant tour to represent a solution for the 3L-SDVRP. A giant tour comprises a permutation of all nodes, but it is not a feasible solution since it is not possible to visit all nodes with a single vehicle without violating any constraints. Therefore, a decoding process is needed to decode a giant tour into a feasible solution. The detailed decoding approach is described in Section 3.2.1.

The 3L-SDVRP has two objectives: minimizing the number of vehicles and the ttd. After decoding, we can calculate the number of vehicles used and the total travel

Algorithm 12 Reassign layers in a route to other routes

Input: r_a: route to be reassigned; R: routes receiving layers from r_a

Output: b_{succ}: a Boolean variable used to indicate the success of the reassignment; R: other routes after reassignment

$rl_{max} \leftarrow$ the maximal residual length of vehicles corresponding to R
$L_{r_a} \leftarrow$ sort layers of r_a in descending order based on layer depths; $b_{succ} \leftarrow True$
for each layer $\ell \in L_{r_a}$ **do**
    $l_\ell \leftarrow$ the depth of ℓ
    $l_{w\_min} \leftarrow$ a big positive number; $s \leftarrow \emptyset$
    for route $r \in R$ **do**
        $l_v \leftarrow$ residual length of the vehicle for route r; $l_{need} \leftarrow l_\ell - l_v$
        if $l_{need} \leq 0$ **then**
            $l_{w1} \leftarrow -l_{need}; l_{w2} \leftarrow 0$
        else if $l_{need} > rl_{max}$ **then** break
        else
            $\mathscr{L}_{remove} \leftarrow$ layers to be removed from r; $l_{re} \leftarrow$ the depth sum of $\mathscr{L}_{remove}$
            if $l_{re} \leq rl_{max}$ **then**
                $l_{w1} \leftarrow l_{re} - l_{need}$; $l_{w2} \leftarrow$ move $\mathscr{L}_{remove}$ to the most suitable route r' and update its vehicle residual length
            end if
        end if
        $l_w \leftarrow l_{w1} + l_{w2}$
        if $l_w < l_{w\_min}$ **then** $l_{w\_min} \leftarrow l_w$; $s \leftarrow$ reassignment arrangement of ℓ **end if**
    end for
    if $s = \emptyset$ **then**
        $b_{succ} \leftarrow False$; **return** b_{succ}, original R
    else
        $r_a, R \leftarrow$ perform reassignment according to s and update r_a and R
    end if
end for
return b_{succ}, R

distance (ttd) of a solution. In line with [10], we prioritize the number of vehicles as the primary objective, while total travel distance serves as a secondary consideration. Therefore, when evaluating the quality of two solutions, we first examine the number of vehicles, with fewer vehicles indicating a superior solution. When the number of vehicles is the same, we then compare the total travel distances, with shorter distances representing better solutions.

Basic Layer and Packing Patterns

The packing algorithm presented in this chapter is inspired by the work of [10], and its overall structure is delineated in Algorithm 13. We adopt a layer-based packing strategy, which has been widely used in various studies [7] [12] [13] [26] [66] [107]. As illustrated in Fig. 4.1, the layer-defining box is the first loaded box of the layer and thus determines the layer's depth. For each node, all boxes form several cuboid layers with varying depths. During the routing search, the cuboid layers serve as the smallest packing units. Consequently, we only need to consider whether the depth of the layers can be accommodated in the vehicle, rather than considering the packing details (i.e., the 3D position and loading sequence) of each box within the cuboid layer. This approach obviates the need to repeatedly address the packing problem during the routing step, focusing instead on the SDVRP, thus reducing computational complexity.

Within this context, there are several key concepts: basic layer, 1C-FLP, 1C-SP, and 2C-SP. A basic layer, as illustrated in Fig. 4.1, is a cuboid layer composed of boxes originating from the same node. Multiple distinct basic layers arise from the boxes of a single node.

Algorithm 14 outlines the packing heuristic method for obtaining all basic layers for a single node. In this method, each layer definition list comprises all unloaded boxes, sorted by their volumes. Initially, the layer defining list of the first layer, $\boldsymbol{LDL}_1$,

Algorithm 13 Overview of our packing approach

Input: problem data, parameters

Output: packing solution

1: generate basic layers for each node by Algorithm 14

2: construct 1C-FLP and 1C-SP for each node based on its basic layers

3: construct 2C-SP pattern based on 1C-SP of each node

4: prepare final packing solution with 1C-FLP, 1C-SP, and 2C-SP

5: **return** packing solution

is generated. Then, for each layer defining box in $\boldsymbol{LDL}_1$, a packing arrangement is produced after completing each outer loop (Algorithm 14 Steps 3-17). To elaborate, for each layer defining box in $\boldsymbol{LDL}_1$, a cuboid layer is created as the first layer (Step 4). Subsequently, numerous candidate layers (stored in $\boldsymbol{L}_{next}$) are generated, and the most suitable one, l_{next}, is selected as the next layer (Steps 6-15). This process involves generating the layer defining list of the next layer, $\boldsymbol{LDL}_n$ (Step 7), followed by the construction of a candidate layer for each layer defining box in $\boldsymbol{LDL}_n$ (Steps 9-12). The best layer from $\boldsymbol{L}_{next}$ is then chosen as the next layer (Step 13). By repeating these steps, various packing arrangements stored in $\boldsymbol{P}$ are obtained. Ultimately, the best packing arrangement is selected as the node's final packing solution. Throughout this process, the packing of different nodes remains independent.

As previously mentioned, the basic layer serves as the smallest unit of vehicle loading in our method. Thus, constructing basic layers from the boxes of each node significantly impacts algorithm performance. To avoid confusion, we introduce the concept of space: as illustrated in Fig. 4.4, a space in this thesis refers to a vacant cuboidal region available for loading boxes. Loading a box into a space entails placing the box such that its rear-bottom-left vertex coincides with that of the space.

Algorithm 15 delineates the process of constructing a basic layer. Initially, an empty layer is determined by the layer defining box b_0 as the first space of the layer (Step

Algorithm 14 Construct basic layers

Input: problem data of one node, parameters

Output: basic layers of the node p

1: $P \leftarrow \emptyset; s \leftarrow \emptyset$
2: $LDL_1 \leftarrow$ generate the layer defining list of the first layer
3: **for** $i \leftarrow 1$ to $|LDL_1|$ **do**
4: $l \leftarrow$ generate a layer with layer defining box $LDL_1[i]$ via Algorithm 15
5: $s \leftarrow s \cup \{l\}$
6: **while** existing unloaded boxes **do**
7: $LDL_n \leftarrow$ generate the layer defining list of the next layer
8: $L_{next} \leftarrow \emptyset$
9: **for** $j \leftarrow 1$ to $|LDL_n|$ **do**
10: $l \leftarrow$ generate a layer with layer defining box $LDL_n[j]$ via Algorithm 15
11: $L_{next} \leftarrow L_{next} \cup \{l\}$
12: **end for**
13: $l_{next} \leftarrow$ select the layer with the best loading rate from L_{next}
14: $s \leftarrow s \cup \{l_{next}\}$
15: **end while**
16: $P \leftarrow P \cup \{s\}; s \leftarrow \emptyset$
17: **end for**
18: $p \leftarrow$ select best packing arrangement
19: **return** p

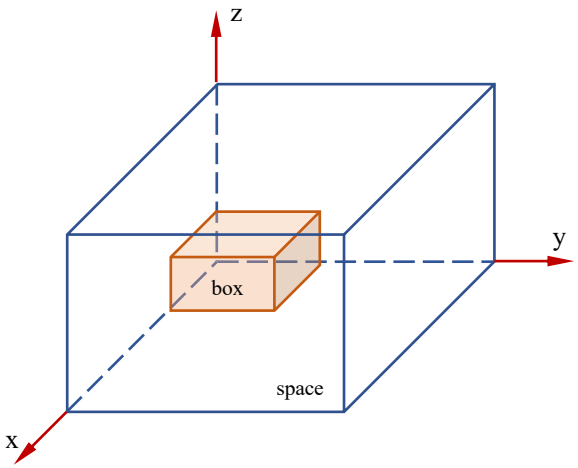


Figure 4.4: Illustration of cuboid space and box.

2). The layer's depth equals the length of b_0 in the x-axis direction, while its width and height correspond to the respective dimensions of the vehicle. After loading b_0, new subspaces are generated along the x, y, and z axes for subsequent loading (Steps 3-4). The following process is repeated until no available spaces remain or all boxes of the node have been loaded: extract spaces sequentially from the space stack S_{avail} (Steps 6-7), then select a box from the set of unloaded boxes B_{free} that can be loaded into the space, load it, and generate new subspaces along the x, y, and z axes (Steps 8-10).

Building upon the basic layer concept, [10] proposed three distinct packing patterns: 1C-FLP, 1C-SP, and 2C-SP. The 1C-FLP, or one Customer Full Load Pattern, is created by combining multiple basic layers from the same node. This pattern requires an empty vehicle to load and directly transport the cargo to the final destination without visiting other nodes. If the sum of the depths of all basic layers of a node exceeds the vehicle capacity, it indicates that a single vehicle cannot accommodate all layers of the node; thus, at least one 1C-FLP can be constructed. After constructing a 1C-FLP for a node, the remaining layers that cannot fill a vehicle (and therefore cannot form a 1C-FLP) constitute a 1C-SP, which stands for one Customer Segment

Algorithm 15 Generate a basic layer

Input: b_0: layer defining box; B_{free}: set of unloaded boxes

Output: a basic layer

1: $S_{avail} \leftarrow \emptyset$
2: $s_0 \leftarrow$ initialize an empty layer as the first space based on b_0 (the length of b_0 in the x-direction corresponds to the depth of the layer)
3: $S_{new} \leftarrow$ load b_0 into s_0 and subsequently generate multiple subspaces
4: $S_{avail} \leftarrow S_{avail} \cup S_{new}$
5: **while** $S_{avail} \neq \emptyset$ and $B_{free} \neq \emptyset$ **do**
6: $\quad s_{curr} \leftarrow$ select the uppermost space from S_{avail}
7: $\quad S_{avail} \leftarrow S_{avail} \setminus \{s_{curr}\}$
8: $\quad b \leftarrow$ select a box that can be loaded in s_{curr} from B_{free}
9: $\quad B_{free} \leftarrow B_{free} \setminus \{b\}$
10: $\quad S_{new} \leftarrow$ load b into s_{curr} and subsequently generate multiple subspaces
11: $\quad S_{avail} \leftarrow S_{avail} \cup S_{new}$
12: **end while**
13: **return** the constructed basic layer

Pattern. As a result, each node will always have one 1C-SP and potentially zero, one, or multiple 1C-FLPs.

Following the construction of 1C-FLP and 1C-SP, the two Customer Segment Pattern (2C-SP) is developed based on the 1C-SP. Initially, a customer pair (c_1, c_2) is selected, and a basic layer is chosen from each of the two customers' 1C-SPs to form a mixed layer. A customer pair may have various mixed layer construction schemes; for each scheme, a saving $\delta = depth(1C-SP(c_1))+depth(1C-SP(c_2))-depth(2C-SP(c_1, c_2))$ is calculated. The construction scheme with the maximum saving δ is selected as the final 2C-SP solution for the customer pair. For additional information on 1C-FLP, 1C-SP, and 2C-SP, please refer to [10].

4.3 Computational Studies

In this study, we have evaluated our algorithm against SDVRLH2 [10], the current state-of-the-art method for the 3L-SDVRP, across three widely-used datasets: B-Y [10], Shanghai [10], and SD instances [13]. These datasets are chosen for two main reasons: (1) *Scale of Problem Instances*: The problem instances used in Chapter 3 are relatively small, averaging approximately 9 nodes and 401 boxes. In contrast, the datasets used in the literature are larger: the B-Y instances average 114 nodes and 5985 boxes, the Shanghai instances average 40 nodes and 955 boxes, and the SD instances average 57 nodes and 2371 boxes. (2) *Facilitation of Comparisons*: Using these widely-used datasets facilitates comparisons with existing work. For clarity, we categorize instances based on the nunmber of nodes: those with fewer than 100 nodes are considered small-scale, while those with 100 to 200 nodes are considered large-scale. The comparative analysis is explained in Section 4.3.2. Additionally, Section 4.3.3 delves into further analyses. This includes examining how different objectives influence outcomes and conducting ablation studies to assess the contribution of our algorithm's novel features.

4.3.1 Experimental Setting

In line with the SDVRLH2 [10] to facilitate comparisons, we prioritize the number of vehicles as the primary objective and total travel distance (ttd) as the secondary objective (with lower priority). Therefore, when evaluating the quality of two solutions, we first examine the number of vehicles, with fewer vehicles indicating a superior solution. When the number of vehicles is equal, we then compare the ttd, with shorter distances representing better solutions. The parameter settings are as follows: in Algorithm 5, $n_{out} = 2, n_{iter} = 100$, and $n_{no\_imp} = 2$; in Algorithm 6, $\alpha_1 = 0.5$ and $\alpha_2 = 0$. It is worth noting that we did not perform a rigorous parameter tuning process because we want to show that the good performance of our algorithm originates from novel algorithm designs, not from finely-tuned parameters. The experiments were conducted 30 times using Python 3.7 on a Dell R370 server with 2x Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz, 128G RAM, and CentOS 7.6 operating system.

Additionally, this chapter provides both CPU time and the number of fitness evaluations (FEs) for SDVRLH2 and our algorithm. While CPU time directly measures computational cost, its reliability is affected by variations in hardware, software, and code efficiency (e.g., differences in coding languages and programming skills). In contrast, the number of FEs, indicating the number of fitness evaluations during the search for an optimal or satisfactory solution, relates directly to the algorithm's time cost and provides a platform-independent metric. Therefore, the FE count provides a more appropriate comparison of algorithm efficiency.

Number of Fitness Evaluations for SDVRLH2

In [10], detailed information on the number of FEs used by SDVRLH2 is absent. Hence, we estimated a lower bound for the FEs based on the algorithm's description and its parameters. To maintain clarity, we adopted the symbols and notation from [10] in our explanation. Symbols nps, nbh, and $niter\_wimpr$ denote the number

of outer iterations, the number of inner iterations, and the iteration threshold for improvement termination (i.e., the iteration terminates if the algorithm does not produce a better solution after $niter\_wimpr$ consecutive iterations.), respectively. n_{swap}, n_{2opt}, and n_{3opt} correspond to the neighborhood sizes of different operators. Thus, the minimal FEs for SDVRLH2 are calculated as: $nps \times nbh \times niter\_wimpr \times (n_{swap} + n_{2opt} + n_{3opt})$.

4.3.2 Comparative Analysis

The experimental results of our proposed algorithm and SDVRLH2 ([10]) are presented in Tables 4.1 and 4.2. The experimental data for SDVRLH2 originates from [10].

Small-scale instances. Tab. 4.1 shows that our algorithm outperforms SDVRLH2 in the average number of vehicles used, achieving better outcomes on 21 instances and similar results on 9 instances. Our proposed algorithm outperforms SDVRLH2 in both average number of vehicles and average ttd on one instance (Sha5), establishing new records for the lowest number of vehicles on 17 instances and the smallest ttd on 2 instances. Notably, our proposed algorithm achieves a maximum reduction of 11 vehicles (SD11) and an 6.36% reduction in ttd (Sha5), contributing to a total reduction of 40.6 vehicles across all instances. Over multiple runs, our algorithm achieved new best-known solutions for the number of vehicles on 17 instances and for both the number of vehicles and ttd on 2 instances. Furthermore, our proposed algorithm requires significantly less CPU time and fewer fitness evaluations (FEs). Across 32 small-scale instances, our algorithm requires, on average, merely 0.18% of the FEs per instance compared to SDVRLH2. Although our algorithm is developed in Python, contrasting with the C-based implementation of SDVRLH2, it still manages to significantly reduce the average CPU time per instance when compared to SDVRLH2.

Large-scale instances. As shown in Tab. 4.2, when applied to 16 larger-scale problem instances, our algorithm significantly reduces the average number of vehicles on 13 instances, with a maximum reduction of 15 vehicles compared to SDVRLH2 (B-Y19). There is a total reduction of 72.6 vehicles across all instances, with an average of 4.5 vehicles per instance. Based on the best results across multiple runs, the proposed algorithm identified new best-known solutions on 12 problem instances in terms of the number of vehicles utilized. The total number of vehicles saved across all large-scale problem instances reached 73, equivalent to an average reduction of 4.6 vehicles per instance. The algorithm also establishes new records by achieving the lowest number of vehicles on 12 instances. Compared to SDVRLH2, our algorithm requires significantly fewer FEs, i.e., computationally more efficient. On all 16 instances, the FEs required by our algorithm never exceed 0.2% of those used by SDVRLH2.

However, our proposed algorithm does increase the ttd, particularly in larger-scale problems, where the substantial reduction in vehicles leads to higher ttd. On small-scale instances, the average ttd increases by 37.79% compared to SDVRLH2. On large-scale instances, the ttd rises by 56.45%. This can be attributed to three factors: (1) The number of vehicles and ttd represent conflicting objectives: an improvement in one leads to a decline in the other. (2) Our method of constructing 2C-SP constructs more 2C-SPs, thereby saving vehicle spaces and reducing the number of vehicles. However, the 2C-SP may be maintained during the route search process if it contributes to reducing the number of vehicles used, potentially increasing the ttd. (3) The algorithm employs a small number of FEs in the route search process. The ttd could be further reduced by allocating additional computing resources to routing. These are the areas for our future studies.

4.3.3 Further Analysis

This section explores three aspects of our algorithm: (1) the effects of changing the

Table 4.1: Comparison results between SD and Ov on small-scale instances (# nodes <100).

Instances (small)	Avg # v	Avg ttd	Best # v	Best ttd	CPU Time	# FE	Instances (small)	Avg # v	Avg ttd	Best # v	Best ttd	CPU Time	# FE
	Ov-SD	(Ov-SD)/SD	Ov-SD	(Ov-SD)/SD	Ov/SD	Ov/SD		Ov-SD	(Ov-SD)/SD	Ov-SD	(Ov-SD)/SD	Ov/SD	Ov/SD
B-Y1	**-1.1**	0.2780	**-2**	0.2141	0.1264	0.0018	Sha9	0	0.2128	0	0.1463	0.1394	0.0027
B-Y2	**-0.5**	0.4953	0	0.4354	0.0882	0.0019	Sha10	**0**	0.1544	**0**	0.0993	0.1331	0.0034
B-Y3	**-1.0**	0.2007	**-1**	0.1249	0.1720	0.0027	Sha11	**-0.2**	0.2208	0	0.1735	0.0376	0.0013
B-Y4	**1.3**	0.5466	1	0.4916	0.1345	0.0019	Sha12	**-2**	0.2075	**-2**	0.1535	0.0641	0.0020
B-Y5	**-2**	0.3155	**-2**	0.2752	0.0627	0.0017	Sha13	**-0.7**	0.2785	**0**	0.2510	0.0182	0.0015
B-Y6	**-0.3**	0.5965	**-1**	0.5636	0.0443	0.0018	SD1	0	0.1749	0	0.1563	1.1004	0.0079
B-Y7	**-2**	0.2363	**-2**	0.1897	0.0710	0.0016	SD2	**-1**	0.2033	**-1**	0.1360	0.3453	0.0036
B-Y8	1.0	0.7038	1	0.6638	0.0514	0.0016	SD3	0	0.1501	0	0.1409	0.0888	0.0034
Sha1	0	0.0238	0	0.0135	2.0054	0.0210	SD4	**-1**	0.4285	**-1**	0.3341	0.1273	0.0026
Sha2	0.2	0.0456	0	0.0239	0.0472	0.0126	SD5	2	0.5586	2	0.5486	0.0618	0.0014
Sha3	0	0.0534	0	0.0132	0.4951	0.0050	SD6	**-2**	0.4721	**-2**	0.4322	0.5147	0.0021
Sha4	0	0.1938	0	0.0718	0.2729	0.0051	SD7	**-2.5**	0.4053	**-3**	0.3213	0.0558	0.0023
Sha5	**-1**	**-0.0636**	**-1**	**-0.0817**	0.1035	0.0083	SD8	**-3**	0.2882	**-3**	0.2431	0.2041	0.0019
Sha6	**-1**	0.3199	**-1**	0.2425	0.0867	0.0035	SD9	**-3**	0.5665	**-3**	0.4841	0.0568	0.0019
Sha7	**-0.7**	0.0600	**-1**	**-0.0071**	0.0967	0.0038	SD10	**-9**	0.3239	**-9**	0.2461	0.0397	0.0019
Sha8	0	0.0494	0	0.0238	0.2648	0.0070	SD11	**-11.1**	0.6625	**-11**	0.5747	0.0238	0.0010

Overall	Avg # v	Avg ttd	Best # v	Best ttd	CPU Time	# FE
Avg	**-1.3**	0.3779	**-1.3**	0.3224	**0.0599**	0.0018
Total	**-40.6**	0.3779	**-42.0**	0.3224	**0.0599**	0.0018

Note: SD refers to the SDVRLH2 algorithm; Ov refers to our algorithm with # v as the first objective. # v = the number of vehicles used; ttd=total travel distance; # FE= the number of fitness evaluations. The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Table 4.2: Comparison results between SD and Ov on large-scale instances (100≤#node≤200).

Instances (large)	Avg # v Ov-SD	Avg ttd (Ov-SD)/SD	Best # v Ov-SD	Best ttd (Ov-SD)/SD	CPU Time Ov/SD	# FE Ov/SD
B-Y9	**-4**	0.4413	**-4**	0.3581	0.2627	0.0015
B-Y10	**-3**	0.6334	**-3**	0.5385	0.1325	0.0016
B-Y11	**-4.1**	0.3295	**-5**	0.2523	0.3184	0.0013
B-Y12	1.0	0.8921	1	0.8323	0.1066	0.0014
B-Y13	**-7.1**	0.4152	**-8**	0.3443	0.5825	0.0012
B-Y14	**-2**	0.6790	**-2**	0.6287	0.1389	0.0013
B-Y15	**-10.5**	0.3515	**-10**	0.2958	0.4684	0.0011
B-Y16	1.1	0.8698	2	0.7213	0.4735	0.0011
B-Y17	**-12.4**	0.3098	**-13**	0.2400	0.7362	0.0008
B-Y18	**-6.6**	0.8257	**-6**	0.7099	0.4009	0.0009
B-Y19	**-14.1**	0.2064	**-15**	0.1631	1.3106	0.0008
B-Y20	**-0.9**	0.8021	0	0.6925	0.5849	0.0009
Sha14	**-2.2**	0.6836	**-2**	0.6144	0.0922	0.0007
Sha15	**-5.0**	0.9501	**-5**	0.8700	0.3990	0.0005
SD12	0	0.3429	0	0.2946	0.1662	0.0015
SD13	**-3**	1.0800	**-3**	1.0135	0.8826	0.0011
Avg	**-4.5**	0.5645	**-4.6**	0.4989	0.4391	0.0009
Total	**-72.6**	0.5645	**-73.0**	0.4989	0.4391	0.0009

Note: SD refers to the SDVRLH2 algorithm; Ov refers to our algorithm with # v as the first objective. # v = the number of vehicles used; ttd=total travel distance; # FE= the number of fitness evaluations. The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

primary objective on performance, (2) the contributions of each novel component through ablation studies, and (3) the efficiency of our algorithm through time analysis. Our experiments use a fixed iteration count, setting a consistent limit on fitness evaluations (FEs). However, Algorithm 5 can terminate early if no improvement is observed after n_{no_imp} consecutive iterations, suggesting a local optimum has been reached. Hence, the actual FEs used may vary across different algorithm configurations, reflecting the distinct search effectiveness of each setting.

Comparative Study of Primary Objectives

In real-world scenarios, reducing the number of vehicles holds greater importance than minimizing total travel distance ([10]), due to the significantly higher costs associated with acquiring, maintaining, and employing drivers for additional vehicles compared to the costs incurred from increased travel distances. Our algorithm, aligned with SDVRLH2, aims to minimize the number of vehicles as the primary objective, with the total travel distance (*ttd*) as the secondary objective. To assess how the objective prioritizations affect algorithmic performance, we have performed experiments comparing the algorithms with the vehicle number (Ov) and *ttd* (Ot) as the primary objective respectively. Tables 4.3 and 4.4 present these results. Note that in algorithm process, solution evaluations are initially based on the primary objective. When solutions have the same primary objective value, the secondary objective then distinguishes their quality.

Small-scale instances. As shown in Tab. 4.3, on 32 small-scale problem instances, in terms of average vehicle counts, Ot underperformed Ov on 10 cases but matched Ov's performance on the remaining instances. Ot utilized 6.5 more vehicles in total compared to Ov. Concerning average *ttd*, Ot exceeded Ov's performance on 11 instances, achieving reductions in *ttd* of up to 5.2%, with an average decrease of 2.2% per instance. Upon analyzing the best results from multiple runs, Ot resulted in a

higher number of vehicles than Ov on 5 instances but equaled Ov's performance in *ttd* across all instances.

Large-scale instance. According to Tab. 4.4, on the 16 large-scale instances, the gap between Ot and Ov widened considerably. Ot needed more vehicles than Ov across the board, with a notable increase on 14 instances, leading to 32.7 more vehicles in total. However, Ot significantly reduced the *ttd*, outperforming Ov by more than 7% on 14 instances and averaging a 6.1% *ttd* reduction. When looking at the best outcomes from multiple runs, Ot underperformed Ov in terms of vehicle count, but matched Ov's *ttd* achievements.

Between Ov and Ot, there is a clear conflict between vehicle count and *ttd*, with vehicle prioritization generally resulting in fewer vehicles and *ttd* prioritization achieving shorter total distances, particularly in larger-scale problems. Whether focusing on minimizing vehicle count (Ov) or reducing *ttd* (Ot), it's challenging to perform well in both objectives simultaneously. From a Pareto dominance aspect, the results emphasizes the inherent difficulty of achieving a balanced optimization between these objectives within a single-objective optimization framework. Thus, our work highlights the ongoing exploration to simultaneously optimize vehicle number and *ttd*, presenting a complex challenge in search of improved overall solutions.

Ablation Study

To evaluate the effectiveness of each innovation in our approach, a comprehensive series of ablation experiments were conducted. Tab. 4.5 presents the algorithm notations employed in these experiments. To assess the impact of each innovation, one innovation was sequentially added to the previous algorithm, followed by 10 repeated runs to obtain experimental results, determining whether the innovation indeed enhanced the algorithm's performance. Tables 4.6 - 4.13 demonstrate the comparative results.

Table 4.3: Comparison results between Ov and Ot on small-scale instances (# nodes <100).

Instances (small)	Avg # v	Avg ttd	Best # v	Best ttd	CPU Time	# FE	Instances (small)	Avg # v	Avg ttd	Best # v	Best ttd	CPU Time	# FE
	Ot-Ov	(Ot-Ov)/Ov	Ot-Ov	(Ot-Ov)/Ov	Ot/Ov	Ot/Ov		Ot-Ov	(Ot-Ov)/Ov	Ot-Ov	(Ot-Ov)/Ov	Ot/Ov	Ot/Ov
B-Y1	0.1	**-0.0197**	1	0	1.353	1.363	Sha9	0.0	**-0.0265**	0	0	1.492	1.582
B-Y2	0.0	**-0.0215**	0	0	1.524	1.417	Sha10	0.0	**-0.0150**	0	0	1.151	1.472
B-Y3	1.0	**-0.0151**	1	0	1.138	1.167	Sha11	0.0	**-0.0153**	0	0	1.222	1.517
B-Y4	0.7	**-0.0400**	1	0	1.535	1.374	Sha12	0.1	**-0.0063**	0	0	1.280	1.382
B-Y5	0.8	**-0.0068**	0	0	1.268	1.338	Sha13	0.6	**-0.0098**	0	0	0.940	1.306
B-Y6	0.3	**-0.0266**	1	0	1.392	1.263	SD1	0.0	**-0.0061**	0	0	1.756	1.403
B-Y7	0.2	**-0.0095**	0	0	1.320	1.488	SD2	0.0	**-0.0319**	0	0	1.837	1.589
B-Y8	0.9	**-0.0198**	0	0	1.325	1.240	SD3	0.0	**-0.0220**	0	0	1.507	1.327
Sha1	0.0	**-0.0066**	0	0	1.594	1.730	SD4	0.2	**-0.0041**	0	0	1.616	1.433
Sha2	**-0.1**	**-0.0084**	0	0	1.846	1.611	SD5	0.0	**-0.0021**	0	0	1.857	1.628
Sha3	0.0	**-0.0126**	0	0	1.304	1.460	SD6	0.0	**-0.0254**	0	0	1.370	1.452
Sha4	0.0	0.0036	0	0	1.846	1.646	SD7	0.5	**-0.0520**	1	0	1.449	1.362
Sha5	0.0	**-0.0012**	0	0	1.778	1.583	SD8	0.0	**-0.0108**	0	0	1.246	1.316
Sha6	0.0	**-0.0134**	0	0	1.618	1.601	SD9	0.3	**-0.0263**	0	0	1.388	1.496
Sha7	0.3	**-0.0114**	0	0	1.341	1.556	SD10	0.6	**-0.0473**	0	0	1.447	1.430
Sha8	0.0	**-0.0105**	0	0	1.475	1.508	SD11	0.0	**-0.0249**	0	0	1.251	1.373

Overall	Avg # v	Avg ttd	Best # v	Best ttd	CPU Time	# FE
Avg	0.2	**-0.0220**	0.16	0	1.322	1.379
Total	6.5	**-0.0220**	5	0	1.322	1.379

Note: Ov (Ot) refers to our algorithm with # v (ttd) as the first objective. # v = the number of vehicles used; ttd=total travel distance; # FE= the number of fitness evaluations. The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Table 4.4: Comparison results between Ov and Ot on large-scale instances (100≤#node≤200).

Instance (large)	Avg # v Ot-Ov	Avg ttd (Ot-Ov)/Ov	Best # v Ot-Ov	Best ttd (Ot-Ov)/Ov	CPU Time Ot/Ov	# FE Ot/Ov
B-Y9	2.0	**-0.1048**	2	0	0.180	1.232
B-Y10	1.3	**-0.0744**	2	0	0.188	1.077
B-Y11	1.7	**-0.0809**	2	0	0.199	1.381
B-Y12	1.9	**-0.0726**	1	0	0.322	1.176
B-Y13	2.9	**-0.1026**	3	0	0.203	1.132
B-Y14	2.0	**-0.0829**	2	0	0.555	1.212
B-Y15	3.1	**-0.0996**	3	0	0.364	1.419
B-Y16	2.5	**-0.1043**	2	0	0.186	1.334
B-Y17	3.6	**-0.0978**	4	0	0.371	1.330
B-Y18	2.4	**-0.1051**	2	0	0.390	1.247
B-Y19	4.2	**-0.1067**	5	0	0.227	1.279
B-Y20	3.2	**-0.1059**	3	0	0.347	1.225
Sha14	1.0	**-0.0857**	1	0	0.128	1.481
Sha15	1.1	**-0.0776**	2	0	0.078	1.269
SD12	0.0	**-0.0286**	0	0	0.186	1.301
SD13	0.0	**-0.0242**	0	0	0.051	1.325
Avg	2.0	**-0.0610**	2.13	0	0.229	1.271
Total	32.7	**-0.0610**	34	0	0.229	1.271

Note: Ov (Ot) refers to our algorithm with # v (ttd) as the first objective. # v = the number of vehicles used; ttd=total travel distance; # FE= the number of fitness evaluations. The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Table 4.5: Notions for algorithms with different components in ablation experiments.

Notation	Algorithm with X	Notation	Algorithm with X
A1	X=Improved packing method	A3	X=A2+Adaptive splitting strategy
A2	X=A1+New search operators	A4	X=A3+New post-optimization

Improved Packing Methods (A1 VS SDVRLH2). As shown in Tables 4.6 and 4.7, on average, on the 32 small-scale instances, A1 reduced the number of vehicles on 20 instances, with a maximum reduction of 11.1 vehicles (SD11) and a cumulative reduction of 43.2 vehicles across all instances. On the 16 large-scale instances, A1 reduced the number of vehicles on 12 instances, with a maximum reduction of 11.2 vehicles (B-Y19) and a cumulative reduction of 51.2 vehicles across all instances. The experimental results indicate that the improved packing methods effectively reduce the number of vehicles, especially on larger-scale problems.

New Search Operators (A2 VS A1). Building upon A1, the old search operators were replaced with our proposed new search operators to obtain Algorithm A2. According to Tables 4.8 and 4.9, for the 32 small-scale instances and the 16 large-scale instances, A2 significantly reduced the total travel distance (*ttd*) on 9 and 16 instances, respectively, compared to A1, with maximum reductions of 6.99% (B-Y8) and 11.09% (SD13). Notably, the new operators exhibited more significant effects on larger-scale instances, reducing the average *ttd* per instance by 6.15%. On an additional 20 small instances, the *ttd* of A2 was found to be as good as that of A1. Importantly, the use of new operators significantly reduced the number of fitness evaluations (FEs) consumed by the algorithm. For the small-scale instances, A2 used only less than 10% of A1's FEs on 28 instances, with a maximum of 35.58% (Sha1) and a minimum of 1.44% (SD11) of A1's usage. For the 16 large-scale instances, A2 used only 20-40% of A1's FEs. This demonstrates that the new operators can greatly improve the search efficiency of the algorithm and help find better solutions.

Table 4.6: Ablation experimental results (A1 VS SD) on small-scale instances (# nodes <100).

Instances (small)	Avg # v	Avg ttd	Best # v	Best ttd	CPU Time	# FE	Instances (small)	Avg # v	Avg ttd	Best # v	Best ttd	CPU Time	# FE
	A1-SD	(A1-SD)/SD	A1-SD	(A1-SD)/SD	A1/SD	A1/SD		A1-SD	(A1-SD)/SD	A1-SD	(A1-SD)/SD	A1/SD	A1/SD
B-Y1	**-1.8**	0.2621	**-2**	0.2019	53.25	9.3636	Sha9	0	0.1404	0	0.1493	21.29	5.1815
B-Y2	**-0.5**	0.3572	0	0.3217	62.04	8.9687	Sha10	0	0.0770	0	0.0791	23.14	7.4412
B-Y3	**-1**	0.1718	**-1**	0.0933	47.16	8.5743	Sha11	**-0.2**	0.1209	0	0.1034	17.85	6.2510
B-Y4	1.1	0.5064	1	0.4446	66.08	8.2154	Sha12	**-2**	0.1333	**-2**	0.1232	18.86	5.5049
B-Y5	**-2**	0.2356	**-2**	0.2149	70.22	15.9788	Sha13	**-0.7**	0.2149	0	0.2106	18.07	10.4225
B-Y6	**-1**	0.5302	**-1**	0.4308	93.59	15.7573	SD1	0	0.0844	0	0.0775	16.86	2.4892
B-Y7	**-2**	0.1720	**-2**	0.1484	63.39	15.7287	SD2	**-1**	0.1118	**-1**	0.0827	27.02	4.6064
B-Y8	1	0.7079	1	0.6746	77.21	13.2315	SD3	0	0.0250	0	0.0194	17.78	7.4335
Sha1	0	0.0135	0	0.0135	151.76	1.3735	SD4	**-1**	0.2706	**-1**	0.2649	38.69	7.4225
Sha2	0	0.0278	0	0.0239	5.22	2.0096	SD5	**2**	0.5496	2	0.5480	35.17	6.3895
Sha3	0	0.0438	0	0.0433	95.87	1.8017	SD6	**-2**	0.4225	**-2**	0.4001	24.97	8.3223
Sha4	0	0.0973	0	0.0718	34.22	2.2397	SD7	**-3**	0.2508	**-3**	0.1689	24.03	9.3373
Sha5	**-1**	**-0.0708**	**-1**	**-0.0887**	8.32	3.1993	SD8	**-3**	0.2652	**-3**	0.2435	20.96	8.0568
Sha6	**-1**	0.2664	**-1**	0.2412	14.89	2.5766	SD9	**-3**	0.4292	**-3**	0.4260	20.38	11.2968
Sha7	**-1**	0.0488	**-1**	0.0156	12.80	2.8038	SD10	**-9**	0.1323	**-9**	0.1089	18.04	13.7202
Sha8	0	0.0196	0	0.0104	18.46	5.1853	SD11	**-11.1**	0.3448	**-11**	0.3372	20.50	17.0319

Overall	Avg # v	Avg ttd	Best # v	Best ttd	CPU Time	# FE
Avg	**-1.4**	0.2176	**-1.3**	0.1939	43.32	12.1300
Total	**-43.2**	0.2644	**-42**	0.2451	43.32	12.1300

Note: # v = the number of vehicles used; ttd=total travel distance; # FE= the number of fitness evaluations. SD refers to the SDVRLH2 algorithm; The meaning of A1 is shown in Tab. 4.5. The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Table 4.7: Ablation experimental results (A1 VS SD) on large-scale instances (100≤#node≤200).

Instances (large)	Avg # v A1-SD	Avg ttd (A1-SD)/SD	Best # v A1-SD	Best ttd (A1-SD)/SD	CPU time A1/SD	# FE A1/SD
B-Y9	**-3**	0.3164	**-3**	0.2479	2.68	1.1103
B-Y10	**-2.7**	0.4860	**-2**	0.4524	1.89	1.1227
B-Y11	**-3**	0.2225	**-3**	0.1956	2.81	1.0722
B-Y12	2	0.7164	2	0.6958	2.03	1.0945
B-Y13	**-5.1**	0.2838	**-6**	0.2689	24.04	1.1049
B-Y14	**-1**	0.5245	**-1**	0.5180	16.70	1.1463
B-Y15	**-8.5**	0.2511	**-8**	0.2115	13.35	1.1068
B-Y16	**2.6**	0.7161	**3**	0.5765	10.09	1.1247
B-Y17	**-9.5**	0.2392	**-10**	0.1901	41.11	1.1234
B-Y18	**-4.9**	0.6911	**-5**	0.5789	28.47	1.1510
B-Y19	**-11.2**	0.1048	**-12**	0.0726	74.49	1.1147
B-Y20	**1.2**	0.6532	2	0.5925	58.18	1.1310
Sha14	**-1.2**	0.4816	**-1**	0.4923	4.55	1.0575
Sha15	**-3.9**	0.8108	**-3**	0.7939	38.58	1.0750
SD12	0	0.0534	0	0.0435	18.05	1.3288
SD13	**-3**	1.0577	**-3**	0.9710	23.50	1.0858
Avg	**-3.2**	0.4755	**-3.1**	0.4313	23.52	1.1198
Total	**-51.2**	0.4098	**-50.0**	0.3693	23.52	1.1198

Note: # v = the number of vehicles used; ttd=total travel distance; # FE= the number of fitness evaluations. SD refers to the SDVRLH2 algorithm; The meaning of A1 is shown in Tab. 4.5. The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Table 4.8: Ablation experimental results (A2 VS A1) on small-scale instances (# nodes <100).

Instances (small)	Avg # v	Avg ttd	Best # v	Best ttd	CPU Time	# FE	Instances (small)	Avg # v	Avg ttd	Best # v	Best ttd	CPU Time	# FE
	A2-A1	(A2-A1)/A1	A2-A1	(A2-A1)/A1	A2/A1	A2/A1		A2-A1	(A2-A1)/A1	A2-A1	(A2-A1)/A1	A2/A1	A2/A1
B-Y1	0.7	**-0.0336**	0	**-0.0155**	0.07	0.0216	Sha9	0	0.0127	0	0.0028	0.13	0.0320
B-Y2	0	0.0104	0	0.0214	0.06	0.0218	Sha10	0	0.0233	0	0.0017	0.07	0.0279
B-Y3	0	**-0.0658**	0	**-0.0114**	0.08	0.0237	Sha11	0	0.0302	0	0.0188	0.05	0.0232
B-Y4	**-0.1**	**-0.0383**	0	**-0.0149**	0.07	0.0238	Sha12	0	0.0169	0	0.0042	0.07	0.0277
B-Y5	0	**-0.0114**	0	**-0.0119**	0.01	0.0173	Sha13	0	0.0067	0	0.0140	0.02	0.0199
B-Y6	0.1	0.0172	0	0.0225	0.01	0.0175	SD1	0	**-0.0080**	0	0.0471	1.62	0.1008
B-Y7	0	**-0.0107**	0	**-0.0176**	0.02	0.0176	SD2	0	**-0.0058**	0	0.0159	0.33	0.0377
B-Y8	0	**-0.0699**	0	**-0.0465**	0.01	0.0207	SD3	0	0.0193	0	0.0194	0.13	0.0269
Sha1	0	0.0000	0	0.0000	0.25	0.3558	SD4	0	**-0.0026**	0	**-0.0207**	0.10	0.0254
Sha2	0	**-0.0030**	0	0.0000	0.23	0.1580	SD5	0	**-0.0008**	0	0.0000	0.08	0.0271
Sha3	0	**-0.0342**	0	**-0.0288**	0.19	0.1267	SD6	0	**-0.0250**	0	**-0.0128**	0.43	0.0260
Sha4	0	**-0.0251**	0	0.0000	0.23	0.0945	SD7	0	**-0.0045**	0	0.0499	0.07	0.0217
Sha5	0	**-0.0139**	0	0.0033	0.24	0.0740	SD8	0	**-0.0201**	0	**-0.0152**	0.23	0.0259
Sha6	0	**-0.0012**	0	**-0.0049**	0.16	0.0687	SD9	0	**-0.0062**	0	0.0012	0.08	0.0203
Sha7	0	**-0.0293**	0	**-0.0236**	0.15	0.0619	SD10	0	0.0212	0	0.0378	0.07	0.0193
Sha8	0	**-0.0143**	0	**-0.0017**	0.21	0.0436	SD11	0	0.0699	0	0.0495	0.05	0.0144

Overall	Avg # v	Avg ttd	Best # v	Best ttd	CPU Time	# FE
Avg	0.02	**-0.0061**	0	0.0026	0.03	0.0191
Total	0.7	0.0022	0	0.0093	0.03	0.0191

Note: # v = the number of vehicles used; ttd=total travel distance; # FE= the number of fitness evaluations. The meaning of A1 and A2 is shown in Tab. 4.5. The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Table 4.9: Ablation experimental results (A2 VS A1) on large-scale instances (100≤#node≤200).

Instances (large)	Avg # v A2-A1	Avg ttd (A2-A1)/A1	Best # v A2-A1	Best ttd (A2-A1)/A1	CPU time A2/A1	# FE A2/A1
B-Y9	0	**-0.0302**	0	**-0.0316**	0.38	0.2963
B-Y10	0	**-0.0304**	0	**-0.0322**	0.35	0.2933
B-Y11	0	**-0.0496**	0	**-0.0462**	0.41	0.3089
B-Y12	0	**-0.0453**	0	**-0.0393**	0.38	0.2996
B-Y13	0.1	**-0.0603**	1	**-0.0585**	0.45	0.3816
B-Y14	0	**-0.0745**	0	**-0.0692**	0.44	0.3679
B-Y15	-0.1	**-0.0784**	0	**-0.0568**	0.85	0.3807
B-Y16	0.4	**-0.0887**	0	**-0.0587**	0.84	0.3741
B-Y17	**-0.5**	**-0.1028**	0	**-0.0751**	0.60	0.4082
B-Y18	**-0.2**	**-0.0896**	0	**-0.0801**	0.48	0.3978
B-Y19	0.2	**-0.0810**	1	**-0.0603**	0.48	0.4103
B-Y20	0.2	**-0.0896**	1	**-0.0769**	0.48	0.4038
Sha14	0	**-0.0574**	0	**-0.0591**	0.24	0.2037
Sha15	0	**-0.0912**	0	**-0.0866**	0.30	0.2895
SD12	0	**0.0952**	0	**0.1003**	0.09	0.2898
SD13	0	**-0.1109**	0	**-0.0694**	0.08	0.3547
Avg	0.01	**-0.0615**	0.2	**-0.0500**	0.43	0.3595
Total	0.1	**-0.0458**	3.0	**-0.0273**	0.43	0.3595

Note: # v = the number of vehicles used; ttd=total travel distance; # FE= the number of fitness evaluations. The meaning of A1 and A2 is shown in Tab. 4.5. The data in bold signifies that our strategy prodsuces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Adaptive Splitting Strategy (A3 VS A2). Algorithm A3 is obtained by incorporating our proposed adaptive splitting strategy into A2. As shown in Tables 4.10 and 4.11, compared to A2, A3 increased the *ttd* while maintaining almost the same number of vehicles, and further significantly reduced the number of FEs. The number of FEs used by A3 accounted for only 0.2-4.09% of those used by A2.

Post-Optimization (A4 VS A3). Algorithm A4 has been developed by integrating an innovative post-optimization technique into Algorithm A3. As demonstrated in Tables 4.12 and 4.13, the results obtained by A4 on most small-scale problem instances show no significant difference from A3. For the 16 large-scale instances, A4 considerably reduce the number of vehicles on 14 instances and the *ttd* on a single instance, resulting in an aggregate reduction of 23.6 vehicles. Further, considering the best results from multiple runs, A4 outperforms A3 on 14 instances for reducing the number of vehicles and on 2 instances for reducing the *ttd*. The experimental results substantiate that the post-optimization method introduced in this study effectively lessens the number of vehicles on larger-scale problems. However, it's crucial to note that reducing the number of vehicles often results in an increase in *ttd*. Moreover, post-optimization for large-scale problems demands considerable time, posing additional challenges.

In summary, the ablation experiments demonstrate the effectiveness and validity of each proposed innovation. The improved packing methods (A1) demonstrated their ability to effectively reduce the number of vehicles, particularly on larger-scale instances. The new search operators (A2) not only further reduced *ttd* on various instances but also significantly improved search efficiency by substantially reducing the number of FEs consumed. The adaptive splitting strategy (A3) managed to maintain a similar number of vehicles while achieving a further reduction in the number of FEs. Lastly, the new post-optimization method (A4) contributed to reducing the number of vehicles on large-scale instances.

Table 4.10: Ablation experiment results (A3 VS A2) on small-scale instances (# node <100).

Instances (small)	Avg # v	Avg ttd	Best # v	Best ttd	# FE	Instances (small)	Avg # v	Avg ttd	Best # v	Best ttd	# FE
	A3-A2	(A3-A2)/A2	A3-A2	(A3-A2)/A2	A3/A2		A3-A2	(A3-A2)/A2	A3-A2	(A3-A2)/A2	A3/A2
B-Y1	0	0.0479	0	0.0292	0.0085	Sha9	0	0.0626	0	0.0209	0.0164
B-Y2	0	0.0983	0	0.0791	0.0102	Sha10	0	0.0487	0	0.0292	0.0166
B-Y3	0.1	0.1074	0	0.0763	0.0130	Sha11	0	0.0623	0	0.0447	0.0090
B-Y4	0.2	0.0797	0	0.0675	0.0095	Sha12	0	0.0714	0	0.0555	0.0119
B-Y5	0	0.0755	0	0.0698	0.0065	Sha13	0	0.0567	0	0.0513	0.0070
B-Y6	0.6	0.0419	0	0.0894	0.0062	SD1	0	0.1004	0	0.0484	0.0281
B-Y7	0	0.0680	0	0.0545	0.0065	SD2	0	0.1045	0	0.0735	0.0207
B-Y8	0.1	0.0805	0	0.0668	0.0059	SD3	0	0.0953	0	0.0982	0.0162
Sha1	0	0.0117	0	0	0.0409	SD4	0	0.1290	0	0.0930	0.0131
Sha2	0.2	0.0259	0	0.0083	0.0411	SD5	0	0.0069	0	0.0004	0.0084
Sha3	0	0.0437	0	0.0103	0.0194	SD6	0	0.0713	0	0.0422	0.0090
Sha4	0	0.1285	0	0	0.0267	SD7	0.6	0.1230	0	0.0766	0.0127
Sha5	0	0.0285	0	0.0044	0.0354	SD8	0	0.0484	0	0.0553	0.0082
Sha6	0	0.0416	0	0.0311	0.0206	SD9	0	0.0978	0	0.0580	0.0096
Sha7	0.2	0.0808	0	0.0603	0.0231	SD10	0	0.1248	0	0.0828	0.0077
Sha8	0	0.0394	0	0.0245	0.0338	SD11	0	0.1559	0	0.1453	0.0041

Overall	Avg # v	Avg ttd	Best # v	Best ttd	# FE
Avg	0.06	0.0737	0	0.0515	0.0158
Total	2.0	0.0891	0	0.0679	0.0078

Note: # v = the number of vehicle used; ttd=total travel distance; # FE= the number of fitness evaluations. The meaning of A2 and A3 is shown in Tab. 4.5. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Table 4.11: Ablation experiment results (A3 VS A2) on large-scale instances (100≤#node≤200).

Instances (large)	Avg # v A3-A2	Avg ttd (A3-A2)/A2	Best # v A3-A2	Best ttd (A3-A2)/A2	# FE A3/A2
B-Y9	0.5	0.0272	0	0.0592	0.0048
B-Y10	0.2	0.1133	0	0.1119	0.0040
B-Y11	0	0.0745	0	0.0649	0.0040
B-Y12	0	0.0912	0	0.0735	0.0044
B-Y13	0.1	0.0693	0	0.0558	0.0026
B-Y14	0	0.1281	0	0.1141	0.0029
B-Y15	0.6	0.0751	0	0.0579	0.0026
B-Y16	0.1	0.0922	1	0.0752	0.0025
B-Y17	0.3	0.0935	0	0.0785	0.0018
B-Y18	0.5	0.0987	1	0.1525	0.0018
B-Y19	0.2	0.0880	**-1**	0.0625	0.0016
B-Y20	**-0.1**	0.1056	**-1**	0.0803	0.0019
Sha14	0	0.1477	0	0.1235	0.0031
Sha15	0	0.1112	0	0.1101	0.0017
SD12	0	0.2068	0	0.1718	0.0035
SD13	0	0.1333	0	0.1124	0.0029
Avg	0.15	0.1035	0.0	0.0940	0.0029
Total	2.4	0.1312	0.0	0.1130	0.0022

Notes: # v = the number of vehicles used; ttd=total travel distance; # FE= the number of fitness evaluations. The meaning of A2 and A3 is shown in Tab. 4.5. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Table 4.12: Ablation experiment results (A4 VS A3) on small-scale instances (# node <100).

Instances (small)	Avg # v	Avg ttd	Best # v	Best ttd	Instances (small)	Avg # v	Avg ttd	Best # v	Best ttd
	A4-A3	(A4-A3)/A3	A4-A3	(A4-A3)/A3		A4-A3	(A4-A3)/A3	A4-A3	(A4-A3)/A3
B-Y1	**-0.9**	0.0942	0	0.0211	Sha9	0	**-0.0122**	0	**-0.0258**
B-Y2	0	**-0.0098**	0	**-0.0020**	Sha10	**-1**	0.0636	**-1**	0.0120
B-Y3	**-0.1**	**-0.0045**	0	**-0.0040**	Sha11	0	**-0.0036**	0	**-0.0007**
B-Y4	**-0.2**	0.0065	0	0	Sha12	0	**-0.0121**	0	**-0.0152**
B-Y5	**-1**	0.0832	**-1**	0.0836	Sha13	**-0.8**	0.0284	**-1**	**-0.0057**
B-Y6	**-0.7**	0.0312	0	**-0.0076**	SD1	0	**-0.0077**	0	0
B-Y7	**-1**	0.0533	**-1**	0.0303	SD2	0	**-0.0091**	0	0
B-Y8	**-0.1**	**-0.0128**	0	**-0.0232**	SD3	0	**-0.0053**	0	**-0.0003**
Sha1	0	0.0000	0	0	SD4	0	**-0.0098**	0	0
Sha2	**-0.2**	**-0.0116**	0	**-0.0056**	SD5	0	**-0.0002**	0	0
Sha3	0	0.0000	0	0	SD6	0	**-0.0168**	0	**-0.0057**
Sha4	0	**-0.0014**	0	0	SD7	**-0.6**	0.0166	0	0.0005
Sha5	0	0.0000	0	0	SD8	0	**-0.0136**	0	**-0.0165**
Sha6	0	**-0.0010**	0	0	SD9	0	**-0.0123**	0	**-0.0175**
Sha7	**-0.2**	**-0.0364**	0	**-0.0557**	SD10	0	**-0.0034**	0	0
Sha8	0	**-0.0012**	0	**-0.0003**	SD11	0	**-0.0032**	0	0

Overall	Avg # v	Avg ttd	Best # v	Best ttd
Avg	**-0.2**	0.0059	**-0.1**	**-0.0012**
Total	**-6.8**	**-0.0020**	**-4**	**-0.0030**

Note: # v = the number of vehicle used; ttd=total travel distance; # FE= the number of fitness evaluations. The meaning of A3 and A4 is shown in Tab. 4.5. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Table 4.13: Ablation experiment results (A4 VS A3) on large-scale instances (100≤#node≤200). # v = the number of vehicles used; ttd=total travel distance; # FE= the number of fitness evaluations. The meaning of A3 and A4 is shown in Tab. 4.5. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Instance (large)	Avg # v A4-A3	Avg ttd (A4-A3)/A3	Best # v A4-A3	Best ttd (A4-A3)/A3
B-Y9	**-1.5**	0.1017	**-1**	0.0888
B-Y10	**-0.3**	0.0064	**-1**	**-0.0156**
B-Y11	**-1.1**	0.0705	**-2**	0.0311
B-Y12	**-1**	0.0708	**-1**	0.0634
B-Y13	**-2.1**	0.0840	**-2**	0.0658
B-Y14	**-1**	0.0529	**-1**	0.0356
B-Y15	**-2.5**	0.0968	**-2**	0.0883
B-Y16	**-2**	0.0898	**-2**	0.0787
B-Y17	**-2.5**	0.0594	**-3**	0.0446
B-Y18	**-2**	0.0884	**-2**	0.0661
B-Y19	**-3.4**	0.1091	**-3**	0.0860
B-Y20	**-2.1**	0.0735	**-2**	0.0723
Sha14	**-1**	0.0628	**-1**	0.0234
Sha15	**-1.1**	0.0748	**-2**	0.0450
SD12	0	**-0.0274**	0	**-0.0280**
SD13	0	**-0.0071**	0	**-0.0096**
Avg	**-1.5**	0.0629	**-1.6**	0.0460
Total	**-23.6**	0.0281	**-25.0**	0.0191

Time Analysis

In real-world industrial scenarios, computational resource consumption is critical, placing higher demands on algorithm efficiency. Since theoretical analysis of computational time complexity is only practical for simple metaheuristic algorithms on artificial problems, we conducted experiments to perform an empirical time analysis. To comprehensively evaluate the efficiency of our proposed algorithm, we compared its CPU runtime with that of the state-of-the-art SDVRLH2 algorithm for the 3L-SDVRP, using results reported in [10].

As shown in Table 4.14, for all small-scale instances, SDVRLH2 required a total of 6627.5 seconds, averaging 207.1 seconds per instance. In contrast, our proposed algorithm, when minimizing the number of vehicles (Ov), took only 397.0 seconds in total, averaging 12.4 seconds per instance—just 6% of the time required by SDVRLH2. When total travel distance was set as the primary objective (Ot), our algorithm took 524.7 seconds in total, averaging 19.8 seconds per instance, which is only 8% of the running time required by SDVRLH2. For all large-scale instances, SDVRLH2 required a total of 30673.1 seconds, averaging 1917.1 seconds per instance. In comparison, our algorithm needed 13468.4 seconds in total, averaging 597.2 seconds per instance when minimizing the number of vehicles (Ov)—only 44% of the time required by SDVRLH2. When total travel distance was the primary objective (Ot), our algorithm required just 3078.0 seconds in total, averaging 192.4 seconds per instance, or only 10% of the CPU time of SDVRLH2. Overall, the runtime of our algorithm was just 37% of SDVRLH2 when optimizing for the number of vehicles (Ov), and only 9% when optimizing for total travel distance (Ot).

Moreover, our algorithm was implemented in Python, whereas SDVRLH2 was implemented in C++. Considering that programs implemented in C++ generally run faster than those in Python, the efficiency advantage of our proposed algorithm is likely even greater than these results suggest. This underscores the high efficiency of

Table 4.14: CPU time analysis (in seconds). Ov (Ot) refers to our algorithm with number of v (ttd) as the first objective. Total = the sum of CPU time across all problem instances; Avg = average CPU time; Std = standard deviation. Small = small problem instances whose number of nodes <100. Large = large problem instances whose number of nodes ≥ 100.

Statistics	SDVRLH2 (SD)		Ov		Ot	
	small	large	small	large	small	large
Total	6627.5	30673.1	397.0	13468.4	524.7	3078.0
Avg	207.1	1917.1	12.4	841.8	16.4	192.4
Std	365.9	441.4	15.4	597.2	19.8	156.1

Overall	Ov/SD			Ot/SD		
	small	large	all	small	large	all
	0.06	0.44	0.37	0.08	0.10	0.09

our method.

4.4 Conclusion

In this chapter, we present an efficient local search algorithm for solving the 3L-SDVRP. Our proposed algorithm encompasses several innovations that contribute to its effectiveness and efficiency. In the packing aspect, we introduce improvements to the box-packing, space-generation, and 2C-SP construction methods, enhancing space utilization and reducing the number of vehicles required. In terms of routing, we propose three novel search operators: LKH mutation, route-pair swap, and multi-pair elitist recombination. These operators leverage the characteristics of the 3L-SDVRP, improving search efficiency. Moreover, we introduce an adaptive splitting strategy

that dynamically determines whether to split nodes based on the loading conditions of vehicles and nodes, thereby further reducing the consumption of computational resources. Finally, we design a new post-optimization method to further improve the solution quality. Comprehensive experimental results and further analysis confirm the efficiency and effectiveness of our proposed method. In particular, compared to the current state-of-the-art algorithm, SDVRLH2 ([10]), our proposed algorithm achieved a considerable decrease in the number of vehicles on most examined problem instances. Furthermore, it achieved a significant reduction in the quantity of fitness evaluations required, decreasing by two orders of magnitude compared to SDVRLH2.

In Chapter 3, our proposed algorithm was tested on problem instances with an average of 9 nodes and 401 boxes per instance. However, it is computationally inefficient for larger problems. For example, solving an instance with 10 nodes and 844 boxes required approximately 1487.5 seconds of CPU time. In contrast, the algorithm presented in this chapter can handle significantly larger problem instances, averaging 76 nodes and 3435 boxes per instance, with a maximum of 200 nodes and 14230 boxes. Our new algorithm significantly reduces computational resource requirements. For instance, solving an instance with 150 nodes and 4508 boxes required only approximately 250.4 seconds of CPU time. This demonstrates that our new method can efficiently tackle larger problems compared to the algorithm presented in Chapter 3.

While our algorithm efficiently reduce the number of vehicles within a much shorter time, it does so at the expense of an increased total travel distance (ttd). By continuing to explore and develop more efficient and effective methods, it may be possible to achieve a better balance between these conflicting objectives, ultimately leading to enhanced solutions for the 3L-SDVRP and similar optimization tasks.

Chapter 5

An Adaptive Interactive Routing-Packing Strategy for 3L-SDVRP*

In Chapters 3 and 4, we introduced the PEAC-HNF algorithm and an efficient local search algorithm to solve the 3L-SDVRP, respectively. The PEAC-HNF algorithm effectively balances two objectives and provides diverse solutions for decision-makers, but it requires significant computational resources for large-scale problems. Conversely, the new local search algorithm proposed in Chapter 4 is more efficient and capable of solving larger-scale problems. Additionally, solving the 3L-SDVRP involves both routing and packing, making the interaction between these components crucial for algorithm performance and solution quality. Current interactive routing-packing strategies have weaknesses, highlighting the need for more effective approaches.

This chapter investigates this interaction and proposes an adaptive interactive routing-packing strategy. The proposed routing-packing strategy can adaptively choose be-

*This chapter is partially based on a paper published at the 2024 Genetic and Evolutionary Computation Conference (GECCO' 24) [115].

tween different packing patterns, providing the flexibility to adjust packing decisions as the individual (i.e., giant tour) changes. Section 5.1 introduces current interactive routing-packing strategies, highlights their weaknesses, and provides our motivation. In Section 5.2, we discuss our proposed adaptive interactive routing-packing strategy, along with other algorithmic details. Section 5.3 details the experimental setup and offers a comprehensive comparison of our method with existing approaches. It also includes further analysis that explains the reasons for the effectiveness of our strategy. Finally, Section 5.4 concludes the chapter.

5.1 Introduction

Fundamentally, the 3L-SDVRP combines two NP-hard problems: the split delivery vehicle routing problem (SDVRP) and the 3D packing/loading problem (3DPP). This combination means that any solution to the 3L-SDVRP must address both routing decisions (determining which nodes each vehicle should visit) and packing decisions (designing the 3D packing plan for each vehicle). These decisions significantly impact the final solution quality, making the interactions between routing and packing during the solution process crucial for algorithm performance. In this chapter, we refer to these interactions as the "interactive routing-packing strategy". The term "interactive" denotes the mutual influence of routing and packing decisions: routing decisions dictate which nodes' boxes a vehicle will load, while packing decisions directly affect the number of boxes that can be loaded within the vehicle's limited capacity.

Prevailing interactive routing-packing strategies are mainly divided into two paradigms: routing first, packing second (R1P2), and packing first, routing second (P1R2). The R1P2 strategy [14] adapts packing decisions during the route search procedure, allowing them to change in response to routing decisions. Conversely, the P1R2 strategy [10] makes packing decisions before routing, thereby reducing the complexity of the routing phase to a direct SDVRP. Additionally, [10] introduced a two-customer seg-

ment pattern (2C-SP) into the P1R2 strategy to further reduce the number of vehicles. The 2C-SP method involves combining two nodes after their packing decisions have been made, and then repacking them. If creating a 2C-SP saves more loading space than packing each node individually, it is maintained.

In the R1P2 strategy, packing decisions are made dynamically during the route planning process. This approach offers considerable flexibility, allowing packing adjustments in response to routing changes. However, this flexibility comes at the cost of increased computational complexity, as each routing adjustment necessitates re-solving the packing problem. Conversely, the P1R2 strategy determines packing decisions prior to route planning, requiring the packing problem to be solved only once, thus significantly speeding up the solution process. The drawback of P1R2 is its rigidity: once packing decisions are made, they cannot adapt to subsequent routing changes. This lack of adaptability can hinder efforts to minimize the number of vehicles used.

Despite the critical role of interactive routing-packing strategies in solving the 3L-SDVRP, there has been no thorough investigation, comparison, or analysis of these strategies. This chapter fills that gap by providing a detailed comparison and analysis of existing interactive routing-packing strategies, supported by extensive experimental validation. Building on insights from the analysis, this chapter introduces an adaptive interactive routing-packing strategy that combines the advanced features of existing approaches. Our strategy introduces adaptability into the routing process, allowing for a choice between independent loading of a node (aligned with the P1R2 strategy) and joint loading of consecutive nodes, utilizing the concept of 2C-SP. This flexible approach permits loading adjustments in response to route modifications, effectively reflecting the core principle of the R1P2 strategy. The effectiveness of our strategy has been rigorously validated through computational experiments. Empirical evidence indicates that our strategy yields solutions that are comparable to or significantly superior to existing strategies, particularly in terms of vehicle count.

5.2 Adaptive Interactive Routing-Packing Strategy

This section begins with a detailed introduction to the proposed adaptive interactive routing-packing strategy, explaining its key features. We then delve into the overall framework of the search algorithm, including how the strategy fits into it. Our proposed routing-packing strategy can be applied to various algorithms for solving the 3L-SDVRP, including those introduced in Chapters 3 and 4.

5.2.1 The Proposed Routing-Packing Strategy

As mentioned in Section 3.2.1, we use the concept of "giant tour" as the representation. Our proposed interactive strategy is designed to operate within the framework of giant tour decoding. In Sections 2.3 and 5.1, we present and analyze the strengths and weaknesses of existing routing-packing strategies. Additionally, in Section 5.3.5, we compare the effectiveness of different packing patterns (i.e., packing two nodes together versus packing a single node alone) across different problem instances. The results indicate that neither pattern is universally superior; their effectiveness depends on the specific instances and nodes. Overall, existing strategies lack the flexibility to adapt to varying situations, such as a vehicle' s remaining space and packing patterns.

Workflow of Our Strategy

The key innovation of our strategy is its ability to make adaptive decisions on different packing patterns (i.e., loading a single node alone or loading two nodes together) based on specific situations, effectively integrating the best features from existing interactive strategies. We have identified criteria that consider various remaining space scenarios of vehicles and the space requirements of different packing patterns at each node. Using these criteria, our strategy adaptively and intelligently chooses between loading a single node independently (following the P1R2 strategy) or loading two

nodes together (using the 2C-SP concept). This decision-making process is tailored to different giant tours. Such flexibility in making loading decisions during the routing process is a fundamental aspect of the R1P2 strategy, allowing for more efficient and adaptable solutions.

The process of our interactive strategy is outlined in Algorithm 16 (red and blue boxes in Algorithm 16 indicate that our strategy can adaptively make decisions between different packing patterns). This strategy employs vertical layers, which is the smallest units of load (Section 2.3.2). The loading feasibility of these layers is assessed by comparing each layer's depth with the vehicle's remaining length. Within our strategy framework, we introduce three key variables: g as the giant tour, l_v as the vehicle's length, and V_f as the set of vehicles generated by decoding g. V_f includes both the routing and loading plans for vehicles.

The algorithm proceeds sequentially through the nodes in g. For each pair of adjacent nodes i and j, we form three sets of layers: L_i, L_j, and L_{ij}. These sets are created by loading i alone, j alone, and both i and j together, respectively (Step 4). We then calculate the total depth for each set, denoted as l_i, l_j, and l_{ij} (Step 5).

The loading process follows some specific criteria identified by us:

(A) If $l_{ij} \leq l_{vr}$ and $l_{ij} < l_i + l_j$, both i and j are loaded together, maximizing space utilization (Steps 6-7).

(B) Special cases arise when either $l_{ij} > l_{vr}$ or $l_{ij} \geq l_i + l_j$. These cases lead to different scenarios:

(B.1) If all layers in L_i can fit in v_{cur}, only i is loaded and v_{cur} is updated (Steps 9-10).

(B.2) If L_i cannot fit into v_{cur}, a new vehicle is introduced (Steps 13-14).

(B.2.a) If $l_{ij} < l_i + l_j$ and can fit in the new vehicle, both nodes are loaded (Steps 15-17).

(B.2.b) If $l_{ij} < l_i + l_j$ but falls between l_v and $2 \times l_v$, the load is split between two new vehicles (Steps 19-23).

(B.2.c) Otherwise, i is loaded into the new vehicle alone (Steps 24-27).

(B.3) If l_{ij} will not fit in v_{cur} but some layers in L_i will, a multi-vehicle approach is considered.

(B.3.a) If $l_{ij} < l_i + l_j$ and could potentially fit into v_{cur} and a new vehicle, we proceed only if two or fewer vehicles are required (Step 32).

(B.3.b) If none of the previous conditions are met, i is loaded individually into v_{cur} and new vehicle(s) (Step 34).

By applying these steps to each node in g, we produce a feasible solution V_f that includes both the routing and loading plans for each vehicle.

Following [10] [76], our research minimizes the vehicle count first, then ttd (with lower priority). Thus, our strategy prioritizes minimizing space occupancy to reduce vehicle usage, which doesn't necessarily reduce ttd. This disconnect arises for two reasons: (1) the relationship between the vehicle count and ttd is complex—lessening one doesn't imply a automatically decrease/increase the other; (2) our approach may favor solutions that load two distant nodes together to save space, beneficial for vehicle reduction but potentially harmful to ttd optimization. Thus, achieving a good balance between two objectives presents a compelling direction for future work.

5.2.2 The Overall Search Algorithm

To facilitate comparisons with existing work, our overall search algorithm employs a local search procedure based on the state-of-the-art SDVRLH2 [10], as detailed in Algorithm 17. The key innovation of Algorithm 17 lies in its use of our proposed adaptive routing-packing strategy, which significantly enhances its performance compared to SDVRLH2. First, the initial random giant tour, s_{init}, serves as the starting

Algorithm 16 Adaptive routing-packing strategy (Colored boxes indicate flexible packing.)

Input: g: a giant tour; l_v: the length of an empty vehicle

Output: V_f: final packed vehicles

$N_{visited} \leftarrow \emptyset$; $V_f \leftarrow \emptyset$; $v_{cur} \leftarrow$ an empty vehicle
for node i in g **do**
  if i in $N_{visited}$ **then** continue **endif**
  $j \leftarrow i+1$; $l_{vr} \leftarrow$ the residual length of v_{cur}; $L_i, L_j, L_{ij} \leftarrow$ get packing layers of node i, j, and node pair (i, j)
  $l_i, l_j, l_{ij} \leftarrow$ the sum of length of layers in L_i, L_j, and L_{ij}
  if $l_{ij} \le l_{vr}$ and $l_{ij} < l_i + l_j$ **then**
    $v_{cur} \leftarrow$ pack i and j by L_{ij} and update v_{cur}; $N_{visited} \leftarrow N_{visited} \cup \{i, j\}$ ▷ pack i and j together
  else // $l_{ij} > l_{vr}$ or $l_{ij} \ge l_i + l_j$
    if $l_i \le l_{vr}$ **then** // L_i can be packed in v_{cur} completely.
      $v_{cur} \leftarrow$ pack i along by L_i and update v_{cur}; $N_{visited} \leftarrow N_{visited} \cup \{i\}$ ▷ pack node i alone
    else
      $l_{i\_min} \leftarrow$ the minimal length of the layer in L_i
      if $l_{i\_min} > l_{vr}$ **then** // None layers in L_i fit in v_{cur}. Use a new vehicle.
        $V_f \leftarrow V_f \cup \{v_{cur}\}$; $v_{cur} \leftarrow$ an empty vehicle; $l_{vr} \leftarrow l_v$
        if $l_{ij} \le l_{vr}$ and $l_{ij} < l_i + l_j$ **then** // L_{ij} can be fully packed in new vehicle.
          $v_{cur} \leftarrow$ pack i and j by L_{ij} and update v_{cur}; ▷ pack i and j together
          $N_{visited} \leftarrow N_{visited} \cup \{i, j\}$
        else
          if $l_{ij} < l_i + l_j$ and $l_{ij} \le 2 * l_v$ **then** // use two new vehicles to pack.
            $V_p \leftarrow$ pack i and j by L_{ij} and get packed vehicles ▷ pack i and j together
            if $len(V_p) \le 2$ **then**
              $V_f \leftarrow V_f \cup V_p$; $v_{cur} \leftarrow$ update v_{cur}; $N_{visited} \leftarrow N_{visited} \cup \{i, j\}$
            end if
          else // only pack node i.
            $v_{cur} \leftarrow$ pack i along by L_i and update v_{cur}; ▷ pack node i alone
            $N_{visited} \leftarrow N_{visited} \cup \{i\}$
          end if
        end if
      else // Part of layers in L_i can be packed in v_{cur}.
        if $l_{ij} < l_i + l_j$ and $l_{ij} < l_{vr} + l_v$ **then** // use v_{cur} and a new vehicle to pack.
          $V_p \leftarrow$ pack i and j by L_{ij} and get packed vehicles ▷ pack i and j together
          if $len(V_p) \le 2$ **then** $V_f \leftarrow V_f \cup V_p$; $v_{cur} \leftarrow$ update v_{cur}; $N_{visited} \leftarrow N_{visited} \cup \{i, j\}$ **end if**
        else // only pack node i.
          $V_p \leftarrow$ pack i along by L_i and get packed vehicles ▷ pack node i alone
          $V_f \leftarrow V_f \cup V_p$; $v_{cur} \leftarrow$ update v_{cur}; $N_{visited} \leftarrow \{i\}$
        end if
      end if
    end if
  end if
end for
return V_f

point, while s_{best} holds the current best solution (Step 1). The swap operator is applied to the current solution, s_{curr}, creating a neighborhood set N_{swap} (Step 5). Each individual of N_{swap} is then decoded using our proposed interactive routing-packing strategy, detailed in Algorithm 16. The most effective solution, $s_{\text{iter\_best}}$, is selected (Step 6). Next, a second neighborhood set, $N_{2\text{opt}}$, is generated by applying the 2-opt operator to $s_{\text{iter\_best}}$ (Step 7). Both $s_{\text{iter\_best}}$ and the individuals of $N_{2\text{opt}}$ are decoded, and the most effective solution replaces $s_{\text{iter\_best}}$ (Steps 8-9). If $s_{\text{iter\_best}}$ outperforms s_{best}, then s_{best} is updated (Step 10). s_{curr} is set to $s_{\text{iter\_best}}$ for the next iteration (Step 11). The search terminates if s_{curr} does not improve after $n_{\text{no\_imp}}$ iterations (Steps 12-14). Upon termination, s_{best} becomes the new s_{curr}, initiating a new round of local search (Steps 3-13).

In relation to packing methodologies, this study adopts heuristic approaches as detailed in [7] [10] [15]. One method is to construct vertical layers, as described in [7] [10]. In this method, a "space" denotes an available cuboidal area for box placement. When a box is put into a space, its back-left-bottom corner matches the same corner of the space. This action consumes the original space and generates new, smaller spaces for future loading. It's evident that the formation of these new spaces after a box insertion significantly impacts the algorithm's efficiency. Let s_{curr} represent the current space under consideration for loading, and S_{avail} represent the set of all available spaces. As noted in [7] [10], two boxes are typically loaded simultaneously. The first box is placed into s_{curr}, creating three new subspaces. A second box is then placed into one of these newly-created subspaces, while the remaining two are added to S_{avail}. Importantly, the second box does not produce additional subspaces. If a suitable pair of boxes cannot be found, a single box that fits within s_{curr} is loaded.

Another prevalent method for box loading relies on Extreme Points (EPs), a notion initially introduced by [15] and subsequently adopted across various studies [6] [13] [26] [92] [103]. EPs are specific points formed by projecting a loaded box along its axes, serving as indicators for available loading space. Each EP indicates the rear-

Algorithm 17 Overall algorithm (Key innovation in red boxes.)

Input: problem instance data

Output: best solution s_{best}

generate initial solution s_{init} and set $s_{best} \leftarrow s_{init}$
for $t \leftarrow 1$ to n_{out} **do**
    $s_{curr} \leftarrow s_{best}$
    for $iter \leftarrow 1$ to n_{iter} **do**
        $N_{swap} \leftarrow$ get neighborhood of s_{curr} by $swap$ operator
        $s_{iter\_best} \leftarrow$ decode individuals by Algorithm 16 and pick best one (with fewest vehicles or lowest ttd for equal vehicles) in N_{swap}
        $N_{2opt} \leftarrow$ get neighborhood of s_{iter_best} by 2-opt operator
        $N \leftarrow N_{2opt} \cup \{s_{iter\_best}\}$
        $s_{iter\_best} \leftarrow$ decode individuals by Algorithm 16 and pick best one (with fewest vehicles or lowest ttd for equal vehicles) in N
        update s_{best} by s_{iter_best} where necessary
        $s_{curr} \leftarrow s_{iter\_best}$
        if s_{curr} has not been improved for n_{no_imp} consecutive iterations **then**
            break
        end if
    end for
end for
return s_{best}

left-bottom vertex of an available space. In EP-based methodologies, the placement of each box generates new EPs, thereby creating new subspaces for future loading. This cycle continues until all boxes are successfully loaded.

5.3 Computational Studies

The experiments were conducted on three benchmark datasets in the field of 3L-SDVRP: the B-Y instances [10], Shanghai instances [10], and SD instances [13]. Instances were divided into two categories: those with fewer than 100 nodes were labeled as small-scale, while those containing 100 to 200 nodes were classified as large-scale datasets. Notably, the SDVRLH2 algorithm [10] has demonstrated exceptional performance across all three datasets, surpassing other methodologies and establishing itself as the state-of-the-art approach. Therefore, to better facilitate comparison with published results, we compared our algorithm to SDVRLH2. This section begins with a comparative evaluation and analysis of current interactive routing-packing strategies (Section 5.3.2). These investigations aid us in discerning the strengths and weaknesses of existing methods, providing both inspiration and insights for the development of new strategies. Following this, we conduct a parameter sensitivity analysis (Section 5.3.4) and compare our proposed method with this leading approach, SDVRLH2, to validate its efficacy (Section 5.3.3). To comprehensively understand our approach's effectiveness, we present an instance-level analysis and interactive strategy comparison highlighting our strategy's significance (Section 5.3.5).

5.3.1 Experimental Setting

The hyperparameters for Algorithm 17 are set as follows: $n_{out} = 4$, $n_{iter} = 100$, $n_{no\_imp} = 2$. Each instance is subjected to 30 runs to ensure a rigorous evaluation. In accordance with cutting-edge research [10], the primary optimization objective is

to minimize the number of vehicles, while the secondary objective aims at reducing the total travel distance (ttd). In other words, when comparing two solutions s_1 and s_2, if the number of vehicles in s_1 is less than in s_2, or if s_1 has the same number of vehicles as s_2 but a lower ttd than s_2, then s_1 is considered better than s_2.

Experiments of our algorithm employ an identical number of iterations, resulting in an equivalent upper limit for the number of fitness evaluations (FEs). However, Algorithm 17 may terminate early if it encounters n_{no_imp} consecutive iterations without improvement (Algorithm 17 Step 12), which suggests entrapment in a local optimum. Thus, different methods may expend varying numbers of FEs within the same upper limit.

Furthermore, it should be noted that the R1P2 strategy demands substantial computational time due to its ongoing resolution of packing problem during routing. Therefore, the number of FEs is not an appropriate metric to reflect its computational demands. In comparisons involving R1P2, CPU time serves as a more accurate indicator. For all other strategies, the number of FEs continues to be the primary measure for assessing time complexity.

For clarity, this chapter presents only the comparative evaluation data among different approaches. Detailed experimental results for each approach are provided in [111]. Experiments were carried out in Python 3.7 on a server equipped with 4x Intel Xeon Platinum 9242 @ 2.30GHz CPU, 256G RAM, and the Ubuntu 20.04 operating system.

5.3.2 Analysis of Current Strategies

P1R2 VS R1P2

Despite the considerable influence of interactive routing packing strategies on problem-solving, a comprehensive comparison and analysis of different strategies are still evidently lacking. This study attempts to fill that gap by providing an extensive exper-

imental examination, focusing on the widely employed P1R2 and R1P2 approaches.

Small Instances: As shown in Tab. 5.1, from the average outcomes over 30 runs, R1P2 exceeded P1R2 regarding vehicle utilization for 23 out of the 32 small-scale instances (instances with a node count less than 100). For the remaining seven instances, no significant difference was observed between the two strategies. Notably, R1P2 reduced vehicle requirements by at least three in three instances and by two or more in six instances when compared to P1R2. Overall, R1P2 employed 67.1 fewer vehicles than P1R2 in the small-scale problem instances, resulting in an average reduction of 2.1 vehicles per instance. Concerning the total travel distance (*ttd*), P1R2 outperformed R1P2 in 22 instances, while no significant difference was detected in four instances. Impressively, P1R2 successfully reduced *ttd* by more than 10% in 12 instances when compared to R1P2, achieving a maximum reduction of up to 35.00%. Considering both optimization objectives—vehicle usage and *ttd*—R1P2 outperformed P1R2 in five instances, while P1R2 demonstrated superior performance in two instances. Looking at the best results from 30 runs, R1P2 achieved a decrease of at least two vehicles compared to P1R2 in 14 instances, and a decrease of three or more vehicles in five instances. In six problem instances, applying both strategies, R1P2 and P1R2, resulted in an identical count of vehicles used. On the other hand, considering *ttd*, P1R2 outperformed by reducing *ttd* by over 10% in 11 instances relative to R1P2, achieving a maximum reduction of up to 21.41%. Moreover, R1P2 demanded significantly higher computational resources than P1R2, using at least four times more resources for all 32 small-scale instances, and exceeding 20 times in 21 instances.

Larger Instances: As demonstrated in Tab. 5.2, when considering the 16 larger problem instances (with the number of nodes ranging between 100 and 200), R1P2 outperformed P1R2 regarding vehicle requirements based on the average results. R1P2 utilized at least three fewer vehicles in 12 instances, with a maximum vehicle savings of up to 6.6. In total, R1P2 required 65.1 fewer vehicles than P1R2, which

Table 5.1: Comparisons between P1R2 and R1P2 (30 runs) on small-scale instances (# node <100).

Instances (small)	Avg # v	Avg ttd	Best # v	Best ttd	CPU Time	Instances (small)	Avg # v	Avg ttd	Best # v	Best ttd	CPU Time
	R-P	(R-P)/P	R-P	(R-P)/P	R/P		R-P	(R-P)/P	R-P	(R-P)/P	R/P
B-Y1	**-1.80**	0.0878	**-2**	0.0777	26.6344	Sha9	**-1.27**	**-0.0546**	**-1**	**-0.0635**	47.5670
B-Y2	**-2**	0.1639	**-2**	0.2063	35.9576	Sha10	**-1.03**	**-0.0056**	**-1**	**-0.0152**	44.7031
B-Y3	**-1**	0.1863	**-1**	0.1619	32.5069	Sha11	**-1.10**	0.0601	**-2**	0.0361	98.0485
B-Y4	**-2.07**	0.1873	**-2**	0.1814	25.7222	Sha12	**-1.67**	0.0188	**-2**	0.0000	56.0697
B-Y5	**-2.50**	0.1094	**-3**	0.0706	25.9907	Sha13	**-3**	**-0.0330**	**-3**	**-0.0535**	116.1346
B-Y6	**-2.03**	0.1870	**-2**	0.2067	41.3536	SD1	0	0.1212	0	0.1012	14.5968
B-Y7	**-0.30**	0.2272	**-1**	0.1851	25.4921	SD2	1	0.1123	1	0.0934	16.4178
B-Y8	**-2.83**	0.3500	**-3**	0.2141	27.6142	SD3	0	0.0620	0	0.0615	14.1937
Sha1	0	0.0547	0	0.0348	6.7945	SD4	**-1**	0.0984	**-1**	0.1272	20.4780
Sha2	1	0.1382	1	0.1367	8.2954	SD5	**-34**	**-0.7474**	**-34**	**-0.7688**	5217.5137
Sha3	0	0.0281	0	**-0.0043**	4.2286	SD6	**-1**	0.0461	**-1**	0.0415	223.0328
Sha4	0	**-0.0694**	0	**-0.0437**	10.9267	SD7	**-0.10**	0.1822	**-1**	0.1881	34.5440
Sha5	**-1**	**-0.1014**	**-1**	**-0.1154**	12.2926	SD8	**-1**	0.0709	**-1**	0.0769	68.0493
Sha6	**-1.27**	**-0.0080**	**-2**	**-0.0585**	15.5658	SD9	**-1.03**	0.0543	**-2**	0.0521	125.9983
Sha7	**-3**	**-0.1363**	**-3**	**-0.1454**	18.6531	SD10	**-1**	0.1617	**-1**	0.0656	121.7826
Sha8	**-0.10**	**-0.0040**	0	**-0.0077**	16.4311	SD11	**-2**	0.0554	**-2**	0.1128	293.2681

Overall	Avg # v	Avg ttd	Best # v	Best ttd	# FE
Avg	**-2.1**	0.0501	**-2.3**	0.0361	213.9643
Total	**-67.1**	**-0.0054**	**-72**	**-0.0208**	115.9932

Note: R refers to R1P2 strategy. P refers to P1R2 strategy. # v = the number of vehicles used; ttd=total travel distance; # FE = the number of fitness evaluations. The data in bold signifies that our strategy produces better results compared to the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

averages to 4.1 fewer vehicles per instance. However, R1P2 underperformed in *ttd*, with P1R2 decreasing *ttd* by over 10% in 14 instances and achieving a maximum reduction of up to 39.77%. Analyzing the best results, R1P2 demonstrated consistently lower vehicle usage than P1R2 across all problem instances, achieving reductions of at least three vehicles in 13 cases. In three instances, R1P2 reduced vehicle usage by five, and in another three, R1P2 managed a maximum saving of seven vehicles. Cumulatively, R1P2 utilized 70 fewer vehicles than P1R2. Considering *ttd*, P1R2 outperformed R1P2, achieving a *ttd* reduction of over 10% in 12 instances and a maximum reduction of up to 32.01% compared to R1P2. Consistent with the observations from the small-scale problems, R1P2 demonstrated a significantly higher demand for computational resources. Across the 16 larger problem instances, the computational resources demanded by R1P2 were at least 20 times those of P1R2, averaging an increase of approximately 67 times per instance.

From the observations, it is clear that the R1P2 strategy, due to its adaptability in addressing the loading sub-problem, significantly reduces the number of vehicles required in comparison to the P1R2 strategy. However, when considering the *ttd*, R1P2 consistently underperforms compared to P1R2. This reveals the intricate features of two objectives that are simultaneously influential and contradictorily conflicting with each other. Furthermore, considering that R1P2 repetitively solves the 3DPP during the routing process, its time complexity is significantly higher than that of P1R2. Empirical evidence demonstrated that, in most problem instances, R1P2 consumed CPU time ranging from dozens to hundreds of times more than P1R2.

Disscussion of The Two Strategies: The order in which routing and packing decisions are made has a fundamental impact on the solution process for the 3L-SDVRP. This ordering determines not only the flexibility of the algorithm in adapting to problem constraints, but also the overall computational burden and the achievable solution quality on different objectives.

In the R1P2 strategy, routing is performed first, and packing is dynamically adapted

Table 5.2: Comparisons between P1R2 (P) and R1P2 (R) on large-scale instances ($100 \leq$ # node ≤ 200). # v = the number of vehicles used; ttd=total travel distance; # FE= the number of fitness evaluations. Our results were obtained from 30 runs. The data in bold signifies that numerically, our strategy produces better results compared to the contrasted strategies. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Instances (large scale)	Avg # v R-P	Avg ttd (R-P)/P	Best # v R-P	Best ttd (R-P)/P	CPU Time R/P
B-Y9	**-3.33**	0.1269	**-4**	0.0999	32.6154
B-Y10	**-3.43**	0.2075	**-4**	0.1796	35.6011
B-Y11	**-0.83**	0.2812	**-1**	0.2216	23.7695
B-Y12	**-3.97**	0.3439	**-4**	0.2542	27.0341
B-Y13	**-4.87**	0.1237	**-5**	0.1245	38.2818
B-Y14	**-5.57**	0.1882	**-5**	0.1092	47.7296
B-Y15	**-1.50**	0.3048	**-2**	0.2438	37.1976
B-Y16	**-5.13**	0.3977	**-5**	0.3201	33.0741
B-Y17	**-6.30**	0.1365	**-7**	0.1180	53.0304
B-Y18	**-6.60**	0.2153	**-7**	0.2109	57.3177
B-Y19	**-1.83**	0.3181	**-3**	0.2705	29.2639
B-Y20	**-6.50**	0.3632	**-7**	0.3381	40.4118
Sha14	**-3.27**	0.0638	**-4**	0.0461	181.0332
Sha15	**-7.60**	0.0031	**-7**	**-0.0327**	239.8252
SD12	**-0.40**	0.1349	**-1**	0.0849	45.6402
SD13	**-4**	0.1002	**-4**	0.1005	149.7439
Avg	**-4.1**	0.2068	**-4.4**	0.1681	66.9731
Total	**-65.1**	0.1654	**-70**	0.1327	50.3070

to each route. This approach maximizes adaptability: as routes change, the packing plan can also be adjusted, which allows the algorithm to better exploit opportunities for split deliveries and improve vehicle utilization. However, the flexibility comes at the expense of increased computational complexity, since every change in routing requires solving new packing subproblems, often making the approach computationally intensive—especially for large instances or strict time budgets.

In contrast, the P1R2 strategy fixes the packing plan before routing begins, effectively reducing the routing problem to a general SDVRP with adjusted loaded layers. While this greatly improves computational efficiency and simplifies the solution process, it also introduces rigidity: once packing decisions are set, the inner packing details of layers cannot adapt to different routes, which may result in suboptimal vehicle usage. In particular, opportunities for effective split delivery can be missed, limiting the flexibility of the solution.

Our experimental results directly reflect these theoretical trade-offs. R1P2 typically achieves better results in minimizing # vehicles, as it can flexibly adjust packing to routing, but often at the cost of higher total travel distance and significantly higher computation time. P1R2 tends to achieve better total travel distances and is much faster, but may require more vehicles due to its inflexibility in adapting packing to routing changes.

In summary, the choice of order between routing and packing is not merely an algorithmic detail but a strategic decision that impacts both the quality and feasibility of solutions for the 3L-SDVRP. In practical applications, the best approach may depend on which objectives are most important (e.g., minimizing vehicles vs. minimizing distance), and the available computational resources. Exploring hybrid or adaptive strategies that dynamically balance these trade-offs could be a promising direction.

Effect of 2C-SP

Bortfeldt and Yi [10] introduced 2C-SP pattern based on the P1R2 strategy, which constructs vertical layers by loading two nodes simultaneously. This research contrasts the integration of P1R2 with 2C-SP, designated as P2, with the conventional application of P1R2 without 2C-SP, labelled as P.

Small Instances: From the average results over 30 runs (Tab. 5.3), P2 required significantly fewer vehicles than P on 20 out of 32 small-scale problem instances. Regarding total travel distance (*ttd*), P2 performed markedly worse than P on 23 instances, with over 10% *ttd* increase on 17 problems. In a cumulative sense, P2 decreased the total number of vehicles utilized across all small-scale problems by 37.67, averaging 1.2 fewer vehicles per instance. However, P2 led to an increase in *ttd* by 11.36% on average per instance. From the best results among the 30 experimental runs, P2 achieved lower vehicle counts than P on 21 instances, cumulatively saving 41 vehicles. In terms of *ttd*, P2 exceeded P by a minimum of 10% on 18 instances, with over 20% increase on ten problems, up to 63.74% maximum.

Larger Instances: As shown in Tab. 5.4, on the 16 larger-scale problem instances, the vehicle reduction achieved by P2 was more substantial. Based on average results across 30 runs, P2 required significantly fewer vehicles than P on 15 instances, with reductions of more than two vehicles observed in 13 problems, up to a maximum decrease of 5.27 vehicles. Compared to P, P2 saved 43.7 vehicles cumulatively on larger instances, averaging a reduction of 2.7 vehicles per instance. Regarding average *ttd*, P2 incurred substantially higher *ttd* than P on 15 cases, with over 10% increase on all problems, peaking at a 34.54% increment. On average, P2 presented a 21.26% higher *ttd* per instance in comparison to P. From the best-of-30 runs, P2 obtained better results than P on 15 instances, with at least two vehicles reduced, up to five maximum. Across all large-scale instances, P2 conserved 46 vehicles in total compared to P, averaging a reduction of 2.9 vehicles per instance. Regarding *ttd*, P2 significantly

Table 5.3: The impacts of 2C-SP on small-scale instances.

Instances (small)	Avg # v	Avg ttd	Best # v	Best ttd	CPU Time	Instances (small)	Avg # v	Avg ttd	Best # v	Best ttd	CPU Time
	P2-P	(P2-P)/P	P2-P	(P2-P)/P	P2/P		P2-P	(P2-P)/P	P2-P	(P2-P)/P	P2/P
B-Y1	**-1**	0.1251	**-1**	0.1475	0.7683	Sha9	**-0.27**	0.0457	0	0.0732	0.7681
B-Y2	**-1**	0.1955	**-1**	0.2841	0.8160	Sha10	**-0.87**	**-0.0168**	0	**-0.0074**	0.7030
B-Y3	**-0.17**	0.0706	**-1**	0.0837	0.9085	Sha11	**-1**	0.0709	**-1**	0.0751	0.6754
B-Y4	**-0.97**	0.1857	0	0.2107	0.7322	Sha12	**-1.23**	0.0797	**-1**	0.1386	0.8020
B-Y5	**-1.37**	0.1528	**-2**	0.1475	0.7018	Sha13	**-1.37**	0.0825	**-2**	0.0920	0.8255
B-Y6	**-1.03**	0.2318	**-1**	0.2851	0.7483	SD1	0	0.1254	0	0.1355	0.9815
B-Y7	**-1.13**	0.1413	**-1**	0.1450	0.6586	SD2	0	0.1055	0	0.1029	0.9117
B-Y8	**-1.17**	0.3343	**-2**	0.3293	0.6467	SD3	0	**-0.0126**	0	0.0073	0.8875
Sha1	0	0.0177	0	0.0135	1.0668	SD4	**-0.67**	0.1401	**-1**	0.1757	0.7558
Sha2	0.1	0.0299	0	0.0498	0.7737	SD5	**-17**	**-0.1231**	**-17**	**-0.1259**	0.7757
Sha3	0	0.0156	0	0.0132	0.9368	SD6	**-1**	0.2380	**-1**	0.2923	0.7200
Sha4	0	**-0.0023**	0	0.0119	0.9944	SD7	**-0.07**	0.1437	**-1**	0.2078	0.9219
Sha5	**-1**	**-0.0584**	**-1**	**-0.0475**	0.8646	SD8	**-0.87**	0.1660	**-1**	0.1934	0.7164
Sha6	**-1**	0.1047	**-1**	0.2110	0.8025	SD9	**-0.93**	0.2807	**-1**	0.2977	0.8104
Sha7	**-0.23**	**-0.0090**	**-1**	0.0014	0.6810	SD10	**-1.33**	0.2976	**-2**	0.3127	0.7071
Sha8	**-0.1**	**-0.0066**	0	**-0.0105**	0.9452	SD11	**-1**	0.4841	**-1**	0.6374	0.9735

Overall	Avg # v	Avg ttd	Best # v	Best ttd	# FE
Avg	**-1.2**	0.1136	**-1.3**	0.1401	0.8119
Total	**-37.67**	0.1548	**-41.0**	0.1821	0.7713

Note: P refers to P1R2 strategy. P2 refers to P1R2 with 2C-SP. # v = the number of vehicle used; ttd=total travel distance; # FE= the number of fitness evaluations.The data in bold signifies that numerically, the P1R2 without 2C-SP produces better results compared to P1R2+2C-SP. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

underperformed in comparison to P, resulting in a 21.26% higher *ttd* per instance on average.

Experimental results show that the 2C-SP pattern enhances box space utilization by co-loading two nodes, consequently leading to a significant reduction in the required number of vehicles. Despite this advantage, the rigid coupling of nodes in 2C-SP presents a drawback: it compromises the minimization of the *ttd*.

Discussion of Current Strategies

From the experimental analysis, we observe that the P1R2 strategy consumes fewer computational resources because it only requires solving the 3D packing problem once per node. However, its packing plan cannot adapt to changes in the giant tour, resulting in lower vehicle loading space utilization and requiring more vehicles compared to the R1P2 strategy. Conversely, the R1P2 strategy adjusts the packing plan as the giant tour changes, thus requiring fewer vehicles. This flexibility, however, reduces computational efficiency, as the 3D packing problem must be resolved for all nodes whenever the giant tour changes, consuming significant computational resources. The 2C-SP pattern (i.e., packing two nodes together) can enhance P1R2's space utilization, but the current approach remains rigid and cannot adapt to giant tour changes. Overall, existing strategies lack the ability to adaptively decide between packing a single node or two nodes together based on varying conditions.

5.3.3 Comparison to State-of-the-Art

To validate the effectiveness of our proposed approach, comparative evaluations are conducted against SDVRLH2 [10] that represents the current state-of-the-art in solving 3L-SDVRP. The comparative outcomes between the two methods are presented in Tables 5.5 and 5.6.

Table 5.4: The impacts of 2C-SP on large-scale instances. P refers to P1R2 strategy. P2 refers to P1R2 with 2C-SP. # v = the number of vehicle used; ttd=total travel distance; # FE= the number of fitness evaluations. The results were obtained from 30 runs. The data in bold signifies that numerically, the P1R2 without 2C-SP produces better results compared to P1R2+2C-SP. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Instances (large-scale)	Avg # v P2-P	Avg ttd (P2-P)/P	Best # v P2-P	Best ttd (P2-P)/P	# FE P2/P
B-Y9	**-2.47**	0.1419	**-2**	0.1408	0.6761
B-Y10	**-2.00**	0.2085	**-3**	0.2513	0.6776
B-Y11	**-1.33**	0.1600	**-2**	0.1636	0.5762
B-Y12	**-2.03**	0.3433	**-3**	0.3472	0.6435
B-Y13	**-4.00**	0.1636	**-4**	0.1850	0.5961
B-Y14	**-2.70**	0.1784	**-3**	0.1953	0.8164
B-Y15	**-1.63**	0.1629	**-2**	0.1597	0.6786
B-Y16	**-2.57**	0.3293	**-2**	0.3550	0.6684
B-Y17	**-5.27**	0.1754	**-5**	0.1833	0.6314
B-Y18	**-3.97**	0.2441	**-4**	0.2840	0.8077
B-Y19	**-2.57**	0.1749	**-3**	0.1789	0.5537
B-Y20	**-3.43**	0.3120	**-4**	0.3252	0.6524
Sha14	**-2.00**	0.1840	**-2**	0.1841	0.7098
Sha15	**-4.80**	0.2759	**-4**	0.3056	0.7928
SD12	0.03	0.0020	0	0.0094	1.0554
SD13	**-3.00**	0.3454	**-3**	0.3832	0.7803
Avg	**-2.7**	0.2126	**-2.9**	0.2282	0.7073
Total	**-43.7**	0.1733	**-46.0**	0.1893	0.6933

Small Instances: As shown in Tab. 5.5, over 30 average runs, our methodology significantly outperformed SDVRLH2 [10] on 22 small instances, achieving superior performance on both vehicles and *ttd* for one problem instance. No significant difference was discerned on two cases. A remarkable reduction of over two vehicles on ten instances, up to 12.1 fewer, was achieved. Vehicle counts were equivalent to SDVRLH2 on eight cases. In total, we decreased vehicles by 57.93 across all small instances compared to SDVRLH2, averaging 1.8 fewer per instance. However, the improvement in *ttd* was notable in only one instance, with no significant difference observed in four others. The best-of-30 runs further illustrated our method's superiority in vehicle counts, achieving better vehicle counts than SDVRLH2 on 22 small instances, tying on one other case. We obtained at least three fewer vehicles on nine problems, with total vehicles reduced by 63 versus SDVRLH2, averaging two fewer per instance. Our best solutions surpassed SDVRLH2 in *ttd* on ten instances.

Larger Instances: Shifting focus to the 16 larger instances (Tab. 5.6), our method consistently reduced vehicles versus SDVRLH2 [10] across all cases. Average runs show a conservation of over three vehicles on nine problems, above five on six, and exceeding eight on three, with a maximum saving of 17.63 vehicles. In total, 86 vehicles were conserved, averaging 5.4 fewer per instance. From the best results, our vehicle counts beat SDVRLH2 on all 16 larger instances. The savings reached five vehicles on eight instances, up to a maximum of 18, with a total reduction of 93 vehicles across the instances, averaging 5.8 fewer per problem.

Evidently, our strategy excels in reducing the number of vehicles, particularly for larger instances, and operates with minimal computational resources. It's important to note, however, that our approach may underperform in terms of *ttd*. This detailed analysis of our method's pros and cons lays the groundwork for future improvements in the 3L-SDVRP field.

Parameter Selection. Hyperparameters significantly impact algorithm performance. In this experiment, we optimized our parameter settings by comparing multi-

Table 5.5: Comparison results between our method (Ours) and SDVRLH2 (S) on small-scale instances (# nodes < 100).

Instances (small)	Avg # v Ours-S	Avg ttd (Ours-S)/S	Best # v Ours-S	Best ttd (Ours-S)/S	# FE Ours/S	Instances (small)	Avg # v Ours-S	Avg ttd (Ours-S)/S	Best # v Ours-S	Best ttd (Ours-S)/S	# FE Ours/S
B-Y1	**-1.70**	0.2471	**-2**	0.1352	**0.0877**	Sha9	**-0.03**	0.1121	**-1**	0.0458	**0.0651**
B-Y2	**-1.47**	0.2750	**-1**	0.1435	**0.0869**	Sha10	0	0.1454	0	0.1002	**0.1080**
B-Y3	**-2.97**	0.2719	**-3**	0.1889	**0.0932**	Sha11	**-0.27**	0.1559	**-1**	0.0775	**0.0528**
B-Y4	**-0.57**	0.3568	**-1**	0.2650	**0.0935**	Sha12	**-2**	0.0449	**-2**	**-0.0148**	**0.0648**
B-Y5	**-2.40**	0.3160	**-3**	0.2422	**0.1151**	Sha13	**-0.37**	0.1366	0	0.0974	**0.0796**
B-Y6	**-1.03**	0.2953	**-2**	0.2168	**0.1111**	SD1	0	0.1595	0	0.1028	**0.0825**
B-Y7	**-4.57**	0.2700	**-5**	0.2075	**0.1218**	SD2	**-1**	0.1178	**-1**	0.0327	**0.0745**
B-Y8	**-1.07**	0.2967	**-2**	0.1840	**0.1168**	SD3	0	0.1870	0	0.0925	**0.0939**
Sha1	0	0.0041	0	0	**0.1296**	SD4	**-1**	0.2621	**-1**	0.1497	**0.0736**
Sha2	0	0.0239	0	**-0.0132**	**0.1002**	SD5	1	0.0846	1	0.0397	**0.0828**
Sha3	0	0.0230	0	**-0.0074**	**0.0493**	SD6	**-3**	0.3358	**-3**	0.2082	**0.1013**
Sha4	0	0.0779	0	**-0.0134**	**0.0582**	SD7	**-3**	0.2256	**-3**	0.0844	**0.0879**
Sha5	**-1**	**-0.1366**	**-1**	**-0.1706**	**0.0919**	SD8	**-4**	0.2665	**-4**	0.1955	**0.0885**
Sha6	**-1**	0.0606	**-1**	**-0.0774**	**0.0487**	SD9	**-3.73**	0.2890	**-4**	0.1404	**0.0993**
Sha7	**-1**	0.0165	**-1**	**-0.0255**	**0.0491**	SD10	**-9.67**	**-0.0081**	**-10**	**-0.0997**	**0.0980**
Sha8	0	0.0131	0	**-0.0190**	**0.1036**	SD11	**-12.10**	0.0679	**-12**	**-0.0517**	**0.0556**

Overall	Avg # v	Avg ttd	Best # v	Best ttd	# FE
Avg	**-1.8**	**0.1561**	**-2.0**	0.0768	**0.0898**
Total	**-57.93**	**0.1790**	**-63**	0.0832	**0.0898**

Note: # v = the number of vehicle used; ttd = total travel distance; # FE = the number of fitness evaluations. The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Table 5.6: Comparison results between our method (Ours) and SDVRLH2 (S) on large-scale instances (100≤#node≤200). # v = the number of vehicle used; ttd = total travel distance; # FE = the number of fitness evaluations. The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Instances (large)	Avg # v Ours-S	Avg ttd (Ours-S)/S	Best # v Ours-S	Best ttd (Ours-S)/S	# FE Ours/S
B-Y9	**-2.60**	0.2809	**-3**	0.1997	**0.1377**
B-Y10	**-2.97**	0.2825	**-3**	0.2306	**0.1318**
B-Y11	**-6.10**	0.2522	**-7**	0.1770	**0.1287**
B-Y12	**-1.03**	0.2976	**-2**	0.2043	**0.1483**
B-Y13	**-4.67**	0.2504	**-5**	0.1982	**0.1517**
B-Y14	**-2.60**	0.2972	**-3**	0.2229	**0.1612**
B-Y15	**-13.37**	0.2225	**-14**	0.1546	**0.1608**
B-Y16	**-2.13**	0.2940	**-2**	0.1577	**0.1697**
B-Y17	**-9.33**	0.1569	**-11**	0.1187	**0.1552**
B-Y18	**-6.73**	0.2937	**-7**	0.2339	**0.1655**
B-Y19	**-17.63**	0.0943	**-18**	0.0425	**0.1574**
B-Y20	**-5.50**	0.2411	**-5**	0.1705	**0.1788**
Sha14	**-1.60**	0.3191	**-2**	0.2863	**0.0684**
Sha15	**-3.93**	0.4114	**-4**	0.3602	**0.0864**
SD12	**-1.07**	0.3662	**-2**	0.3294	**0.1243**
SD13	**-4.73**	0.5354	**-5**	0.4036	**0.1271**
Avg	**-5.4**	0.2872	**-5.8**	0.2181	**0.1425**
Total	**-86.0**	0.3299	**-93**	0.2615	**0.1425**

ple sets for our search algorithm (i.e., Algorithm 17) and selecting the best-performing set. The next section presents a comparison of the different parameter settings.

5.3.4 Parameter Sensitivity Analysis of Our Algorithm

To thoroughly evaluate the impact of varying algorithmic parameters on performance, we experimented with different parameter values in Algorithm 17 and analyzed the outcomes. The configurations are denoted as Ax(a, b, c), where in Algorithm 17, $n_{out} = a$, $n_{iter} = b$, and $n_{no\_imp} = c$. We explored five distinct configurations: A1(2,100,2), A2(4,100,2), A3(4,100,6), A4(6,100,6), and A5(6,200,6).

Detailed comparative results are presented in Tables 5.7, 5.8, 5.9, and 5.10. Our focus was on contrasting A2(4,100,2) with A1(2,100,2), A3(4,100,6) with A2(4,100,2), A4(6,100,6) with A3(4,100,6), and A5(6,200,6) with A4(6,100,6) to discern the effects of incremental parameter increases. A2, compared to A1, exhibited a significant reduction in ttd for 17 small-scale instances, averaging a decrease of 3.66% per instance, with 1.4 times more fitness evaluations (FEs) than A1. In 16 larger-scale instances, A2's ttd was notably lower than A1's on 14 cases, with an average decrease of 3.35% and 1.26 times the FEs. A3 further reduced ttd compared to A2 across 27 small-scale instances, with an average reduction of 5.95% and 2.3 times the FEs. In larger-scale scenarios, A3 consistently surpassed A2 in ttd reduction, averaging 6.94% decrease with 1.97 times the FEs of A2. A5, compared to A4, showed no significant difference in either solution quality or FEs.

These results indicate that while increasing parameter values lead to higher FEs, they also result in lower ttd values. At sufficiently high parameter levels, the algorithm demonstrates improved stability and convergence regarding ttd. The number of vehicles remained remarkably stable across different parameter configurations, underscoring the algorithm's robustness and reliability for practical applications where consistent performance is essential.

Based on these findings, we selected configuration A2 for our comparative analysis with SDVRLH2, achieving a balanced performance with efficient resource utilization.

5.3.5 Further Analysis

This subsection presents an in-depth analysis to explain the reasons behind the effectiveness of our proposed strategy. We conducted further investigations from two primary angles. First, we assessed the strategy's validity in terms of problem-specific characteristics. Second, we conducted interactive strategy comparison to show the benefits of our proposed interactive routing-packing strategy.

Insights from Problem-Specific Characteristics: No Free Lunch

To confirm the reasoning behind our proposed method, we first analyzed it in the context of specific problem instances. Consider any two nodes i and j $(i \neq j)$ within a problem instance, which may be subject to either combined or independent loading. Let l_i, l_j, and l_{ij} represent the space lengths occupied by independently loading i, independently loading j, and loading i and j together, respectively. Let l_v represent the space length of an empty vehicle. We then calculate two indicators, α and β, where $\alpha = l_{ij}/l_v$, $\beta = (l_i + l_j)/l_v$. Based on α and β, an intuitive judgment can be made: if $\alpha < \beta$, it indicates that loading the two nodes together occupies less space; otherwise, it is better to load i and j separately.

For each instance, we compute α and β for node pairs to investigate their distribution and mutual relationship. Figure 5.1 illustrates these variation. For instance, in B-Y1, a significant proportion of node pairs exhibit $\alpha < \beta$, highlighting the advantage of loading two nodes together. Conversely, numerous pairs in B-Y1 also display $\beta < \alpha$, suggesting the benefit of separate loading. In B-Y3, the majority of node pairs demonstrate $\alpha < \beta$, showing the spatial efficiency of paired loading. In SD1 and

Table 5.7: Comparative analysis for our algorithm with different parameter configurations (A1, A2, and A3) on small-scale instances (# nodes < 100).

Instances (small)	A2(4,100,2) VS A1(2,100,2)			A3(4,100,6) VS A2(4,100,2)			Instances (small)	A2(4,100,2) VS A1(2,100,2)			A3(4,100,6) VS A2(4,100,2)		
	Avg # v A2-A1	Avg ttd (A2-A1)/A1	# FE A2/A1	Avg # v A3-A2	Avg ttd (A3-A2)/A2	# FE A3/A2		Avg # v A2-A1	Avg ttd (A2-A1)/A1	# FE A2/A1	Avg # v A3-A2	Avg ttd (A3-A2)/A2	# FE A3/A2
B-Y1	-0.07	-0.0347	1.5053	-0.07	-0.0470	2.3255	Sha9	0	-0.0170	1.5290	-0.10	-0.0294	2.6834
B-Y2	0	-0.0438	1.3734	-0.03	-0.0886	2.2031	Sha10	0	-0.0149	1.4977	0	-0.0263	2.3820
B-Y3	0	-0.0176	1.3204	-0.03	-0.0464	2.2433	Sha11	0	-0.0173	1.4205	0	-0.0360	2.6523
B-Y4	-0.10	-0.0286	1.4380	-0.17	-0.0741	2.3523	Sha12	0	-0.0165	1.4044	0	-0.0272	2.2946
B-Y5	-0.10	-0.0143	1.3314	-0.13	-0.0621	2.3216	Sha13	0	-0.0139	1.3183	-0.17	-0.0361	2.4345
B-Y6	0	-0.0467	1.4256	-0.03	-0.0720	2.1613	SD1	0	-0.0348	1.5660	-0.03	-0.0301	2.5560
B-Y7	-0.13	-0.0223	1.3803	-0.17	-0.0464	2.3488	SD2	0	-0.0325	1.5011	0	-0.0325	2.3263
B-Y8	0	-0.0347	1.3228	-0.07	-0.0666	2.1163	SD3	0	-0.0261	1.4714	0	-0.0476	2.3715
Sha1	0	-0.0108	1.7061	0	-0.0040	2.6401	SD4	0	-0.0298	1.4071	0	-0.0677	2.4233
Sha2	0	-0.0042	1.5527	0	-0.0019	2.5328	SD5	0	-0.0226	1.3745	0	-0.0179	2.0366
Sha3	0	-0.0082	1.5503	0	-0.0182	2.5979	SD6	0	-0.0440	1.4198	0	-0.0721	2.2394
Sha4	0	-0.0366	1.5660	0	-0.0300	2.6025	SD7	-0.03	-0.0552	1.4294	0	-0.0826	2.2448
Sha5	0	-0.0081	1.5703	0	-0.0145	2.6286	SD8	0	-0.0417	1.4570	0	-0.0810	2.3610
Sha6	0	-0.0180	1.5889	0	-0.0261	2.5785	SD9	-0.07	-0.0583	1.4277	-0.17	-0.0714	2.0906
Sha7	0	-0.0179	1.5590	0	-0.0201	2.4934	SD10	-0.03	-0.0450	1.4875	-0.13	-0.0778	2.3593
Sha8	0	-0.0148	1.5985	0	-0.0220	2.5557	SD11	0	-0.0368	1.4889	0	-0.0730	2.4590

Overall	A2(4,100,2) VS A1(2,100,2)			A3(4,100,6) VS A2(4,100,2)		
	Avg # v	Avg ttd	# FE	Avg # v	Avg ttd	# FE
Avg	-1.2	0.1136	0.8119	-1.2	0.1136	0.8119
Total	-37.67	0.1548	0.7713	-37.67	0.1548	0.7713

Note: # v = the number of vehicle used; ttd = total travel distance; # FE = the number of fitness evaluations. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$. The results were obtained from 30 runs.

Table 5.8: Comparative analysis for our algorithm with different parameter configurations (A1, A2, and A3) on large-scale instances (100≤#node≤200). # v = the number of vehicle used; ttd = total travel distance; # FE = the number of fitness evaluations. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$. The results were obtained from 30 runs.

Instances (large)	A2(4,100,2) VS A1(2,100,2)			A3(4,100,6) VS A2(4,100,2)		
	Avg # v A2-A1	Avg ttd (A2-A1)/A1	# FE A2/A1	Avg # v A3-A2	Avg ttd (A3-A2)/A2	# FE A3/A2
B-Y9	-0.07	-0.0201	1.3133	-0.10	-0.0453	2.0357
B-Y10	0	-0.0433	1.3365	-0.07	-0.0758	2.0612
B-Y11	-0.03	-0.0184	1.2888	-0.07	-0.0545	2.3372
B-Y12	-0.03	-0.0372	1.3256	0	-0.0677	2.0149
B-Y13	-0.03	-0.0308	1.3586	-0.07	-0.0524	2.0023
B-Y14	-0.03	-0.0247	1.2328	-0.10	-0.0862	2.0116
B-Y15	-0.07	-0.0229	1.2819	-0.03	-0.0506	1.9684
B-Y16	0	-0.0293	1.2170	0	-0.0730	1.9092
B-Y17	-0.03	-0.0221	1.2419	-0.13	-0.0448	1.8280
B-Y18	-0.03	-0.0423	1.2758	-0.03	-0.0749	1.8377
B-Y19	-0.03	-0.0182	1.2117	-0.30	-0.0507	2.0209
B-Y20	0	-0.0356	1.2526	-0.17	-0.0554	1.7555
Sha14	-0.03	-0.0372	1.4081	-0.07	-0.0538	2.3277
Sha15	-0.03	-0.0235	1.2131	-0.17	-0.0809	2.2867
SD12	0	-0.0422	1.4030	-0.07	-0.0691	2.1231
SD13	0	-0.0359	1.3005	-0.13	-0.0944	2.2130
Avg	-0.03	-0.0335	1.2610	-0.09	-0.0694	1.9731
Total	-0.43	-0.0335	1.2610	-1.5	-0.0694	1.9731

Table 5.9: Comparative analysis for our algorithm with different parameter configurations (A3, A4, and A5) on small-scale instances (# nodes < 100).

Instances (small)	A4(6,100,6) VS A3(4,100,6)			A5(6,200,6) VS A4(6,100,6)			Instances (small)	A4(6,100,6) VS A3(4,100,6)			A5(6,200,6) VS A4(6,100,6)		
	Avg # v A4-A3	Avg ttd (A4-A3)/A3	# FE A4/A3	Avg # v A5-A4	Avg ttd (A5-A4)/A4	# FE A5/A4		Avg # v A4-A3	Avg ttd (A4-A3)/A3	# FE A4/A3	Avg # v A5-A4	Avg ttd (A5-A4)/A4	# FE A5/A4
B-Y1	0	-0.0139	1.2368	0	0	1	Sha9	0	-0.0062	1.2932	0	0	1
B-Y2	0	-0.0139	1.2250	0	0	1	Sha10	0	-0.0079	1.3183	0	0	1
B-Y3	0	-0.0166	1.2556	0	0	1	Sha11	0	-0.0065	1.2448	0	0	1
B-Y4	-0.03	-0.0177	1.2272	0	0	1	Sha12	0	-0.0084	1.3037	0	0	1
B-Y5	-0.03	-0.0105	1.2109	0	0	1	Sha13	0	-0.0064	1.2097	0	0	1
B-Y6	0	-0.0123	1.1954	0	0	1	SD1	0	-0.0100	1.3558	0	0	1
B-Y7	-0.10	-0.0058	1.2170	0	0	1	SD2	0	-0.0244	1.3378	0	0	1
B-Y8	0	-0.0166	1.2167	0	0	1	SD3	0	-0.0076	1.2796	0	0	1
Sha1	0	0.0000	1.4162	0	0	1	SD4	0	-0.0139	1.2742	0	0	1
Sha2	0	-0.0070	1.4305	0	0	1	SD5	0	-0.0099	1.2715	0	0	1
Sha3	0	-0.0053	1.3836	0	0	1	SD6	0	-0.0226	1.2799	0	0	1
Sha4	0	-0.0102	1.3695	0	0	1	SD7	0	-0.0269	1.2479	0	0	1
Sha5	0	-0.0037	1.3752	0	0	1	SD8	0	-0.0093	1.2198	0	0	1
Sha6	0	-0.0033	1.3543	0	0	1	SD9	0	-0.0196	1.2452	0	0	1
Sha7	0	-0.0017	1.3589	0	0	1	SD10	-0.07	-0.0216	1.2778	0	0	1
Sha8	0	-0.0046	1.3317	0	0	1	SD11	0	-0.0266	1.3005	0	0	1

Overall	A4(6,100,6) VS A3(4,100,6)			A5(6,200,6) VS A4(6,100,6)		
	Avg # v	Avg ttd	# FE	Avg # v	Avg ttd	# FE
Avg	-0.01	-0.0161	1.2406	0	0	1
Total	-0.23	-0.0161	1.2406	0	0	1

Note: # v = the number of vehicle used; ttd = total travel distance; # FE = the number of fitness evaluations. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$. The results were obtained from 30 runs.

Table 5.10: Comparative analysis for our algorithm with different parameter configurations (A3, A4, and A5) on large-scale instances (100≤#node≤200). # v = the number of vehicle used; ttd = total travel distance; # FE = the number of fitness evaluations. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$. The results were obtained from 30 runs.

Instances (large)	A4(6,100,6) VS A3(4,100,6)			A5(6,200,6) VS A4(6,100,6)		
	Avg # v A4-A3	Avg ttd (A4-A3)/A3	# FE A4/A3	Avg # v A5-A4	Avg ttd (A5-A4)/A4	# FE A5/A4
B-Y9	0	-0.0119	1.1999	0	-0.0001	1.0018
B-Y10	-0.07	-0.0127	1.2232	0.03	-0.0029	0.9942
B-Y11	-0.03	-0.0106	1.1886	0	-0.0002	1.0060
B-Y12	0	-0.0145	1.1774	0	0.0000	0.9995
B-Y13	-0.10	-0.0114	1.2021	-0.03	-0.0008	1.0146
B-Y14	0	-0.0140	1.1560	0	-0.0023	1.0211
B-Y15	-0.07	-0.0134	1.2253	-0.03	-0.0017	1.0357
B-Y16	-0.03	-0.0118	1.1810	0	-0.0048	1.0291
B-Y17	0	-0.0113	1.1763	0	-0.0027	1.0501
B-Y18	0	-0.0123	1.1616	0	-0.0020	1.0222
B-Y19	-0.03	-0.0135	1.2175	-0.07	-0.0024	1.0461
B-Y20	0	-0.0178	1.1829	-0.03	-0.0053	1.0632
Sha14	0	-0.0107	1.2015	0	0	1.0017
Sha15	0	-0.0169	1.1802	0	-0.0046	1.0421
SD12	-0.07	-0.0138	1.2265	0	0.0007	1.0064
SD13	0	-0.0231	1.1786	0	-0.0009	1.0085
Avg	0	-0.0153	1.1882	-0.01	-0.0011	1.0000
Total	-0.07	-0.0153	1.1646	-0.13	-0.0011	1.0203

Sha8, α and β are largely similar, indicating that both loading methods are essentially equivalent in terms of space utilization.

The observations reveal that no single loading method consistently outperforms the other across various problem instances. Accordingly, our proposed method dynamically selects between joint or individual node loading, optimizing space usage and reducing the vehicle count.

Interactive Strategy Comparison

To further validate the effectiveness of our proposed strategy, this study conducts two sets of experiments. In these experiments, the P1R2 and R1P2 strategies replace our proposed strategy within the algorithm (specifically, in Algorithm 17, we substitute Algorithm 16 with P1R2 and R1P2, respectively). The outcomes of these modifications are then compared to evaluate their impact.

Our Strategy VS P1R2. Tables 5.11 and 5.12 present the comparative results between our proposed strategy and the P1R2 method. *Small Instances*: Over 30 runs, our method outperformed P1R2 on 26 out of 32 small-scale tests, excelling in both vehicle count and total travel distance (ttd) for nine instances. On 24 instances, our method used significantly fewer vehicles, even reducing the count by more than two on ten instances. In total, we achieved a total reduction of 61.2 vehicles, averaging 1.9 fewer per instance. On the ttd metric, we outperformed P1R2 on 11 instances. When considering only the best runs, our method surpassed P1R2 on 23 instances and reduced the total vehicle count by 65, averaging two fewer per case. *Larger Instances*: For the 16 larger-scale tests, our method consistently required fewer vehicles than P1R2. On average, we saved over three vehicles on 14 cases and more than five on nine instances, peaking at a reduction of 11.27 vehicles. This resulted in a total savings of 90 vehicles, or an average of 5.6 per instance. Considering the best runs, we surpassed P1R2 on all 16 cases, reducing vehicle usage by up to 12 and achieving a total saving of

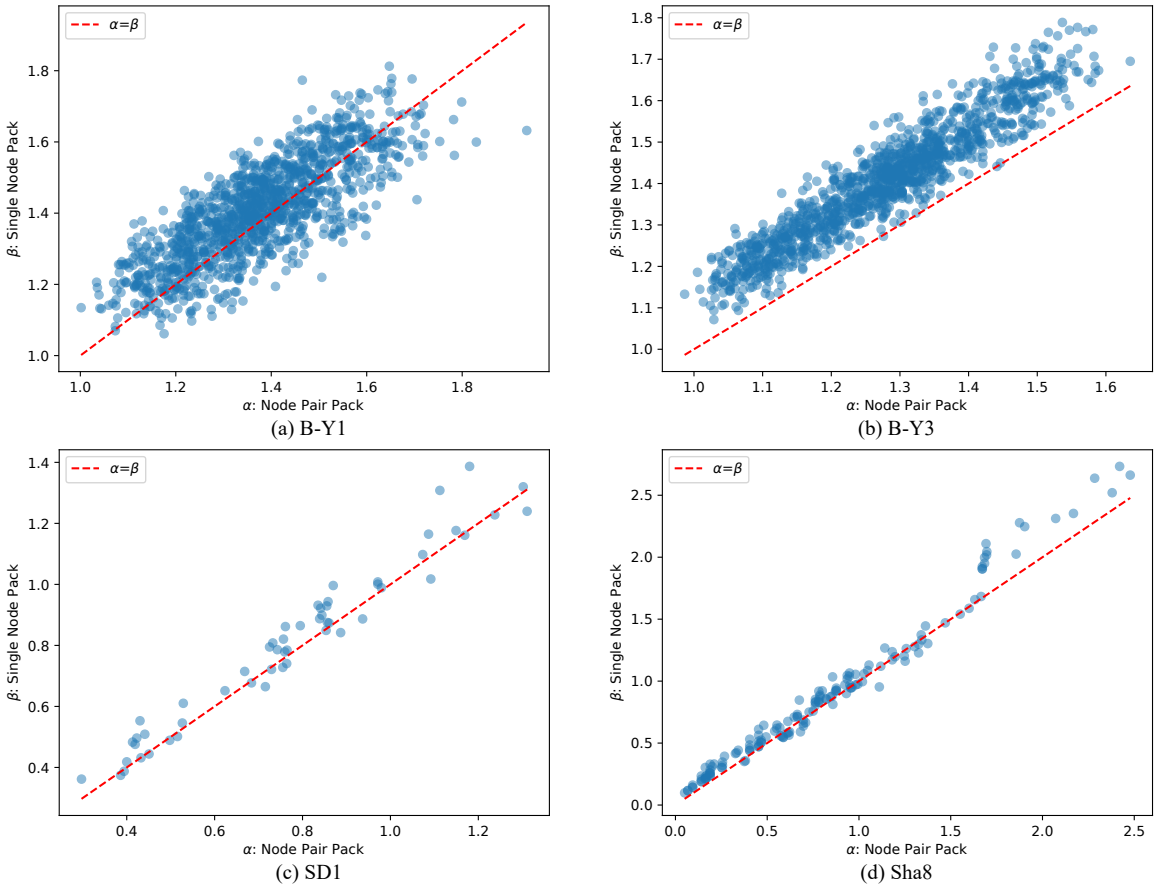


Figure 5.1: Comparison of the space occupied by loading any two nodes together (i.e., α) versus loading them separately (i.e., β) across some problem instances. Both α and β represent the sum of the spaces occupied by the two nodes. The red dashed line represents $\alpha = \beta$. Points above this line suggest that joint loading is more effective, while points below it indicate that separate loading is preferable.

96 vehicles, averaging six fewer per instance. Our best *ttd* outcomes also outperformed P1R2 on five cases. The experiments show that our method outperforms P1R2 in reducing vehicle usage, especially on larger instances. However, it underperforms P1R2 regarding *ttd*.

Our Strategy VS R1P2. Tables 5.13 and 5.14 present the comparative results between our proposed strategy and R1P2. *Small Instances*: Over 30 runs, our method outperformed R1P2 on 24 of 32 small-scale tests, excelling in both vehicle count and total travel distance (*ttd*) for eight cases. For vehicles, we significantly surpassed R1P2 on 11 instances. On *ttd*, we led on 21 cases. In the best runs, we outperformed R1P2 in vehicle count on 10 tests and tied on 16 more. Our strongest *ttd* performances beat R1P2 on 25 cases, achieving over 8% reduction on 10 problem instances. *Larger Instances*: Our approach consistently outperformed R1P2 on 11 of 16 larger-scale tests. We used fewer vehicles on eight instances, saving a total of 24.9 vehicles. In the best runs, we saved up to seven vehicles and reduced the overall count by 26, maintaining an average saving of 1.6 per case. Our top *ttd* performances outperformed R1P2 on 13 tests, with reductions exceeding 10% on six instances and peaking at 25.02%. *Computational Efficiency*: Since the R1P2 strategy requires continuously solving the 3D packing problem as the giant tour changes during the routing search process, each fitness evaluation takes more computational time compared to other strategies. Therefore, when comparing with R1P2, we use the algorithm's CPU time to measure computational resource consumption. Our method was more computationally efficient, requiring less than 1% of R1P2's CPU time on 19 of the 32 small-scale instances and below 0.1% on three. For the 16 larger tests, our method required less than 1% of the CPU time needed for R1P2 on 15 instances. These results clearly reveal that our proposed interactive routing-packing approach plays a critical role in the overall algorithm.

Table 5.11: Comparison results between our method (Ours) and P1R2 (P) on small-scale instances (# nodes < 100).

Instances (small)	Avg # v Ours-P	Avg ttd (Ours-P)/P	Best # v Ours-P	Best ttd (Ours-P)/P	# FE Ours/P	Instances (small)	Avg # v Ours-P	Avg ttd (Ours-P)/P	Best # v Ours-P	Best ttd (Ours-P)/P	# FE Ours/P
B-Y1	**-1.70**	0.1178	**-2**	0.0731	**0.1781**	Sha9	**-0.30**	**-0.0320**	**-1**	**-0.0459**	**0.2332**
B-Y2	**-1.97**	0.0348	**-2**	0.0440	**0.2685**	Sha10	**-0.87**	**-0.0210**	0	**-0.0071**	**0.3115**
B-Y3	**-2.97**	0.1575	**-3**	0.1254	**0.1405**	Sha11	**-1.07**	0.0170	**-2**	**-0.0065**	**0.2755**
B-Y4	**-3.53**	0.0731	**-3**	0.0460	**0.1917**	Sha12	**-1.27**	**-0.0609**	**-1**	**-0.0477**	**0.3558**
B-Y5	**-2.40**	0.1652	**-3**	0.1324	**0.1451**	Sha13	**-1.67**	**-0.0418**	**-2**	**-0.0622**	**0.3512**
B-Y6	**-2.07**	0.0128	**-3**	0.0308	**0.2563**	SD1	0	0.1101	0	0.0830	**0.2974**
B-Y7	**-3.80**	0.1801	**-4**	0.1654	**0.1014**	SD2	0	0.0442	0	0.0041	**0.2866**
B-Y8	**-4.07**	0.0195	**-5**	**-0.0370**	**0.1795**	SD3	0	0.0323	0	**-0.0229**	**0.4114**
Sha1	0	0.0024	0	0	**0.3795**	SD4	**-1**	0.0105	**-1**	0.0335	**0.3403**
Sha2	**-0.03**	0.0144	0	0.0118	**0.3474**	SD5	**-18**	**-0.3885**	**-18**	**-0.4131**	**0.7196**
Sha3	0	0.0058	0	**-0.0074**	**0.3624**	SD6	**-2**	0.1357	**-2**	0.0839	**0.0771**
Sha4	0	**-0.0822**	0	**-0.0686**	**0.3577**	SD7	**-1**	0.0307	**-1**	**-0.0031**	**0.5237**
Sha5	**-1**	**-0.1282**	**-1**	**-0.1398**	**0.3588**	SD8	**-2**	0.1432	**-2**	0.1421	**0.0879**
Sha6	**-1**	**-0.1033**	**-1**	**-0.0955**	**0.3680**	SD9	**-1.73**	0.0581	**-2**	**-0.0106**	**0.1921**
Sha7	**-1**	**-0.0515**	**-1**	**-0.0401**	**0.2086**	SD10	**-2.67**	0.0175	**-3**	**-0.0291**	**0.2299**
Sha8	**-0.10**	**-0.0378**	0	**-0.0458**	**0.3078**	SD11	**-2**	**-0.0239**	**-2**	0.0137	**0.1139**

Overall	Avg # v	Avg ttd	Best # v	Best ttd	# FE
Avg	**-1.9**	**0.0129**	**-2.0**	**-0.0029**	**0.1545**
Total	**-61.2**	**0.0012**	**-65**	**-0.0260**	**0.1545**

Note: # v = the number of vehicle used; ttd = total travel distance; # FE = the number of fitness evaluations. The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Table 5.12: Comparison results between our method (Ours) and P1R2 (P) on large-scale instances (100≤#node≤200). # v = the number of vehicle used; ttd = total travel distance; # FE = the number of fitness evaluations. The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Instances (large)	Avg # v Ours-P	Avg ttd (Ours-P)/P	Best # v Ours-P	Best ttd (Ours-P)/P	# FE Ours/P
B-Y9	**-3.07**	0.1297	**-3**	0.0910	**0.1278**
B-Y10	**-3.27**	0.0186	**-4**	0.0241	**0.2766**
B-Y11	**-5.10**	0.1820	**-6**	0.1385	**0.0806**
B-Y12	**-6.03**	**-0.0103**	**-7**	**-0.0568**	**0.1826**
B-Y13	**-4.50**	0.1360	**-4**	0.1408	**0.1263**
B-Y14	**-5.27**	**-0.0163**	**-5**	**-0.0376**	**0.1968**
B-Y15	**-7.63**	0.1691	**-8**	0.1336	**0.1060**
B-Y16	**-8.17**	0.0059	**-8**	**-0.0251**	**0.2066**
B-Y17	**-5.87**	0.1471	**-7**	0.1406	**0.1570**
B-Y18	**-6.47**	**-0.0266**	**-7**	**-0.0129**	**0.3508**
B-Y19	**-10.27**	0.1892	**-10**	0.1610	**0.1106**
B-Y20	**-11.27**	**-0.0082**	**-12**	0.0033	**0.2290**
Sha14	**-2.40**	0.0201	**-3**	0.0278	**0.4041**
Sha15	**-4.90**	**-0.0051**	**-5**	**-0.0054**	**0.4732**
SD12	**-1.07**	0.0254	**-2**	0.0444	**0.4811**
SD13	**-4.73**	0.0106	**-5**	0.0029	**0.1828**
Avg	**-5.6**	0.0604	**-6.0**	0.0481	**0.1781**
Total	**-90.0**	0.0501	**-96**	0.0486	**0.1781**

Table 5.13: Comparison results between our method (Ours) and R1P2 (R) on small-scale instances (# nodes < 100).

Instances (small)	Avg # v Ours-R	Avg ttd (Ours-R)/R	Best # v Ours-R	Best ttd (Ours-R)/R	T Ours/R	Instances (small)	Avg # v Ours-R	Avg ttd (Ours-R)/R	Best # v Ours-R	Best ttd (Ours-R)/R	T Ours/R
B-Y1	0.10	0.0276	0	**-0.0043**	**0.0066**	Sha9	0.97	0.0239	0	0.0188	**0.0049**
B-Y2	0.03	**-0.1109**	0	**-0.1345**	**0.0074**	Sha10	0.17	**-0.0155**	1	0.0082	**0.0070**
B-Y3	**-1.97**	**-0.0242**	**-2**	**-0.0314**	**0.0043**	Sha11	0.03	**-0.0406**	0	**-0.0411**	**0.0029**
B-Y4	**-1.47**	**-0.0962**	**-1**	**-0.1146**	**0.0074**	Sha12	0.40	**-0.0783**	1	**-0.0477**	**0.0065**
B-Y5	0.10	0.0503	0	0.0577	**0.0056**	Sha13	1.33	**-0.0091**	1	**-0.0092**	**0.0030**
B-Y6	**-0.03**	**-0.1467**	**-1**	**-0.1458**	**0.0062**	SD1	0	**-0.0099**	0	**-0.0165**	**0.0211**
B-Y7	**-3.50**	**-0.0384**	**-3**	**-0.0167**	**0.0039**	SD2	**-1**	**-0.0613**	**-1**	**-0.0817**	**0.0175**
B-Y8	**-1.23**	**-0.2448**	**-2**	**-0.2068**	**0.0065**	SD3	0	**-0.0280**	0	**-0.0795**	**0.0296**
Sha1	0	**-0.0496**	0	**-0.0337**	**0.0587**	SD4	0	**-0.0800**	0	**-0.0831**	**0.0166**
Sha2	**-1.03**	**-0.1087**	**-1**	**-0.1099**	**0.0452**	SD5	16	1.4204	16	1.5386	**0.0001**
Sha3	0	**-0.0217**	0	**-0.0031**	**0.0861**	SD6	**-1**	0.0857	**-1**	0.0407	**0.0003**
Sha4	0	**-0.0138**	0	**-0.0260**	**0.0340**	SD7	**-0.90**	**-0.1281**	0	**-0.1609**	**0.0151**
Sha5	0	**-0.0298**	0	**-0.0275**	**0.0298**	SD8	**-1**	0.0675	**-1**	0.0606	**0.0013**
Sha6	0.27	**-0.0961**	1	**-0.0393**	**0.0238**	SD9	**-0.70**	0.0037	0	**-0.0596**	**0.0015**
Sha7	2	0.0982	2	0.1232	**0.0104**	SD10	**-1.67**	**-0.1241**	**-2**	**-0.0889**	**0.0019**
Sha8	0	**-0.0340**	0	**-0.0384**	**0.0190**	SD11	0	**-0.0751**	0	**-0.0891**	**0.0004**

Overall	Avg # v	Avg ttd	Best # v	Best ttd	T
Avg	0.2	0.0035	**0.2**	0.0050	**0.0014**
Total	5.9	0.0067	**7**	**-0.0053**	**0.0014**

Note: # v = the number of vehicle used; ttd = total travel distance. The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Table 5.14: Comparison results between our method (Ours) and R1P2 (R) on large-scale instances (100≤#node≤200). # v = the number of vehicle used; ttd = total travel distance. The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Instances (large)	Avg # v Ours-R	Avg ttd (Ours-R)/R	Best # v Ours-R	Best ttd (Ours-R)/R	T Ours/R
B-Y9	0.27	0.0026	1	**-0.0081**	**0.0039**
B-Y10	0.17	**-0.1565**	0	**-0.1319**	**0.0078**
B-Y11	**-4.27**	**-0.0774**	**-5**	**-0.0680**	**0.0034**
B-Y12	**-2.07**	**-0.2636**	**-3**	**-0.2479**	**0.0067**
B-Y13	0.37	0.0110	1	0.0145	**0.0033**
B-Y14	0.30	**-0.1721**	0	**-0.1324**	**0.0041**
B-Y15	**-6.13**	**-0.1040**	**-6**	**-0.0886**	**0.0028**
B-Y16	**-3.03**	**-0.2803**	**-3**	**-0.2615**	**0.0062**
B-Y17	0.43	0.0093	0	0.0201	**0.0029**
B-Y18	0.13	**-0.1990**	0	**-0.1848**	**0.0061**
B-Y19	**-8.43**	**-0.0978**	**-7**	**-0.0862**	**0.0038**
B-Y20	**-4.77**	**-0.2724**	**-5**	**-0.2502**	**0.0056**
Sha14	0.87	**-0.0411**	1	**-0.0175**	**0.0022**
Sha15	2.70	**-0.0082**	2	0.0282	**0.0020**
SD12	**-0.67**	**-0.0965**	**-1**	**-0.0373**	**0.0105**
SD13	**-0.73**	**-0.0815**	**-1**	**-0.0887**	**0.0012**
Avg	**-1.6**	**-0.1142**	**-1.6**	**-0.0963**	**0.0035**
Total	**-24.9**	**-0.0990**	**-26**	**-0.0742**	**0.0035**

5.4 Conclusion

In this chapter, we introduce an adaptive interactive routing-packing strategy that can be applied to various algorithms for solving the 3L-SDVRP, including those presented in Chapters 3 and 4. The key innovation of our approach lies in its ability to adaptively select the appropriate packing pattern based on various conditions, such as the vehicle's remaining space and the space requirements of different packing patterns at each node. Our strategy leverages the P1R2 idea of loading a single node independently and the 2C-SP concept of loading two nodes together. Additionally, our strategy provides flexibility and adaptability in adjusting packing decisions as the giant tour changes, resulting in improved performance. Our experimental results demonstrate the effectiveness of this approach, especially in enhancing solution quality in terms of vehicle count across most problem instances, with significant benefits observed in larger-scale cases. To explore the underlying reasons for our method's success, we further analyze problem instances and conduct comprehensive interactive strategy evaluation and parameter sensitivity studies.

While our approach efficiently finds high-quality solutions, especially in terms of the number of vehicles used, it does not outperform the state-of-the-art 3L-SDVRP algorithm, SDVRLH2 [10], in terms of total travel distance (ttd) in some cases. The reduction in the vehicle number inadvertently led to an increase in ttd. In the next chapter, we aim to leverage domain knowledge to develop approaches that reduce the ttd while maintaining a minimal number of vehicles.

Chapter 6

Knowledge-Guided Optimization for 3L-SDVRP*

In Chapter 3, we introduced the PEAC-HNF, a multi-objective evolutionary algorithm designed to solve the 3L-SDVRP. While this method effectively balances the two conflicting objectives and provides diverse solutions for decision-makers, it is less suitable for large-scale problems due to its substantial computational demands. Building on this foundation, Chapter 4 presented an enhanced local search method, derived from the state-of-the-art 3L-SDVRP algorithm SDVRLH2 [10], specifically tailored to address large-scale instances of the problem more efficiently. Given that 3L-SDVRP encompasses both routing and packing—each an NP-hard problem—the interaction between these two components is crucial for achieving high-quality solutions. Chapter 5 delves into this critical aspect, proposing an adaptive routing-packing strategy that significantly reduces the number of vehicles required. Although our proposed methods outperforms the state-of-the-art SDVRLH2 [10] in terms of vehicle reduction, it still falls short in optimizing the total travel distance (ttd).

*This chapter is partially based on a paper published at the 18th International Conference on Parallel Problem Solving From Nature (PPSN' 24) [117].

To address this limitation and improve *ttd*, this chapter introduces a novel approach by incorporating domain knowledge into the search algorithm, guiding the search process more effectively and enhancing the overall solution quality. Specifically, we propose an Adaptive Knowledge-Guided Insertion (AKI) operator, which integrates domain expertise and allows for larger step sizes in the search process. Building on this, we develop the Adaptive Knowledge-Guided Search (AKS) algorithm, which offers two primary advantages: first, it utilizes the domain knowledge embedded in the AKI operator to provide a more informed and well-directed search; second, it strikes a better balance between exploration and exploitation by employing the AKI operator for broader, large-step searches while utilizing traditional neighborhood operators for more precise, small-step refinements. This balanced approach significantly enhances the algorithm' s overall search capabilities.

This chapter is organized as follows. Section 6.1 provides an introduction to the background and outlines our motivation for this chapter. Section 6.2 delves into the extraction of heuristics from domain knowledge, introducing the proposed AKI operator and the AKS algorithm. Section 6.3 presents our computational studies, where we compare our proposed method with the state-of-the-art algorithms and analyze its effectiveness. Finally, Section 6.4 summarizes the key findings and contributions of this chapter.

6.1 Introduction

Due to the high complexity of 3L-SDVRP, exact methods have proven ineffective for solving it. Consequently, intelligent optimization algorithms, particularly metaheuristic algorithms, have become the common and effective approach for tackling such complex problems [13] [14] [50] [51] [107]. These methods are based on a generate-and-test iterative strategy [106], where each iteration involves the generation of a new set of potential solutions from the existing ones using various search

operators, with the hope of finding improved solutions. This process is iteratively repeated to ultimately find an approximate optimal solution.

Search operators are crucial in these algorithms as they determine the search direction and step size in the solution space. Many general search operators used for solving vehicle routing problems (VRPs) with 3D loading constraints are as follows. *Swap:* Exchange the positions of two nodes [10]. *Shift:* Move a node to another position [14]. *2-opt:* Breaks two edges in a node sequence and reconnects two new edges to alter the node order [10]. *3-opt:* Breaks three edges in a node sequence and reconnects three new edges [10]. *Move & Rotate Block:* Groups identical boxes into blocks for a block sequence representation, then selects and inserts blocks between sequences [13]. *Split:* Divides a node sequence into two non-empty sequences [121]. *Best Cost Route Crossover:* Selects two node sequences, inserts two consecutive nodes from each into the other [65]. *1-point Crossover:* Randomly divides two node sequences at a selected position and recombines them [14]. *2-point Crossover:* Randomly selects two points in two node sequences and exchanges the segments [84].

Additionally, search step size, defining the extent of solution change by search operator per iteration, is crucial for algorithm performance. Methods for 3L-SDVRP can be categorized into local search-based methods [10] [13] [14] [66] [107] with smaller step sizes and global search-based methods [50] [51] [65] [70] with larger ones. Although local search is efficient, it risks falling into local optima, whereas global search tends to converge more slowly. Researchers have explored balancing small and large search step sizes in optimization. [104] showed larger neighborhood sizes in simulated annealing improve effectiveness. Large neighborhood search (LNS) [86] and adaptive large neighborhood search (ALNS) [79] algorithms are effective and adaptable to different VRPs. The memetic algorithm (MA) [64], blending local and global searches, has been applied to diverse problems but needs more computational resources. However, directly applying these methods to 3L-SDVRP is ineffective as they do not incorporate domain knowledge which is crucial for addressing such a complex prob-

lem. Thus, achieving a balanced trade-off between large and small step sizes in this context remains a significant challenge.

In this chapter, we introduce a knowledge-guided approach that leverages domain-specific heuristics and balances small-large search step sizes to enhance the search for high-quality solutions to the 3L-SDVRP. It is important to clarify the distinction between the knowledge utilized in our approach and the concept of knowledge transfer commonly discussed in evolutionary computation. In classical evolutionary algorithms, "knowledge transfer" typically refers to the process where useful information, patterns, or learned models are passed from one generation to the next, either explicitly (e.g., via learned parameters) or implicitly (through the genetic makeup of the population). In contrast, our method does not involve such inter-generational transfer.

Instead, the knowledge embedded in our algorithms is derived from human understanding of the problem' s structure, constraints, and search space. These domain insights are incorporated into the algorithm in the form of carefully designed heuristics, rules, and search operators, which remain fixed during the search process. The aim is to enhance search efficiency and solution quality by directing the exploration towards more promising regions of the search space, rather than by adapting or evolving knowledge through population dynamics.

As a result, the effectiveness of our approach relies on the depth and quality of the incorporated domain knowledge, rather than on learning or adapting from the search history across generations. This static integration of knowledge is well suited for highly complex and constrained combinatorial problems like the 3L-SDVRP, where online adaptation or automated knowledge discovery may be computationally infeasible.

Motivated by the above analysis, the contributions of this chapter are as follows:

- Heuristics are extracted from domain knowledge. Specifically, based on the

"giant tour" representation, we develop a hypothesis about "what constitutes a good giant tour" through observation. We use a node insertion approach to change the order of nodes in the current giant tour to improve its quality and propose two node insertion rules.

- An Adaptive Knowledge-guided Insertion (AKI) operator is developed which can adaptively select suitable node insertion rules based on node distribution characteristics. The proposed AKI operator utilizes domain knowledge and has a large search step size.

- The AKI operator is integrated into a local search framework to form an Adaptive Knowledge-guided Search (AKS) algorithm. This algorithm combines small step search (exploitation) of traditional neighborhood operators with the AKI operator's larger step (exploration), improving search capabilities.

6.2 Knowledge-Guided Optimization Algorithm for 3L-SDVRP

This section starts with heuristics extracted from domain knowledge, discusses the features of an effective giant tour, and introduces two node insertion rules. It then details the proposed AKI operator and AKS algorithm. Our approach incorporates domain knowledge and balances search step size to improve solution quality.

6.2.1 Extracting Heuristics from Domain Knowledge

This study employs the "giant tour" representation, which encodes the solution as a sequence of all nodes (for a detailed description, refer to Section 3.2.1). A key question is what defines an effective giant tour. Our aim is to identify high-quality

giant tour characteristics, leveraging domain knowledge for targeted heuristic search in the algorithm to enhance solution quality.

Our study hypothesizes that in an effective giant tour, adjacent or nearby nodes in the sequence should be physically close on the actual map, as illustrated in Fig. 6.1. For example, routes derived from giant tour 1 are less efficient intuitively due to vehicles traveling between distant nodes, bypassing nearer ones, whereas routes from giant tour 2 avoid such inefficiencies, indicating higher quality solutions. This underscores the importance of logical node sequencing in route planning to improve solution quality.

Transitioning from a general to an optimized giant tour, like giant tour 1 to giant tour 2 in Fig. 6.1, is inefficient with traditional operators due to their small step sizes and lack of domain knowledge. For instance, evolving from route 2 to route 2′ needs multiple steps even under ideal conditions (e.g., four/five consecutive steps by Swap/Shift operator). This highlights the need for search operators that have large step sizes for efficiency and leverage domain knowledge for strategic direction in giant tour optimization.

To align with our hypothesis, we employ a node insertion strategy, inserting a node between each pair of consecutive nodes i and j in a giant tour to optimize node sequence. We propose two insertion rules, as shown in Fig. 6.2: the Proximity rule, ensuring inserted nodes are near node i, and the Connectivity rule, making the inserted node a 'bridge' for a better route. These rules aim to reorganize the giant tour into a sequence that better matches our hypothesis for an optimal layout.

Experiments evaluating two node insertion rules, detailed in Section 6.3.3, demonstrated their effectiveness. However, significant performance differences were observed between the two rules. This raised questions about what affect rule performance and the possibility of developing a flexible search operator. Analysis in Section 6.3.3 highlighted the importance of node distribution, leading to the development of an

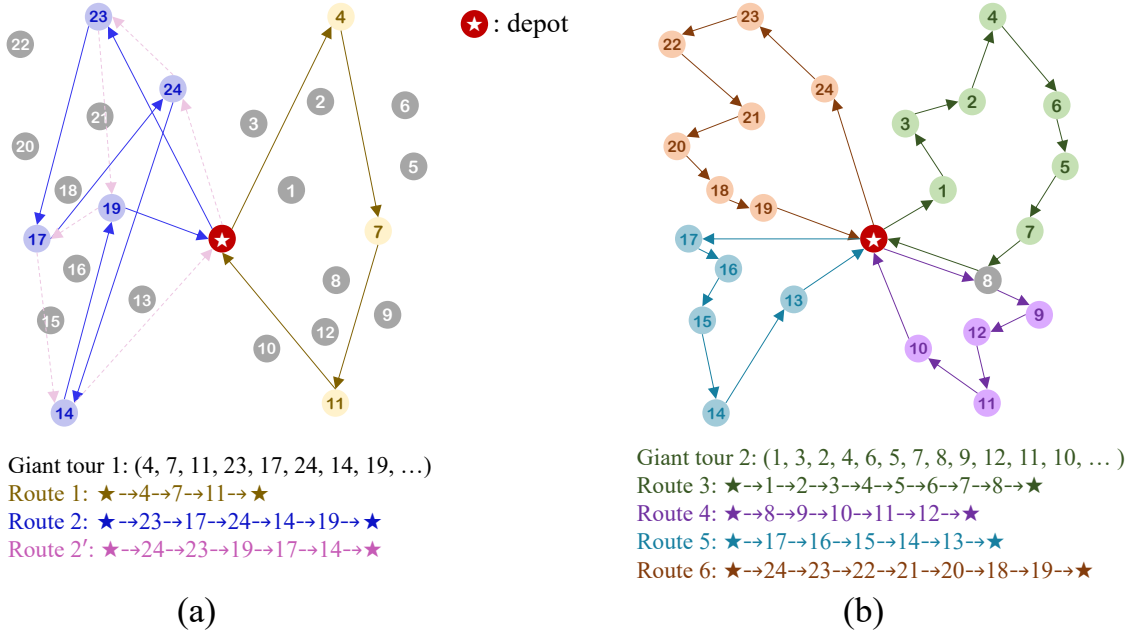


Figure 6.1: Illustration of giant tours and routes. In (a), routes 1-2 are obtained from giant tour 1. In (b), routes 3-6 are obtained from giant tour 2.

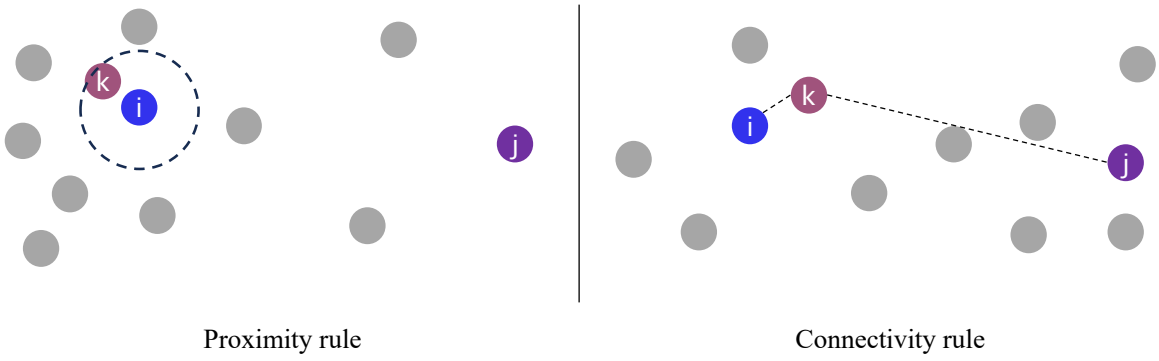


Figure 6.2: Two node insertion rules. Proximity rule: The inserted node k is as close as possible to node i. Connectivity rule: The inserted node k acts as a bridge between nodes i and j.

adaptive knowledge-guided insertion (AKI) operator based on these findings.

6.2.2 Adaptive Knowledge-guided Insertion (AKI) Operator

Algorithm 18 illustrates our proposed AKI operator. It traverses giant tour g, inserting nodes between consecutive nodes i and j based on node densities ρ_i and ρ_j. If the ratio of $max(\rho_i, \rho_j)$ to $min(\rho_i, \rho_j)$ exceeds threshold θ_{dens} (Step 7), the Proximity rule (Steps 8-9) or Connectivity rule (Steps 10-11) is applied, depending on density. If densities are similar (Step 13), rules are chosen based on nodes' positions relative to the depot o and node density ρ_{ij} between i and j (Steps 15-16 or 19-23). The g updates after each node insertion.

To demonstrate its larger step size, the AKI operator was empirically tested on 100 random giant tours. We applied AKI once per giant tour and compared its effect to traditional Swap (exchanges the positions of two nodes in the giant tour) and Shift (moves a node to another position) operators, calculating the average operations needed for similar results. Specifically, given a giant tour denoted as a, the AKI operator produces a new tour b. We then apply Swap and Shift operators multiple times to transform a into b, calculating the minimum number of operations required for both Swap and Shift operators. The calculation methods are detailed in Algorithms 19 and 20. Results in Table 6.1 show Swap needed 30.2 operations and Shift 15.84 for 50-node instances, increasing to 125.32 and 70.82 for 200 nodes. This demonstrates AKI's significantly larger step size, especially for larger problems.

Calculating Node Density in AKI Operator

Our proposed AKI operator adaptively selects the most suitable node insertion rule by analyzing node distribution characteristics. We quantify these characteristics using 'node distribution density'. Algorithm 21 details how we calculate this density around

Algorithm 18 Adaptive Knowledge-guided Insertion (AKI) Operator

Input: g: a giant tour; θ_{dens}: density ratio threshold

Output: g: the giant tour after node insertion

$n \leftarrow$ the number of node in g; $\kappa \leftarrow 0$ //κ: Current node index
while $\kappa < n - 2$ **do**
  $i \leftarrow g[\kappa]; j \leftarrow g[\kappa + 1]$ // Inserting node between i and j
  $d_{ij} \leftarrow$ distance between i and j;
  $\rho_i \leftarrow$ get node density around i by Algorithm 21 (with parameters $\vartheta = \kappa, \vartheta' = \kappa + 1$)
  $\rho_j \leftarrow$ get node density around j by Algorithm 21 (with parameters $\vartheta = \kappa + 1, \vartheta' = \kappa + 2$)
  if $max(\rho_i, \rho_j)/min(\rho_i, \rho_j) > \theta_{dens}$ **then** // A large gap in node density around i and j.
    if $\rho_i > \rho_j$ **then** // Node density around i is larger.
      $g \leftarrow$ use *Proximity rule* to insert a node between i and j, then update g
    else // Node density around j is larger.
      $g \leftarrow$ use *Connectivity rule* to insert a node between i and j, then update g
    end if
  else // Node density around i and j is similar.
    $d_{io}, d_{jo} \leftarrow$ distance between i (or j) and depot o
    if $d_{ij} > d_{io}$ and $d_{ij} > d_{jo}$ **then** // i and j are roughly on opposite sides of the depot.
      $g \leftarrow$ use *Proximity rule* to insert a node between i and j, then update g
    else // i and j are on the same side of the depot.
      $\rho_{ij} \leftarrow$ get average node density between i and j by Algorithm 22 (with $\eta = \kappa$)
      if $\rho_{ij}/min(\rho_i, \rho_j) > \theta_{dens}$ **then** // Node density between i and j are larger.
        $g \leftarrow$ use *Connectivity rule* to insert a node between i and j, and update g
      else // Node density between i and j are not significantly larger.
        $g \leftarrow$ use *Proximity rule* to insert a node between i and j, then update g
      end if
    end if
  end if
  $\kappa \leftarrow \kappa + 1$
end while
return g

Table 6.1: Average number of continuous steps required by Swap and Shift operators to achieve equivalent change as one step by the AKI operator in giant tours. n is #nodes of giant tour.

Operator	n=31	n=50	n=75	n=100	n=200
Swap	18.38	30.2	46.4	61.82	125.32
Shift	9.06	15.84	24.92	34.04	70.82

Algorithm 19 Calculate the Minimum Number of Swaps

Input: a: the original giant tour; b: the giant tour generated from a after applying the AKI operator

Output: *swaps*: the minimum number of swaps to convert a into b

1: **if** len(a) is not equal to len(b) or set of a is not equal to set of b **then**
2: **raise** ValueError("Giant tours 'a' and 'b' must have the same elements and length.")
3: **end if**
4: $index\_map \leftarrow$ {value: index **for** index, value **in** enumerate(b)}
5: $swaps \leftarrow 0$
6: **for** $i \leftarrow 0$ **to** length of a **do**
7: $correct\_value \leftarrow b[i]$
8: **if** $a[i] \neq correct\_value$ **then**
9: $swap\_index \leftarrow index\_map[a[i]]$
10: Swap $a[i]$ **and** $a[swap\_index]$
11: Update $index\_map$ after the swap
12: $index\_map[a[swap\_index]] \leftarrow swap\_index$
13: $swaps \leftarrow swaps + 1$
14: **end if**
15: **end for**
16: **return** *swaps*

Algorithm 20 Calculate the Minimum Number of Insertions

Input: a: the original giant tour; b: the giant tour generated from a after applying the AKI operator

Output: $inversions$: the minimum number of insertions to convert a into b

1: **if** len(a) is not equal to len(b) **or** set of a is not equal to set of b **then**
2: **raise** ValueError("Giant tours 'a' and 'b' must have the same elements and length.")
3: **end if**
// Create a map $index\_map$ to store the index of each element in list b
4: **for** each $index, value$ in enumerate(b) **do**
5: $index\_map[value] \leftarrow index$
6: **end for**
// Convert a to a list of indices $indexed\_a$ according to the order of elements in b
7: **for** each $value$ in a **do**
8: Append $index\_map[value]$ to $indexed\_a$
9: **end for**
10: $lis\_length \leftarrow$ calculate the length of the longest increasing subsequence in $indexed\_a$
11: $insertions \leftarrow$ len(a) – lis_length
12: **return** $insertions$

a given node i. ϑ represents the index of node i in the giant tour g (Step 2), and D_i is used to store the distances between node i and other nodes (node $g[\vartheta']$ and subsequent nodes in the giant tour) (Steps 3-5). D_i is sorted in ascending order, and the average of the first $\lfloor \frac{n-\vartheta-2}{2} \rfloor$ distances is computed to represent the node distribution density around node i.

Algorithm 21 Calculate node density around node i

Input: g: a giant tour; ϑ, ϑ': node indexes in g

Output: ρ: the node density around $g[\vartheta]$

1: $n \leftarrow$ the number of node in g
2: $i \leftarrow g[\vartheta]; D_i \leftarrow \emptyset$ `// g indexing starts from 0`
3: **for** $k \leftarrow g[\vartheta']$ to $g[n-1]$ **do**
4: $\quad d_{ki} \leftarrow$ distance between nodes k and i; $D_i \leftarrow D_i \cup \{d_{ki}\}$
5: **end for**
6: $D_i \leftarrow$ sort D_i in increasing order
7: $\rho \leftarrow$ get the average of the first $\lfloor \frac{n-\vartheta-2}{2} \rfloor$ elements in D_i
8: **return** ρ

Algorithm 22 demonstrates the process for calculating the node distribution density between two nodes i and j, which are adjacent in a giant tour. During the calculation, for each node k in the giant tour g that is positioned after node j (Step 4), the algorithm determines the relative position of node k in relation to nodes i and j based on the distances between each pair of nodes i, j, and k. If $d_{ik} < d_{ij}$ and $d_{jk} < d_{ij}$ (Step 6), it is considered that node k is approximately between i and j. The node distribution density ρ_k around node k is then calculated using Algorithm 21 and stored in P (Steps 7-8). Finally, the average value of all elements in P is used to represent the node distribution density between nodes i and j (Steps 11-13).

Algorithm 22 Calculate node density between nodes i and j

Input: g: a giant tour; η: index of node i in g

Output: ρ: the node density between i and j

1: $n \leftarrow$ the number of node in g; $P \leftarrow \emptyset$; $\rho \leftarrow 0$
2: $i \leftarrow g[\eta]$; $j \leftarrow g[\eta+1]$ `// g indexing starts from 0`
3: **for** $\eta' \leftarrow \eta+2$ to $n-1$ **do**
4: $\quad k \leftarrow g[\eta']$
5: $\quad d_{ij}, d_{ik}, d_{jk} \leftarrow$ distances between each pair of nodes i, j, and k
6: $\quad$ **if** $d_{ik} < d_{ij}$ and $d_{jk} < d_{ij}$ **then**
7: $\quad\quad \rho_k \leftarrow$ get node density around k by Algorithm 21 `(with` $\vartheta = \eta', \vartheta' = \eta+2$`)`
8: $\quad\quad P \leftarrow P \cup \{\rho_k\}$
9: $\quad$ **end if**
10: **end for**
11: **if** $P \neq \emptyset$ **then**
12: $\quad \rho \leftarrow$ get the mean of all elements in P
13: **end if**
14: **return** ρ

6.2.3 Adaptive Knowledge-guided Search (AKS) Algorithm

The AKS algorithm (Algorithm 23) combines the AKI operator with local search framework, starting with a random giant tour. It employs both Swap and 2-opt operators to generate neighborhoods (Steps 6-12), applying AKI based on probability p for potential improvements. The best solution of each iteration, s_{iter_best}, is utilized to update both s_{best} and s_{curr}. (Steps 13-14). Iterations stop after n_{no_imp} non-improvements, followed by a restart (Steps 15-17). In the AKS algorithm, traditional operators conduct detailed searches with small step sizes, while the AKI operator enables expansive exploration through its larger step size. Furthermore, incorporating domain knowledge into the AKI operator significantly enhances the overall performance of the AKS algorithm.

Other Details of AKS Algorithm

Our packing method and routing-packing interactive strategy are based on the approach proposed in Section 5, which enables adaptive packing decision adjustments during route planning process. Additionally, in real-world scenarios, the reduction of vehicle numbers is more important than decreasing total travel distance, due to the substantially greater expenses involved in purchasing, maintaining more vehicles, and employing additional drivers compared to the costs associated with longer travel distances. Thus, following the SDVRLH2 method [10], we prioritized vehicle number as the primary objective, with ttd as the secondary objective. To clarify, when comparing two solutions s_1 and s_2 within the AKS algorithm, s_1 is considered superior to s_2 if it uses fewer vehicles, or if both have the same number of vehicles but s_1 has a shorter total travel distance (ttd) than s_2.

Algorithm 23 AKS Algorithm (Key innovation in red boxes.)

Input: problem instance data

Output: best solution s_{best}

$P \leftarrow$ get packing solution for each node
$s_{init} \leftarrow$ generate initial giant tour; $s_{best} \leftarrow s_{init}$
for $t \leftarrow 1$ to n_{out} **do**
    $s_{curr} \leftarrow s_{best}$
    for $iter \leftarrow 1$ to n_{iter} **do**
        $N_{swap} \leftarrow$ get neighborhood of s_{curr} by $Swap$
        $N_{swap} \leftarrow$ apply the AKI operator (Algorithm 18) with probability p to N_{swap}
        $s_{iter\_best} \leftarrow$ decode individuals and pick best individual in N_{swap}
        $N_{2opt} \leftarrow$ get neighborhood of s_{iter_best} by 2-opt
        $N_{2opt} \leftarrow$ apply the AKI operator (Algorithm 18) with probability p to N_{2opt}
        $N \leftarrow N_{2opt} \cup \{s_{iter\_best}\}$
        $s_{iter\_best} \leftarrow$ decode individuals and pick best individual in N
        $s_{best} \leftarrow$ update s_{best} by s_{iter_best} where necessary
        $s_{curr} \leftarrow s_{iter\_best}$
        if s_{curr} has not been improved for n_{no_imp} consecutive iterations **then**
            break
        end if
    end for
end for
return s_{best}

6.3 Computational Studies

This section assesses the performance of our AKS algorithm against the state-of-the-art method for 3L-SDVRP. We also evaluate two node insertion rules we proposed, observing notable performance differences and pinpointing node distribution as a key factor. Additional experiments on newly created problem instances with varied node distributions validate the effectiveness of our AKI operator. For clarity, the chapter's main text includes only comparative data between methods. Detailed experimental results are available online in [114].

6.3.1 Experimental Setting

In this chapter, experiments were conducted using three widely used datasets for 3L-SDVRP: the B-Y instances [10], Shanghai instances [10], and SD instances [13]. A total of 48 problem instances were tested, with 32 small-scale instances involving fewer than 100 nodes and 16 larger-scale instances containing between 100 and 200 nodes.

In our experiments, we adjusted θ_{dens} in Algorithm 18 according to the node count: 1.2 for under 75 nodes, and 1.5 for 75 or more. Hyperparameters in Algorithm 23 were set as $n_{out} = 4$, $n_{iter} = 100$, $n_{no\_imp} = 2$, and $p = 0.1$. It is worth noting that we did not fine-tune the parameters to demonstrate that the strong performance of our algorithm arise from its innovative design rather than from careful parameter tuning. Experiments were run 30 times on a server with 4x Intel Xeon Platinum 9242 CPUs, 256G RAM, Python 3.7, and Ubuntu 20.04.

6.3.2 Comparing to State-of-the-Art

In this study, we evaluated our AKS algorithm against the state-of-the-art SDVRLH2 method [10] for 3L-SDVRP. The detailed comparative results are included in Tables 6.2 and 6.3.

Small-Scale Problem Instances (# node<100)

Overall Average Results: As shown in Tab. 6.2, in the majority of cases (25 out of 32), the AKS algorithm demonstrated superior performance compared to SDVRLH2 [10]. It achieved significantly better results in both vehicle count and *ttd* on 19 instances, while on four instances, there was no significant difference.

Average Vehicle Count: The AKS algorithm was significantly better on 21 instances, reducing two or more vehicles on seven instances, and up to 12.1 vehicles. No significant difference in vehicle count was observed on ten instances. In total, AKS reduced 52.23 vehicles across all instances, averaging a reduction of 1.63 vehicles per instance.

Average ttd: AKS was significantly better on 23 instances, with no significant difference on six instances. *ttd* reduction exceeded 5% on 12 instances and 10% on seven instances, with the highest reduction being 40.34%, averaging a decrease of 12.78% per instance.

Best Vehicle Count: In terms of the best results from multiple runs, our method achieved better vehicle count on 21 instances, with six instances showing a reduction of three or more vehicles compared to SDVRLH2's best. Overall, our method reduced 53 vehicles in total, averaging 1.66 per instance.

Best ttd: Our method's best solutions were significantly better on 22 instances, with reductions exceeding 5% on 16 instances and 10% on eight instances. The highest reduction was 41.31%, averaging 13.23% per instance.

Large-Scale Problem Instances (100 ≤ #node ≤ 200)

Overall Average Results: As shown in Tab. 6.3, the AKS algorithm was significantly better on 15 out of 16 larger-scale instances, with both vehicle count and *ttd* being significantly better than SDVRLH2 on 14 instances.

Average Vehicle Count: Our method had significantly fewer vehicles on 15 instances. Compared to SDVRLH2 [10], the reduction in vehicles exceeded three on seven instances and five on five instances, with a maximum reduction of about ten vehicles. Overall, our method reduced a total of 58.53 vehicles, averaging a reduction of 3.66 vehicles per instance.

Average ttd: AKS was significantly better on 14 instances, with no significant difference on two instances. *ttd* reduction exceeded 5% on eight instances and 10% on three instances, with the highest reduction being 12.53%, averaging a decrease of 7.74% per instance.

Best Vehicle Count: Our method achieved better results in vehicle count on all 16 instances, with a reduction of five or more vehicles on five instances, reaching up to 11 vehicles. Overall, our method reduced 61 vehicles in total, averaging 3.81 per instance.

Best ttd: AKS's best results were significantly better than SDVRLH2, with reductions exceeding 5% on 11 instances, 10% on three instances, and a maximum reduction of 19.5%, averaging 8.07% per instance.

These findings demonstrate the AKS algorithm's superior performance over SDVRLH2 in reducing both the number of vehicles required and *ttd*, across both small-scale and larger-scale problem instances.

Table 6.2: Comparison results of our AKS algorithm and SDVRLH2 (SD) on small-scale instances (# node < 100).

Instances (small)	Avg # v AKS-SD	Avg ttd (AKS-SD)/SD	Best # v AKS-SD	Best ttd (AKS-SD)/SD	# FE AKS/SD	Instances (small)	Avg # v AKS-SD	Avg ttd (AKS-SD)/SD	Best # v AKS-SD	Best ttd (AKS-SD)/SD	# FE AKS/SD
B-Y1	**-1**	**-0.0263**	**-1**	**-0.0448**	0.0533	Sha9	**-0.6**	**-0.0337**	**-1**	**-0.0351**	0.0644
B-Y2	**-1.5**	**-0.0220**	**-1**	**-0.0686**	0.0666	Sha10	0	0.0083	0	**-0.0046**	0.0738
B-Y3	**-1**	**-0.0309**	**-1**	**-0.0521**	0.0479	Sha11	**-0.2**	**-0.0082**	0	**-0.0270**	0.0461
B-Y4	**-0.1**	**-0.0233**	**-1**	**-0.0141**	0.0575	Sha12	**-2**	**-0.0820**	**-2**	**-0.1030**	0.0557
B-Y5	**-1**	**-0.0148**	**-1**	**-0.0352**	0.0602	Sha13	**-0.7**	0.0045	0	0.0019	0.0614
B-Y6	**-1.2**	**-0.0243**	**-2**	**-0.0191**	0.0592	SD1	0	**-0.0127**	0	0.0243	0.0588
B-Y7	**-1.87**	**-0.0383**	**-2**	**-0.0567**	0.0873	SD2	**-1**	**-0.0968**	**-1**	**-0.0819**	0.0546
B-Y8	**-1**	**-0.0352**	**-1**	**-0.0306**	0.0595	SD3	0	**-0.0258**	0	**-0.0167**	0.0616
Sha1	0	0	0	0	0.1145	SD4	**-1**	**-0.0529**	**-1**	**-0.0581**	0.0523
Sha2	0	0.0213	0	0.0136	0.0948	SD5	1	0.0267	1	0.0176	0.0510
Sha3	0	**-0.0223**	0	**-0.0303**	0.0412	SD6	**-3**	**-0.1275**	**-3**	**-0.1134**	0.0715
Sha4	0	0.0191	0	0.0011	0.0445	SD7	**-3**	**-0.1294**	**-3**	**-0.1463**	0.0701
Sha5	**-1**	**-0.1657**	**-1**	**-0.1706**	0.0725	SD8	**-4**	**-0.1305**	**-4**	**-0.1424**	0.0644
Sha6	**-1**	**-0.0601**	**-1**	**-0.0990**	0.0401	SD9	**-3.97**	**-0.1105**	**-4**	**-0.1270**	0.0643
Sha7	**-1**	**-0.0607**	**-1**	**-0.0674**	0.0486	SD10	**-10**	**-0.3754**	**-10**	**-0.3862**	0.0694
Sha8	0	**-0.0373**	0	**-0.0424**	0.0899	SD11	**-12.1**	**-0.4034**	**-12**	**-0.4131**	0.0426

Overall	Avg # v	Avg ttd	Best # v	Best ttd	# FE
Avg	**-1.63**	**-0.1278**	**-1.66**	**-0.1323**	0.0591
Total	**-52.23**	**-0.1278**	**-53.00**	**-0.1323**	0.0591

Note: # v = the number of vehicles used; ttd = total travel distance; # FE = the number of fitness evaluations. The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Table 6.3: Comparison results of our AKS algorithm and SDVRLH2 (SD) on large-scale instances (100≤#node≤200)). # v = the number of vehicles used; ttd = total travel distance; # FE = the number of fitness evaluations. The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Instances (large)	Avg # v AKS-SD	Avg ttd (AKS-SD)/SD	Best # v AKS-SD	Best ttd (AKS-SD)/SD	# FE AKS/SD
B-Y9	**-1**	**-0.0128**	**-1**	**-0.0294**	0.0745
B-Y10	**-2.8**	**-0.0389**	**-3**	**-0.0355**	0.0686
B-Y11	**-2.3**	**-0.0555**	**-3**	**-0.0598**	0.0714
B-Y12	**-0.5**	**-0.0039**	**-1**	**-0.0151**	0.0739
B-Y13	**-2.4**	**-0.0517**	**-3**	**-0.0547**	0.0783
B-Y14	**-2.27**	**-0.0369**	**-3**	**-0.0447**	0.0757
B-Y15	**-7.6**	**-0.0818**	**-7**	**-0.0865**	0.0846
B-Y16	**-1.3**	**-0.0207**	**-1**	**-0.0635**	0.0761
B-Y17	**-6.33**	**-0.1224**	**-8**	**-0.1280**	0.0776
B-Y18	**-6.23**	**-0.0438**	**-6**	**-0.0555**	0.0795
B-Y19	**-10.07**	**-0.1875**	**-11**	**-0.1950**	0.0708
B-Y20	**-4.63**	**-0.0540**	**-4**	**-0.0831**	0.0839
Sha14	**-1.2**	**-0.0291**	**-1**	**-0.0539**	0.0425
Sha15	**-3.9**	**-0.0536**	**-3**	**-0.0788**	0.0496
SD12	**-1**	**-0.0499**	**-1**	**-0.0395**	0.0748
SD13	**-5**	**-0.1253**	**-5**	**-0.1258**	0.0775
Avg	**-3.66**	**-0.0774**	**-3.81**	**-0.0807**	0.0716
Total	**-58.53**	**-0.0774**	**-61**	**-0.0807**	0.0716

6.3.3 Further Analysis

Validating Effectiveness of Our Node Insertion Rules

We tested each rule's effectiveness, comparing LSP and LSC against LS in Tabs. 6.4 - 6.7. The key differences among LS, LSP, LSC, and AKS are as follows. LS does not use any node insertion rules or the AKI operator. In contrast, LSP and LSC employ node insertion rules, specifically using the Proximity and Connectivity rules, respectively. AKS further incorporates the AKI operator. Results show similar vehicle counts but significant *ttd* improvements with node insertion rules. For small-scale instances (Tabs. 6.4 and 6.6), LSP and LSC reduced *ttd* by an average of 6.95% and 7.34%, respectively. In large-scale instances (Tabs. 6.5 and 6.7), they achieved 10.57% and 8.88% average reductions, with maximum improvements of 18.94% and 16.02%.

Applying node insertion rules significantly reduced the number of fitness evaluations (FEs), particularly in large-scale problems. In small-scale instances (Tabs. 6.4 and 6.6), LSP and LSC used, on average, 57.41% and 64.67% of LS's FEs, respectively. For larger-scale problems (Tabs. 6.5 and 6.7), both LSP and LSC required less than half of LS's FEs, highlighting the improved computational efficiency of the proposed rules.

The results convincingly show the node insertion rules in this study significantly lowered *ttd* in most instances and enhanced computational efficiency.

Node Distribution Impacts Rule Performance

The performance of our node insertion rules, as seen in Tabs. 6.8 and 6.9, differs across instances. For example, LSP performs better on instances like SD11 and Sha15, while LSC outperforms on SD6 and SD7. This reveals that the effectiveness of the two rules varies by instance.

Table 6.4: Comparison results of LSP and LS algorithm on small-scale instances (# node < 100).

Instances (small)	Avg # v LSP-LS	Avg ttd (LSP-LS)/LS	Best # v LSP-LS	Best ttd (LSP-LS)/LS	# FE LSP/LS	Instances (small)	Avg # v LSP-LS	Avg ttd (LSP-LS)/LS	Best # v LSP-LS	Best ttd (LSP-LS)/LS	# FE LSP/LS
B-Y1	0	**-0.0482**	0	**-0.0273**	0.6198	Sha9	**-0.2**	**-0.0290**	0	**-0.0029**	0.6864
B-Y2	0	**-0.0680**	0	**-0.0641**	0.7107	Sha10	0	**-0.0405**	0	**-0.0233**	0.6378
B-Y3	0	**-0.0432**	0	**-0.0308**	0.5454	Sha11	0	0.0002	0	0.0092	0.6732
B-Y4	0.2	**-0.0568**	1	**-0.0213**	0.4623	Sha12	0	**-0.0455**	0	**-0.0101**	0.8690
B-Y5	0.1	**-0.0473**	1	**-0.0337**	0.4738	Sha13	0	**-0.0283**	0	**-0.0251**	0.7063
B-Y6	**-0.1**	**-0.0969**	**-1**	**-0.0525**	0.4720	SD1	0	**-0.0400**	0	0.0273	0.7761
B-Y7	0.3	**-0.0223**	0	**-0.0171**	0.6423	SD2	0	**-0.0719**	0	**-0.0536**	0.7381
B-Y8	0	**-0.0936**	0	**-0.0692**	0.4592	SD3	0	**-0.0784**	0	**-0.0576**	0.6670
Sha1	0	0	0	0	1.0691	SD4	0	**-0.0857**	0	**-0.0230**	0.5953
Sha2	0	0.0032	0	0	0.9095	SD5	0	**-0.0184**	0	**-0.0031**	0.6856
Sha3	0	**-0.0063**	0	0.0183	0.9034	SD6	0	**-0.0300**	0	**-0.0471**	0.7314
Sha4	0	0.0048	0	0.0319	0.7842	SD7	0	**-0.0388**	0	**-0.0237**	0.9601
Sha5	0	**-0.0115**	0	0.0000	1.0913	SD8	0	**-0.0995**	0	**-0.0702**	0.6577
Sha6	0	**-0.0326**	0	0	0.9091	SD9	**-0.1**	**-0.0942**	0	**-0.1010**	0.6492
Sha7	0	**-0.0107**	0	**-0.0043**	0.9207	SD10	**-0.2**	**-0.0999**	0	**-0.0755**	0.4840
Sha8	0	**-0.0086**	0	**-0.0009**	0.9101	SD11	0	**-0.1621**	0	**-0.0928**	0.4889

Overall	Avg # v	Avg ttd	Best # v	Best ttd	# FE
Avg	0	**-0.0695**	0.03	**-0.0469**	0.5741
Total	0	**-0.0695**	1	**-0.0469**	0.5741

Note: LS and AKS differ because LS does not use the AKI operator. The LSP algorithm uses the Proximity rules for node insertion. # v = the number of vehicles used; ttd = total travel distance; # FE = the number of fitness evaluations. The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Table 6.5: Comparison results of LSP and LS algorithms on large-scale instances (100≤#node≤200). LS and AKS differ because LS does not use the AKI operator. The LSP algorithm uses the Proximity rules for node insertion. # v = the number of vehicles used; ttd = total travel distance; # FE = the number of fitness evaluations. The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Instances (large)	Avg # v LSP-LS	Avg ttd (LSP-LS)/LS	Best # v LSP-LS	Best ttd (LSP-LS)/LS	# FE LSP/LS
B-Y9	0	**-0.0487**	0	**-0.0404**	0.5440
B-Y10	0.4	**-0.1001**	1	**-0.0543**	0.4203
B-Y11	0.2	**-0.0354**	0	**-0.0286**	0.4717
B-Y12	0.6	**-0.0728**	0	**-0.0439**	0.4685
B-Y13	0.4	**-0.0554**	0	**-0.0419**	0.4065
B-Y14	0.4	**-0.1151**	0	**-0.0835**	0.4005
B-Y15	0	**-0.0479**	0	**-0.0410**	0.4494
B-Y16	0.1	**-0.1098**	0	**-0.0874**	0.4654
B-Y17	0.3	**-0.0407**	**-1**	**-0.0255**	0.4831
B-Y18	0.4	**-0.1062**	1	**-0.1182**	0.4134
B-Y19	0.5	**-0.0440**	1	**-0.0387**	0.3177
B-Y20	0.2	**-0.0770**	1	**-0.0934**	0.4458
Sha14	0.3	**-0.0892**	0	**-0.0684**	0.5648
Sha15	0	**-0.1183**	1	**-0.1144**	0.5238
SD12	0.2	**-0.1139**	1	**-0.0719**	0.3707
SD13	0	**-0.1894**	0	**-0.1551**	0.3804
Avg	0.25	**-0.1057**	0.31	**-0.0804**	0.4333
Total	4	**-0.1057**	5	**-0.0804**	0.4333

Table 6.6: Comparison results of LSC and LS algorithm on small-scale instances (# node < 100).

Instances (small)	Avg # v LSC-LS	Avg ttd (LSC-LS)/LS	Best # v LSC-LS	Best ttd (LSC-LS)/LS	# FE LSC/LS	Instances (small)	Avg # v LSC-LS	Avg ttd (LSC-LS)/LS	Best # v LSC-LS	Best ttd (LSC-LS)/LS	# FE LSC/LS
B-Y1	0	**-0.0463**	0	**-0.0233**	0.7499	Sha9	**-0.5**	**-0.0282**	0	**-0.0188**	0.8039
B-Y2	0	**-0.0650**	0	**-0.0461**	0.8091	Sha10	0.1	**-0.0321**	1	0.0006	0.6166
B-Y3	0	**-0.0421**	0	**-0.0291**	0.7029	Sha11	0	**0.0181**	0	0.0340	0.6601
B-Y4	0.1	**-0.0460**	0	0.0016	0.5921	Sha12	0	**-0.0340**	0	0.0125	0.8816
B-Y5	0	**-0.0377**	0	**-0.0262**	0.6550	Sha13	0	**-0.0133**	0	**-0.0212**	0.8215
B-Y6	0	**-0.0805**	0	**-0.0451**	0.5436	SD1	0	**-0.0008**	0	0.0143	0.8432
B-Y7	0.2	**-0.0239**	0	**-0.0265**	0.6048	SD2	0	**-0.0665**	0	**-0.0546**	0.7522
B-Y8	0	**-0.0707**	0	**-0.0485**	0.5772	SD3	0	**-0.0698**	0	**-0.0413**	0.8998
Sha1	0	0	0	0	1.0556	SD4	0	**-0.0895**	0	**-0.0457**	0.6487
Sha2	0	0.0032	0	0	0.9095	SD5	0	**-0.0041**	0	0.0003	0.6511
Sha3	0	**-0.0150**	0	0.0183	1.0066	SD6	0	**-0.1104**	0	**-0.0741**	0.7151
Sha4	0	**-0.0116**	0	0.0139	0.7542	SD7	0	**-0.1141**	0	**-0.0525**	0.7806
Sha5	0	**-0.0096**	0	0	1.0378	SD8	0	**-0.0874**	0	**-0.0950**	0.7694
Sha6	0	**-0.0131**	0	0	0.9696	SD9	**-0.1**	**-0.1084**	0	**-0.0972**	0.6739
Sha7	0	**-0.0112**	0	**-0.0025**	1.1010	SD10	**-0.2**	**-0.0960**	0	**-0.0706**	0.7446
Sha8	0	**-0.0186**	0	**-0.0059**	0.9235	SD11	0	**-0.1184**	0	**-0.0408**	0.5044

Overall	Avg # v	Avg ttd	Best # v	Best ttd	# FE
Avg	**-0.01**	**-0.0734**	0.03	**-0.0484**	0.6467
Total	**-0.4**	**-0.0734**	1	**-0.0484**	0.6467

Note: LS and AKS differ because LS does not use the AKI operator. The LSC algorithm uses the Connectivity rules for node insertion. # v = the number of vehicles used; ttd = total travel distance; # FE = the number of fitness evaluations. The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Table 6.7: Comparison results of LSC and LS algorithms on large-scale instances (100≤#node≤200). LS and AKS differ because LS does not use the AKI operator. The LSC algorithm uses the Connectivity rules for node insertion. # v = the number of vehicles used; ttd = total travel distance; # FE = the number of fitness evaluations. The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Instances (large)	Avg # v LSC-LS	Avg ttd (LSC-LS)/LS	Best # v LSC-LS	Best ttd (LSC-LS)/LS	# FE LSC/LS
B-Y9	0	**-0.0431**	0	**-0.0249**	0.5578
B-Y10	**-0.2**	**-0.0851**	0	**-0.0347**	0.5271
B-Y11	0.2	**-0.0321**	0	**-0.0240**	0.4814
B-Y12	0.3	**-0.0517**	0	**-0.0359**	0.4434
B-Y13	0.1	**-0.0449**	0	**-0.0296**	0.4257
B-Y14	0	**-0.1076**	0	**-0.0826**	0.4970
B-Y15	0	**-0.0415**	0	**-0.0338**	0.4564
B-Y16	0.2	**-0.0940**	0	**-0.0769**	0.5195
B-Y17	0.4	**-0.0376**	**-1**	**-0.0234**	0.4133
B-Y18	0.4	**-0.1121**	1	**-0.1072**	0.4003
B-Y19	0.3	**-0.0369**	0	**-0.0341**	0.4150
B-Y20	0.1	**-0.0915**	1	**-0.0824**	0.5260
Sha14	0	**-0.0331**	0	**-0.0518**	0.7226
Sha15	0	**-0.0577**	1	**-0.0575**	0.5978
SD12	0.2	**-0.0953**	1	**-0.0630**	0.4179
SD13	**-0.1**	**-0.1602**	0	**-0.1295**	0.4894
Avg	0.12	**-0.0888**	0.19	**-0.0667**	0.4715
Total	1.9	**-0.0888**	3	**-0.0667**	0.4715

Table 6.8: Comparison results of LSP and LSC algorithm on small-scale instances (# node < 100).

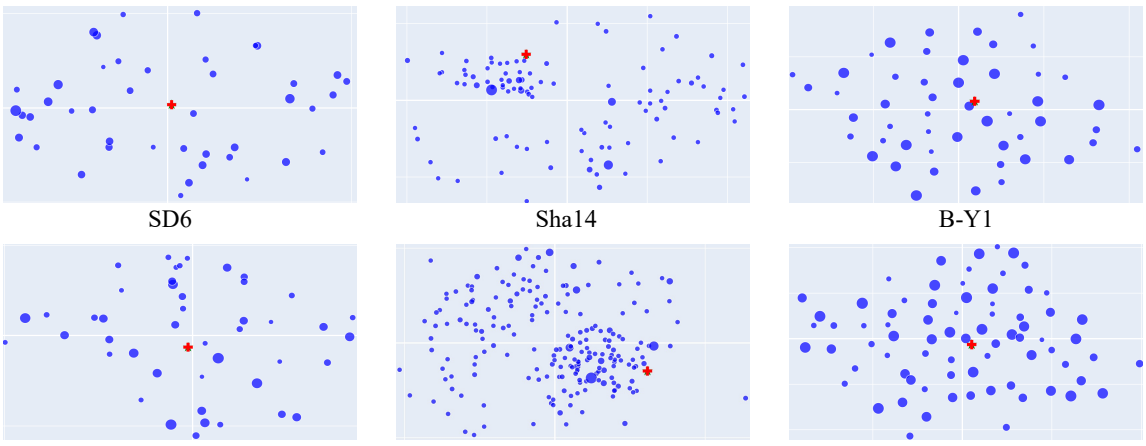
Instances (small)	Avg # v LSP-LSC	Avg ttd (LSP-LSC)/LSC	Best # v LSP-LSC	Best ttd (LSP-LSC)/LSC	# FE LSP/LSC	Instances (small)	Avg # v LSP-LSC	Avg ttd (LSP-LSC)/LSC	Best # v LSP-LSC	Best ttd (LSP-LSC)/LSC	# FE LSP/LSC
B-Y1	0	**-0.0019**	0	**-0.0041**	0.8265	Sha9	0.3	**-0.0008**	0	0.0162	0.8538
B-Y2	0	**-0.0032**	0	**-0.0189**	0.8783	Sha10	**-0.1**	**-0.0086**	**-1**	**-0.0239**	1.0343
B-Y3	0	**-0.0011**	0	**-0.0017**	0.7759	Sha11	0	**-0.0176**	0	**-0.0240**	1.0198
B-Y4	0.1	**-0.0114**	1	**-0.0229**	0.7807	Sha12	0	**-0.0119**	0	**-0.0223**	0.9857
B-Y5	0.1	**-0.0100**	1	**-0.0077**	0.7233	Sha13	0	**-0.0153**	0	**-0.0040**	0.8597
B-Y6	**-0.1**	**-0.0178**	**-1**	**-0.0078**	0.8683	SD1	0	**-0.0392**	0	0.0127	0.9204
B-Y7	0.1	0.0016	0	0.0096	1.0620	SD2	0	**-0.0058**	0	0.0011	0.9813
B-Y8	0	**-0.0246**	0	**-0.0217**	0.7955	SD3	0	**-0.0092**	0	**-0.0170**	0.7412
Sha1	0	0	0	0	1.0128	SD4	0	0.0042	0	0.0238	0.9177
Sha2	0	0	0	0	1	SD5	0	**-0.0143**	0	**-0.0033**	1.0531
Sha3	0	0.0089	0	0	0.8974	SD6	0	0.0903	0	0.0291	1.0228
Sha4	0	0.0166	0	0.0177	1.0398	SD7	0	0.0851	0	0.0305	1.2298
Sha5	0	**-0.0019**	0	0.0000	1.0515	SD8	0	**-0.0132**	0	0.0274	0.8547
Sha6	0	**-0.0198**	0	0	0.9376	SD9	0	0.0160	0	**-0.0042**	0.9634
Sha7	0	0.0005	0	**-0.0018**	0.8363	SD10	0	**-0.0043**	0	**-0.0052**	0.6500
Sha8	0	0.0102	0	0.0050	0.9855	SD11	0	**-0.0496**	0	**-0.0541**	0.9693

Overall	Avg # v	Avg ttd	Best # v	Best ttd	# FE
Avg	0.01	0.0042	0	0.0015	0.8878
Total	0.4	0.0042	0	0.0015	0.8878

Note: The LSP or LSC algorithms use the Proximity or Connectivity rules for node insertion respectively.# v = the number of vehicles used; ttd = total travel distance; # FE = the number of fitness evaluations. The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Table 6.9: Comparison results of LSP and LSC algorithm on large-scale instances (100≤#node≤200). The LSP or LSC algorithms use the Proximity or Connectivity rules for node insertion, respectively. # v = the number of vehicles used; ttd = total travel distance; # FE = the number of fitness evaluations. The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Instances (large)	Avg # v LSP-LSC	Avg ttd (LSP-LSC)/LSC	Best # v LSP-LSC	Best ttd (LSP-LSC)/LSC	# FE LSP/LSC
B-Y9	0	**-0.0059**	0	**-0.0158**	0.9752
B-Y10	0.6	**-0.0164**	1	**-0.0203**	0.7975
B-Y11	0	**-0.0034**	0	**-0.0047**	0.9798
B-Y12	0.3	**-0.0223**	0	**-0.0082**	1.0567
B-Y13	0.3	**-0.0110**	0	**-0.0127**	0.9550
B-Y14	0.4	**-0.0083**	0	**-0.0010**	0.8058
B-Y15	0	**-0.0067**	0	**-0.0074**	0.9845
B-Y16	**-0.1**	**-0.0174**	0	**-0.0114**	0.8959
B-Y17	**-0.1**	**-0.0032**	0	**-0.0022**	1.1690
B-Y18	0	0.0066	0	**-0.0124**	1.0326
B-Y19	0.2	**-0.0073**	1	**-0.0048**	0.7655
B-Y20	0.1	0.0160	0	**-0.0120**	0.8475
Sha14	0.3	**-0.0580**	0	**-0.0176**	0.7816
Sha15	0	**-0.0643**	0	**-0.0605**	0.8762
SD12	0	**-0.0206**	0	**-0.0095**	0.8870
SD13	0.1	**-0.0348**	0	**-0.0294**	0.7772
Avg	0.13	**-0.0185**	0.13	**-0.0146**	0.9189
Total	2.1	**-0.0185**	2	**-0.0146**	0.9189



(a) LSC has significantly better ttd. (b) LSP has significantly better ttd. (c) No significant difference.

Figure 6.3: Node distribution of different problem instances. (a) The ttd of LSP is significantly worse than LSC on SD6 and SD7. (b) The ttd of LSP is significantly better than LSC on Sha14 and Sha15. (c) There is no significant difference between LSP and LSC. For comparative ttd data of LSP and LSC, please refer to Tabs 6.8 and 6.9.

We analyzed LSP and LSC's performance regarding node distribution, shown in Fig. 6.3. LSP's ttd is worse than LSC's on some instances (like SD6 and SD7), but better in others (e.g., Sha14, Sha15). In some cases, such as B-Y1 and B-Y7, both algorithms show similar performance. The node distributions in these instances are significantly different intuitively.

Further analysis involved altering node distributions in some instances to observe performance changes. We used $\Delta = (ttd_{lsp} - ttd_{lsc})/ttd_{lsc}$ to compare LSP's and LSC's ttd. A negative Δ means LSP performed better, while a positive one indicates the opposite. Changes in SD6 and SD7 reduced the ttd gap between LSP and LSC, as Fig. 6.4 shows. Meanwhile, SD1, SD8, and SD9, initially with minimal differences, showed significant changes after modification.

Moreover, we modified the node distribution of existing instances to generate 19 new instances. Tab. 6.10 shows features of new problem instances created based on

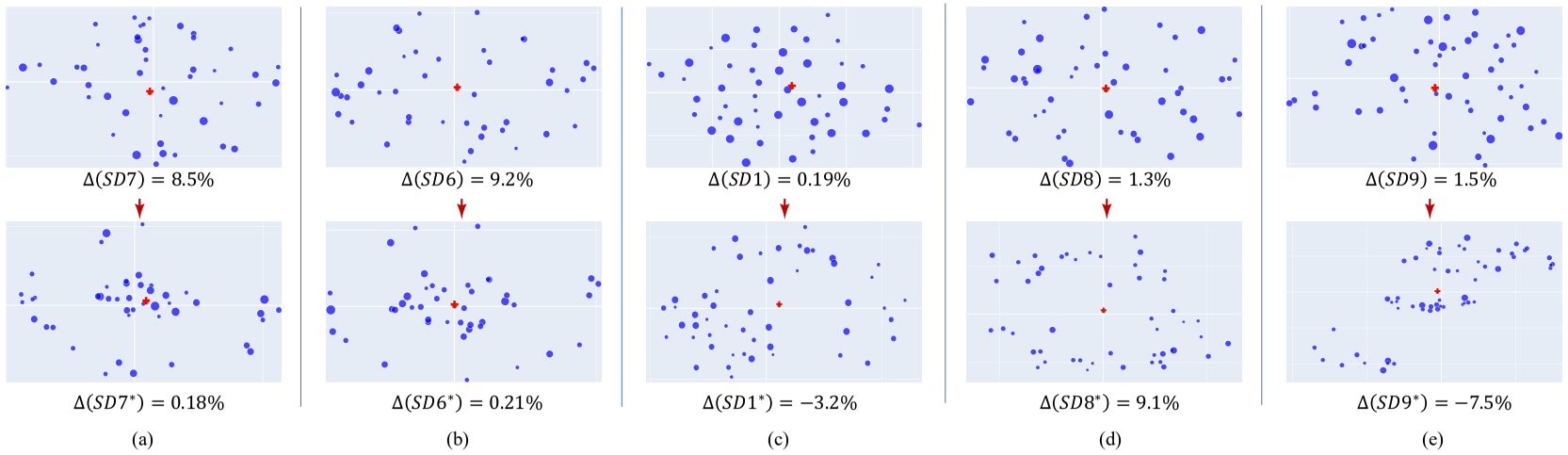


Figure 6.4: Comparing LSP and LSC on modified instances with new node layouts. E.g., SD7 was altered to SD7*. We tracked each algorithm's *ttd* and used Δ to see which performed better: negative Δ indicates LSP was more effective, while positive Δ favours LSC.

Table 6.10: Description of new instances

Entry	# nodes	# box types	# boxes
Min	43	14	284
Max	200	634	8060
Avg	85.3	146.1	4473.8

existing ones by changing the node distribution. Results in Tab. 6.11 show node distribution's significant effect on rule effectiveness. LSP performed well on eight instances but was less effective on another ten instances compared to LSC.

This emphasizes the significant influence of node distribution on the suitability and performance of our insertion rules, indicating that the selection of a rule should correspond to the specific characteristics of each instance's node distribution.

Further Validating Our AKI Operator

To further validate our AKI operator, we generated 19 new problem instances with varied node distributions by modifying existing instances (characteristics of these new instances are presented in Tab. 6.10). We then compared the performance of the AKS algorithm (incorporating the AKI operator) against LSP, LSC, and LSrdm (which alternates randomly between two rules) on these newly created instances. As shown in Tabs. 6.12 - 6.14, AKS consistently reduced *ttd* more than LSP (nine instances), LSC (18 instances), and LSrdm (ten instances). This shows AKI's ability to adaptively select the best rule for node insertion, efficiently handling diverse node distributions by blending the strengths of both rules.

Table 6.11: Comparison results between LSP and LSC algorithms on new instances. The LSP or LSC algorithms use the Proximity or Connectivity rules for node insertion, respectively. # v = the number of vehicles used; ttd=total travel distance; # FE= the number of fitness evaluations.The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Instance	Avg #v LSP-LSC	Avg ttd (LSP-LSC)/LSC	Best #v LSP-LSC	Best ttd (LSP-LSC)/LSC	#FE LSP/LSC
Ins1	0	0.0630	0	**-0.0038**	1.2593
Ins2	0.1	0.0728	0	0.0169	1.1064
Ins3	0	0.0518	0	0.0036	1.1660
Ins4	0	0.0685	0	0.0311	1.0359
Ins5	0	0.0070	0	**-0.0037**	1.1053
Ins6	0	0.0352	0	0.0474	1.0659
Ins7	0	-0.0030	0	0.0075	0.8502
Ins8	0	0.0567	0	**-0.0126**	1.2233
Ins9	0	0.0328	0	0.0031	1.1607
Ins10	0	**-0.0842**	0	**-0.0468**	0.8297
Ins11	0	**-0.0726**	0	**-0.0641**	0.7540
Ins12	0	**-0.0612**	0	**-0.0467**	0.9289
Ins13	0	**-0.0617**	0	**-0.0584**	1.0223
Ins14	0	**-0.0288**	0	**-0.0049**	0.9382
Ins15	0	**-0.0301**	0	**-0.0232**	0.8478
Ins16	0.4	**-0.0723**	1	**-0.0483**	0.7220
Ins17	0.1	**-0.0556**	1	**-0.0353**	0.8045
Ins18	0.1	**-0.0360**	0	**-0.0248**	0.9742
Ins19	0	**-0.0489**	0	**-0.0249**	0.7892
Avg	0.04	0.0001	0.11	**-0.0114**	0.8851
Total	0.7	0.0001	2	**-0.0114**	0.8851

Table 6.12: Comparison results between AKS and LSP algorithms on new instances. AKS employs the AKI operator for node insertion, while LSP utilizes the Proximity rule. # v = the number of vehicles used; ttd=total travel distance; # FE= the number of fitness evaluations.The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Instance	Avg #v AKS-LSP	Avg ttd (AKS-LSP)/LSP	Best #v AKS-LSP	Best ttd (AKS-LSP)/LSP	#FE AKS/LSP
Ins1	0	**-0.0797**	0	**-0.0197**	0.8067
Ins2	**-0.1**	**-0.0880**	0	**-0.0137**	0.8470
Ins3	0	**-0.0703**	0	0.0137	0.7887
Ins4	0	**-0.0844**	0	**-0.0347**	0.7499
Ins5	0	**-0.0307**	0	**-0.0133**	0.8268
Ins6	0	**-0.0654**	0	**-0.0172**	0.7612
Ins7	0	**-0.0263**	0	**-0.0416**	1.0159
Ins8	0	**-0.0742**	0	0.0107	0.7262
Ins9	0	**-0.0469**	0	**-0.0141**	0.7359
Ins10	0	**-0.0019**	0	0.0050	1.0684
Ins11	0	0.0041	0	**-0.0199**	1.1614
Ins12	0	0.0071	0	**-0.0009**	0.9805
Ins13	0	0.0037	0	0.0302	0.8865
Ins14	0	0.0026	0	0.0101	0.9940
Ins15	0	**-0.0131**	0	**-0.0030**	0.9679
Ins16	**-0.03**	0.0015	**-1**	**-0.0187**	1.1115
Ins17	**-0.07**	0.0178	**-1**	0.0029	0.9862
Ins18	0	**-0.0157**	1	**-0.0085**	0.8551
Ins19	0	0.0095	0	0.0170	1.0586
Avg	**-0.01**	**-0.0365**	**-0.05**	**-0.0061**	0.9559
Total	**-0.2**	**-0.0365**	**-1**	**-0.0061**	0.9559

Table 6.13: Comparison results between AKS and LSC algorithms on new instances. AKS employs the AKI operator for node insertion, while LSC utilizes the Connectivity rule. # v = the number of vehicles used; ttd=total travel distance; # FE = the number of fitness evaluations.The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Instance	Avg #v AKS-LSC	Avg ttd (AKS-LSC)/LSC	Best #v AKS-LSC	Best ttd (AKS-LSC)/LSC	#FE AKS/LSC
Ins1	0	**-0.0218**	0	**-0.0235**	1.0159
Ins2	0	**-0.0216**	0	0.0030	0.9372
Ins3	0	**-0.0221**	0	0.0173	0.9196
Ins4	0	**-0.0217**	0	**-0.0047**	0.7768
Ins5	0	**-0.0239**	0	**-0.0169**	0.9139
Ins6	0	**-0.0325**	0	0.0295	0.8114
Ins7	0	**-0.0292**	0	**-0.0344**	0.8637
Ins8	0	**-0.0218**	0	**-0.0021**	0.8884
Ins9	0	**-0.0157**	0	**-0.0111**	0.8541
Ins10	0	**-0.0859**	0	**-0.0421**	0.8865
Ins11	0	**-0.0688**	0	**-0.0827**	0.8757
Ins12	0	**-0.0546**	0	**-0.0476**	0.9107
Ins13	0	**-0.0582**	0	**-0.0300**	0.9062
Ins14	0	**-0.0263**	0	**0.0051**	0.9326
Ins15	0	**-0.0429**	0	**-0.0261**	0.8207
Ins16	0.37	**-0.0709**	0	**-0.0661**	0.8025
Ins17	0.03	**-0.0387**	0	**-0.0325**	0.7933
Ins18	0.1	**-0.0511**	1	**-0.0331**	0.8330
Ins19	0	**-0.0398**	0	**-0.0083**	0.8354
Avg	0.03	**-0.0364**	0.05	**-0.0175**	0.8461
Total	0.5	**-0.0364**	1	**-0.0175**	0.8461

Table 6.14: Comparison results between AKS and LSrdm (rdm) algorithms on new instances. AKS employs the AKI operator for node insertion, while LSrdm randomly selects between Proximity and Connectivity rules for node insertion. # v = the number of vehicles used; ttd = total travel distance; # FE = the number of fitness evaluations.The data in bold signifies that our strategy produces better results than the contrasted strategies numerically. The Mann-Whitney U test was employed, and the data with underlined values indicate a significant difference at the significance level $\alpha = 0.05$.

Instance	Avg #v AKS-rdm	Avg ttd (AKS-rdm)/rdm	Best #v AKS-rdm	Best ttd (AKS-rdm)/rdm	#FE AKS/rdm
Ins1	0	**-0.0066**	0	**-0.0127**	1.0910
Ins2	0	**-0.0019**	0	**-0.0114**	1.0487
Ins3	0	0.0058	0	0.0235	1.0224
Ins4	0	**-0.0139**	0	0.0123	0.9831
Ins5	0	**-0.0092**	0	**-0.0022**	1.0438
Ins6	0	**-0.0071**	0	0.0132	1.0313
Ins7	0	**-0.0022**	0	**-0.0283**	1.0997
Ins8	0	**-0.0047**	0	0.0038	0.9949
Ins9	0	0.0073	0	**-0.0204**	0.9846
Ins10	0	**-0.0408**	0	**-0.0023**	0.7547
Ins11	0	**-0.0322**	0	**-0.0531**	1.0764
Ins12	0	**-0.0431**	0	**-0.0637**	1.1278
Ins13	0	**-0.0233**	0	**-0.0130**	0.9760
Ins14	0	**-0.0236**	0	**-0.0272**	0.9679
Ins15	0	**-0.0288**	0	**-0.0373**	1.0100
Ins16	0.37	**-0.0288**	0	**-0.0287**	0.8527
Ins17	0.33	**-0.0419**	0	**-0.0450**	0.8633
Ins18	0	**-0.0337**	0	**-0.0398**	0.9857
Ins19	0	**-0.0214**	0	**-0.0141**	1.0112
Avg	0.04	**-0.0163**	0	**-0.0160**	0.9797
Total	0.7	**-0.0163**	0	**-0.0160**	0.9797

6.4 Conclusion

This chapter presents a knowledge-guided approach for solving complex 3L-SDVRP. We hypothesize that in an effective giant tour, physically proximate nodes should be adjacent or close in a node sequence. To realize this, we employ node insertion to modify a giant tour and develop two node insertion rules: Proximity and Connectivity. Further, we develop the AKI operator, which adaptively selects the most appropriate node insertion rules based on node distribution characteristics and offers a larger search step size than traditional methods. The integration of the AKI operator within a local search framework leads to the development of the AKS algorithm.

Extensive experimental analyses validate the effectiveness of our proposed AKI operator and AKS algorithm in improving search efficiency and solution quality, particularly in reducing the total travel distance (ttd). This highlights the importance of a deep understanding of problem characteristics, the strategic use of domain knowledge to guide the search process, and the appropriate balance of small and large search step sizes in intelligent optimization algorithms. Although this chapter concentrates on the 3L-SDVRP, our proposed AKI operator and AKS algorithm could be adapted to solve other VRP variants. Future research in this field should focus on further exploring the integration of domain knowledge with optimization algorithms. This could involve investigating additional problem characteristics, refining the AKI operator, or applying the AKS algorithm to other complex optimization scenarios.

Chapter 7

Conclusion and Future Directions

The Split Delivery Vehicle Routing Problem with Three-Dimensional Loading Constraints (3L-SDVRP) is a complex combinatorial optimization problem, which combines two NP-hard problems: vehicle routing and three-dimensional packing. In 3L-SDVRP, a fleet of vehicles depart from the starting point (depot), visit various nodes to load boxes, and ultimately return to the ending point (the starting and ending points may differ depending on the scenario). The objective is to minimize the number of vehicles required and the total travel distance (ttd) while ensuring all boxes from each node are fully loaded. Fundamentally, any 3L-SDVRP solution must address both routing decisions (determining which nodes each vehicle should visit) and packing decisions (designing the 3D packing plan for each vehicle). Effectively and efficiently solving the 3L-SDVRP remains a significant challenge in both academic and industrial contexts.

This chapter concludes the thesis by summarizing our comprehensive research on the 3L-SDVRP problem. In Section 7.1, we provide a chapter-wise recap of our work, highlighting the key innovations and insights gained throughout the study. Building upon this summary, we emphasize the significant contributions made by this thesis. Furthermore, Section 7.2 explores potential directions for future research, identifying

promising opportunities for further investigation and development on 3L-SDVRP.

7.1 Conclusion

This thesis presents a comprehensive study on 3L-SDVRP, focusing on various aspects of combinatorial optimization and algorithmic design. In Chapter 3, we developed a Hierarchical Neighborhood Filtering (HNF) mutation operator characterized by its use of diverse neighborhood structures, such as swap, 2-opt, and 3-opt, to generate a wide range of offspring from a single parent, thereby improving solution diversity and algorithm exploitation capability. This operator employs a hierarchical approach to mutation, prioritizing individuals with higher nondomination ranks to concentrate the search on promising candidates. Integrating the HNF mutation into the Evolutionary Algorithm (EA) framework resulted in the development of a novel Pareto-based Evolutionary Algorithm with Concurrent crossover and Hierarchical Neighborhood Filtering mutation (PEAC-HNF) for 3L-SDVRP. This approach enhances the balance between exploration and exploitation, demonstrating significant improvements in effectiveness. Experimental studies validated the crucial role of the HNF mutation in enhancing algorithmic performance.

In Chapter 4, we introduced a more efficient algorithm based on the state-of-the-art SDVRLH2 algorithm for solving 3L-SDVRP, significantly enhancing search efficiency and solution quality. The proposed algorithm improves the box loading, subspace generation, and 2C-SP construction methods in the loading procedure, thereby enhancing loading performance and reducing the number of vehicles used. Additionally, three new search operators were designed, leveraging the problem's characteristics as heuristic information to improve search efficiency. An adaptive splitting strategy was proposed, dynamically determining whether to split a node's boxes based on the status of the vehicle and node, thereby further reducing computational resource consumption. Furthermore, a new post-optimization method was developed to further

reduce the number of vehicles required.

Chapter 5 proposed an adaptive interactive routing-packing strategy that integrates the advanced features of existing interactive strategies. Our strategy introduces adaptability in the loading process during routing, allowing for a choice between independent loading of a node (aligned with the P1R2 strategy) and joint loading of consecutive nodes (utilizing the idea of 2C-SP). This flexible approach permits loading adjustments in response to route modifications, effectively reflecting the core principle of the R1P2 strategy. The effectiveness of our strategy was rigorously validated through computational experiments, yielding solutions that are comparable to or significantly superior to existing strategies, particularly in terms of the number of vehicles used. Additionally, despite the crucial role of interactive routing-packing strategies in addressing the complex 3L-SDVRP, there has been a lack of comprehensive investigation or comparison of these strategies. Our research fills this gap by providing a detailed comparative analysis and evaluation of existing interactive routing-packing strategies, substantiated by extensive experimental validation.

In Chapter 6, we integrated domain knowledge into the optimization algorithm to effectively guide the search process. Heuristics were extracted from domain knowledge, particularly using the "giant tour" representation. Through observation, we developed a hypothesis about "what constitutes a good giant tour" and employed a node insertion approach to rearrange the order of nodes in the current giant tour, aiming to enhance its quality. Two node insertion rules were proposed: the Proximity rule and the Connectivity rule. An Adaptive Knowledge-guided Insertion (AKI) operator was developed, which adaptively selects suitable node insertion rules based on node distribution. This AKI operator, leveraging domain knowledge and featuring a large search step size, was integrated into a local search framework to form the Adaptive Knowledge-guided Search (AKS) algorithm. In the AKS algorithm, traditional neighborhood search operators conduct searches with small step sizes (exploitation), while our AKI operator performs searches with larger step sizes (exploration), thereby

improving the algorithm's search capability. Comprehensive experimental results validated the effectiveness of the AKS algorithm.

In summary, the contributions of this thesis include:

- More Efficient Search Operators: We introduced new efficient search operators, e.g., the Hierarchical Neighborhood Filtering (HNF) and Adaptive Knowledge-guided Insertion (AKI) operators, which significantly enhance algorithm performance.

- New Methods for Balancing Exploration and Exploitation: We explored methods to balance exploration and exploitation in meta-heuristic algorithms, proposing new approaches for both global and local search-based algorithms.

- Novel Multi-Objective Algorithm: We developed a novel multi-objective algorithm, PEAC-HNF, to effectively solve the 3L-SDVRP under limited computational resources.

- More Effective Local Search Algorithms: We proposed new local search algorithms that significantly improve solution quality and reduce computational resource consumption.

- Adaptive Interactive Routing-Packing Strategy: We introduced an adaptive interactive routing-packing strategy capable of making adaptive decisions on different packing patterns based on specific situations. By combining the strengths of existing strategies, this approach improves solution quality.

- Extensive Experimental Studies: Through extensive experimental validation and comparative analysis, we demonstrated the superior performance of our proposed algorithms across various widely used benchmark datasets, showing significant improvements in solution quality and computational efficiency.

It is noteworthy that the methods proposed in this thesis, such as the HNF operator, the AKI operator, and the approach for balancing exploration and exploitation, are not limited to solving the 3L-SDVRP. These methods can also be applied to other complex combinatorial optimization problems.

7.2 Future Directions

As human society advances, the challenges encountered in real industrial scenarios are becoming increasingly complex and large-scale. With advancements in computer algorithms and the rapid improvement in hardware computing power, researchers are continually exploring solutions for these more complex and large-scale problems, which are closer to real-world scenarios. Consequently, it is becoming increasingly feasible to provide better solutions for complex combinatorial optimization problems. Future research directions for the 3L-SDVRP include the following aspects:

- Balancing Dual Objectives: The 3L-SDVRP encompasses two primary objectives. Investigating the interplay between these objectives and developing more efficient multi-objective optimization algorithms to achieve high-quality solutions for both is a challenging and critical area of research.
- Dynamic Nature and Uncertainty of Problems: Real-world scenarios are often dynamic and uncertain. Factors such as the number of boxes required to load at each customer node can change, new delivery demands may arise, and decision variables and objectives can evolve over time [36], [37], [54], [83], [95], [96]. Effectively addressing these dynamic and uncertain conditions remains a significant challenge.
- Scalability of Algorithms: In the era of globalization and industrial expansion, the increasing scale of real-world optimization problems has led to an exponential growth in solution space complexity, compounded by complex constraints,

making it increasingly difficult to conduct effective searches within this vast solution landscape. Consequently, the scalability of algorithms has become critical. Enhancing algorithm performance to ensure efficient and effective search capabilities in tackling these large-scale problems is a critical and significant research direction.

- Domain Knowledge-Guided Search: Meta-heuristic algorithms inherently possess global search capabilities due to their ability to explore the solution space stochastically. This characteristic allows them to potentially escape local optima and move towards finding optimal or near-optimal solutions. However, when dealing with highly constrained and large-scale problems, these methods become inefficient and demand substantial computational resources. By extracting domain knowledge through a deep understanding of the problem and integrating this knowledge into the algorithm to guide the search process, we can significantly enhance search efficiency and overall algorithm performance.

- Utilizing Machine Learning Methods: Machine learning, particularly deep learning, is rapidly advancing and achieving remarkable results in various recognition tasks. Integrating meta-heuristic algorithms with machine learning techniques to solve decision optimization problems more swiftly and effectively is an emerging field. This approach has garnered significant interest from researchers and has been explored in various contexts [5] [33] [43] [53] [87] [90].

By addressing these research directions, future studies can further advance the field of combinatorial optimization and develop more robust and efficient solutions for the 3L-SDVRP and other similarly complex problems.

References

[1] Rafael E Aleman, Xinhui Zhang, and Raymond R Hill. An adaptive memory algorithm for the split delivery vehicle routing problem. *Journal of Heuristics*, 16(3):441–473, 2010.

[2] Nabila Azi, Michel Gendreau, and Jean-Yves Potvin. An adaptive large neighborhood search for a vehicle routing problem with multiple routes. *Computers & Operations Research*, 41:167–173, 2014.

[3] Cynthia Barnhart and Gilbert Laporte. *Handbooks in Operations Research and Management Science: Transportation*. Elsevier, 2006.

[4] John E Beasley. Route first—cluster second methods for vehicle routing. *Omega*, 11(4):403–408, 1983.

[5] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d' horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

[6] Andreas Bortfeldt. A hybrid algorithm for the capacitated vehicle routing problem with three-dimensional loading constraints. *Computers & Operations Research*, 39(9):2248–2257, 2012.

[7] Andreas Bortfeldt and Hermann Gehring. A hybrid genetic algorithm for the container loading problem. *European Journal of Operational Research*, 131(1):143–161, 2001.

[8] Andreas Bortfeldt, Thomas Hahn, Dirk Männel, and Lars Mönch. Hybrid algorithms for the vehicle routing problem with clustered backhauls and 3d loading constraints. *European Journal of Operational Research*, 243(1):82–96, 2015.

[9] Andreas Bortfeldt and Jörg Homberger. Packing first, routing second—a heuristic for the vehicle routing and loading problem. *Computers & Operations Research*, 40(3):873–885, 2013.

[10] Andreas Bortfeldt and Junmin Yi. The split delivery vehicle routing problem with three-dimensional loading constraints. *European Journal of Operational Research*, 282(2):545–558, 2020.

[11] Lei Cao, Chun-ming Ye, Ran Cheng, and Zhen-kun Wang. Memory-based variable neighborhood search for green vehicle routing problem with passing-by drivers: A comprehensive perspective. *Complex & Intelligent Systems*, 8(3):2507–2525, 2022.

[12] Sara Ceschia and Andrea Schaerf. Local search for a multi-drop multi-container loading problem. *Journal of Heuristics*, 19(2):275–294, 2013.

[13] Sara Ceschia, Andrea Schaerf, and Thomas Stützle. Local search techniques for a routing-packing problem. *Computers & Industrial Engineering*, 66(4):1138–1149, 2013.

[14] Zongyi Chen, Mingkang Yang, Yijun Guo, Yu Liang, Yifan Ding, and Li Wang. The split delivery vehicle routing problem with three-dimensional loading and time windows constraints. *Sustainability*, 12(17):6987, 2020.

[15] Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei. Extreme point-based heuristics for three-dimensional bin packing. *Informs Journal on computing*, 20(3):368–384, 2008.

[16] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[17] Tansel Dokeroglu and Ender Sevinc. Memetic teaching–learning-based optimization algorithms for large graph coloring problems. *Engineering Applications of Artificial Intelligence*, 102:104282, 2021.

[18] Christophe Duhamel, Philippe Lacomme, and Caroline Prodhon. Efficient frameworks for greedy split and new depth first search split procedures for routing problems. *Computers & Operations Research*, 38(4):723–739, 2011.

[19] Dorian Dumez, Fabien Lehuédé, and Olivier Péton. A large neighborhood search approach to the vehicle routing problem with delivery options. *Transportation Research Part B: Methodological*, 144:103–132, 2021.

[20] Racha El-Hajj, Rym Nesrine Guibadj, Aziz Moukrim, and Mehdi Serairi. A PSO based algorithm with an efficient optimal split procedure for the multiperiod vehicle routing problem with profit. *Annals of Operations Research*, 291(1):281–316, 2020.

[21] Luis Miguel Escobar-Falcón, David Álvarez-Martínez, Mauricio Granada-Echeverri, John Willmer Escobar, and Rubén Augusto Romero-Lázaro. A matheuristic algorithm for the three-dimensional loading capacitated vehicle routing problem (3l-cvrp). *Revista Facultad de Ingeniería Universidad de Antioquia*, (78):09–20, 2016.

[22] Christian Friedrich and Ralf Elbert. Adaptive large neighborhood search for vehicle routing problems with transshipment facilities arising in city logistics. *Computers & Operations Research*, 137:105491, 2022.

[23] Guenther Fuellerer, Karl F Doerner, Richard F Hartl, and Manuel Iori. Metaheuristics for vehicle routing problems with three-dimensional loading constraints. *European Journal of Operational Research*, 201(3):751–759, 2010.

[24] Liang Gao, Guohui Zhang, Liping Zhang, and Xinyu Li. An efficient memetic algorithm for solving the job shop scheduling problem. *Computers & Industrial Engineering*, 60(4):699–705, 2011.

[25] H. Gehring, K. Menschner, and M. Meyer. A computer-based heuristic for packing pooled shipment containers. *European Journal of Operational Research*, 44(2):277–288, 1990.

[26] Michel Gendreau, Manuel Iori, Gilbert Laporte, and Silvano Martello. A tabu search algorithm for a routing and container loading problem. *Transportation Science*, 40(3):342–350, 2006.

[27] John A George and David F Robinson. A heuristic for packing boxes into a container. *Computers & Operations Research*, 7(3):147–156, 1980.

[28] Bruce L Golden, James S DeArmon, and Edward K Baker. Computational experiments with algorithms for a class of routing problems. *Computers & Operations Research*, 10(1):47–59, 1983.

[29] Bruce L Golden, Subramanian Raghavan, and Edward A Wasil. *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43. Springer Science & Business Media, 2008.

[30] Guiliang Gong, Qianwang Deng, Raymond Chiong, Xuran Gong, and Hezhiyuan Huang. An effective memetic algorithm for multi-objective job-shop scheduling. *Knowledge-Based Systems*, 182:104840, 2019.

[31] Oscar M González, Carlos Segura, S Ivvan Valdez Peña, and Coromoto León. A memetic algorithm for the capacitated vehicle routing problem with time windows. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 2582–2589. IEEE, 2017.

[32] Aldy Gunawan, Audrey Tedja Widjaja, Pieter Vansteenwegen, and F Yu Vincent. Adaptive large neighborhood search for vehicle routing problem with cross-docking. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2020.

[33] Abhishek Gupta, Yew-Soon Ong, and Liang Feng. Insights on transfer optimization: Because experience is the best teacher. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(1):51–64, 2018.

[34] Franklin T Hanshar and Beatrice M Ombuki-Berman. Dynamic vehicle routing using genetic algorithms. *Applied Intelligence*, 27(1):89–99, 2007.

[35] Keld Helsgaun. General k-opt submoves for the Lin–Kernighan TSP heuristic. *Mathematical Programming Computation*, 1:119–163, 2009.

[36] Daniel Herring, Michael Kirley, and Xin Yao. Responsive multi-population models for the dynamic travelling thief problem. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 297–304, 2020.

[37] Daniel Herring, Michael Kirley, and Xin Yao. A comparative study of evolutionary approaches to the bi-objective dynamic travelling thief problem. *Swarm and Evolutionary Computation*, 84:101433, 2024.

[38] Zhi-Hua Hu, Yingxue Zhao, Sha Tao, and Zhao-Han Sheng. Finished-vehicle transporter routing problem solved by loading pattern discovery. *Annals of Operations Research*, 234(1):37–56, 2015.

[39] Huawei. EMO2021 Huwawei logistics competion, 2021. Retrieved January 26, 2024 from https://www.noahlab.com.hk/logistics-ranking/#/home/the-competition.

[40] Huawei. EMO2021 Huwawei logistics competion award announcement, 2021. Retrieved January 28, 2024 from https://www.noahlab.com.hk/logistics-ranking/#/result.

[41] Huawei. EMO2021 Huwawei logistics competion details, 2021. Retrieved January 28, 2024 from https://www.noahlab.com.hk/logistics-ranking/#/competition_details.

[42] Leonardo Junqueira and Reinaldo Morabito. Heuristic algorithms for a three-dimensional loading capacitated vehicle routing problem in a carrier. *Computers & Industrial Engineering*, 88:110–130, 2015.

[43] Maryam Karimi-Mamaghan, Mehrdad Mohammadi, Patrick Meyer, Amir Mohammad Karimi-Mamaghan, and El-Ghazali Talbi. Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *European Journal of Operational Research*, 296(2):393–422, 2022.

[44] Selma Khebbache-Hadji, Christian Prins, Alice Yalaoui, and Mohamed Reghioui. Heuristics and memetic algorithm for the two-dimensional loading capacitated vehicle routing problem with time windows. *Central European Journal of Operations Research*, 21(2):307–336, 2013.

[45] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[46] Henriette Koch, Andreas Bortfeldt, and Gerhard Wäscher. A hybrid algorithm for the vehicle routing problem with backhauls, time windows and three-dimensional loading constraints. *OR Spectrum*, 40(4):1029–1075, 2018.

[47] Henriette Koch, Maximilian Schlögell, and Andreas Bortfeldt. A hybrid algorithm for the vehicle routing problem with three-dimensional loading constraints and mixed backhauls. *Journal of Scheduling*, 23(1):71–93, 2020.

[48] Philippe Lacomme, Hélène Toussaint, and Christophe Duhamel. A GRASP×ELS for the vehicle routing problem with basic three-dimensional loading constraints. *Engineering Applications of Artificial Intelligence*, 26(8):1795–1810, 2013.

[49] Wenxing Lan, Ziyuan Ye, Peijun Ruan, Jialin Liu, Peng Yang, and Xin Yao. Region-focused memetic algorithms with smart initialization for real-world large-scale waste collection problems. *IEEE Transactions on Evolutionary Computation*, 26(4):704–718, 2022.

[50] Xijun Li, Mingxuan Yuan, Di Chen, Jianguo Yao, and Jia Zeng. A data-driven three-layer algorithm for split delivery vehicle routing problem with 3D container loading constraint. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 528–536, 2018.

[51] Fei Liu, Qingfu Zhang, Qingling Zhu, Xialiang Tong, and Mingxuan Yuan. Machine learning assisted multiobjective evolutionary algorithm for routing and packing. *IEEE Transactions on Evolutionary Computation*, early access, 2024. https://ieeexplore.ieee.org/abstract/document/10423590.

[52] Shengcai Liu, Ke Tang, and Xin Yao. Memetic search for vehicle routing with simultaneous pickup-delivery and time windows. *Swarm and Evolutionary Computation*, page 100927, 2021.

[53] Shengcai Liu, Yu Zhang, Ke Tang, and Xin Yao. How good is neural combinatorial optimization? a systematic evaluation on the traveling salesman problem. *IEEE Computational Intelligence Magazine*, 18(3):14–28, 2023.

[54] Xiaofen Lu, Ke Tang, Stefan Menzel, and Xin Yao. A competitive co-evolutionary optimization method for the dynamic vehicle routing problem. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 305–312, 2020.

[55] Zhipeng Lü and Jin-Kao Hao. A memetic algorithm for graph coloring. *European Journal of Operational Research*, 203(1):241–250, 2010.

[56] Wissam F. Maarouf, Aziz M. Barbar, and Michel J. Owayjan. A new heuristic algorithm for the 3d bin packing problem. In Khaled Elleithy, editor, *Innovations and Advanced Techniques in Systems, Computing Sciences and Software Engineering*, pages 342–345, Dordrecht, 2008. Springer Netherlands.

[57] Batoul Mahvash, Anjali Awasthi, and Satyaveer Chauhan. A column generation based heuristic for the capacitated vehicle routing problem with three-dimensional loading constraints. *International Journal of Production Research*, 55(6):1730–1747, 2017.

[58] Jacek Mańdziuk and Adam Żychowski. A memetic approach to vehicle routing problem with dynamic requests. *Applied Soft Computing*, 48:522–534, 2016.

[59] Dirk Männel and Andreas Bortfeldt. A hybrid algorithm for the vehicle routing problem with pickup and delivery and three-dimensional loading constraints. *European Journal of Operational Research*, 254(3):840–858, 2016.

[60] Youssef Meliani, Yasmina Hani, Sâad Lissane Elhaq, and Abderrahman El Mhamedi. A tabu search based approach for the heterogeneous fleet vehicle routing problem with three-dimensional loading constraints. *Applied Soft Computing*, 126:109239, 2022.

[61] Raul Mencia, Maria R Sierra, Carlos Mencia, and Ramiro Varela. Memetic algorithms for the job shop scheduling problem with operators. *Applied Soft Computing*, 34:94–105, 2015.

[62] Lixin Miao, Qingfang Ruan, Kevin Woghiren, and Qi Ruo. A hybrid genetic algorithm for the vehicle routing problem with three-dimensional loading constraints. *RAIRO-Operations Research-Recherche Opérationnelle*, 46(1):63–82, 2012.

[63] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[64] Pablo Moscato et al. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech Concurrent Computation Program, C3P Report*, 826:1989, 1989.

[65] Ana Moura. A multi-objective genetic algorithm for the vehicle routing with time windows and loading problem. In *Intelligent Decision Support*, pages 187–201. Springer, 2008.

[66] Ana Moura and José Fernando Oliveira. An integrated approach to the vehicle routing and container loading problems. *OR Spectrum*, 31(4):775–800, 2009.

[67] B Ombuki, Morikazu Nakamura, and O Maeda. A hybrid search based on genetic algorithms and tabu search for vehicle routing. In *6th IASTED Intl. Conf. On Artificial Intelligence and Soft Computing (ASC 2002)*, pages 176–181, 2002.

[68] Beatrice Ombuki, Brian J Ross, and Franklin Hanshar. Multi-objective genetic algorithms for vehicle routing problem with time windows. *Applied Intelligence*, 24(1):17–30, 2006.

[69] Shannon Pace, Ayad Turky, Irene Moser, and Aldeida Aleti. Distributing fibre boards: a practical application of the heterogeneous fleet vehicle routing problem with time windows and three-dimensional loading constraints. *Procedia Computer Science*, 51:2257–2266, 2015.

[70] Jiyuan Pei, Chengpeng Hu, Jialin Liu, Yi Mei, and Xin Yao. Bi-objective splitting delivery VRP with loading constraints and restricted access. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 01–09. IEEE, 2021.

[71] Jiyuan Pei, Yi Mei, Jialin Liu, and Xin Yao. An investigation of adaptive operator selection in solving complex vehicle routing problem. In Sankalp Khanna, Jian Cao, Quan Bai, and Guandong Xu, editors, *PRICAI 2022: Trends in Artificial Intelligence*, pages 562–573, Cham, 2022. Springer Nature Switzerland.

[72] Hanne Pollaris, Kris Braekers, An Caris, Gerrit K Janssens, and Sabine Limbourg. Vehicle routing problems with loading constraints: state-of-the-art and future directions. *OR Spectrum*, 37:297–330, 2015.

[73] Jean-Yves Potvin and Samy Bengio. The vehicle routing problem with time windows part ii: genetic search. *INFORMS journal on Computing*, 8(2):165–172, 1996.

[74] Christian Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12):1985–2002, 2004.

[75] Christian Prins, Philippe Lacomme, and Caroline Prodhon. Order-first split-second methods for vehicle routing problems: A review. *Transportation Research Part C: Emerging Technologies*, 40:179–200, 2014.

[76] Maryam Rajaei, Ghasem Moslehi, and Mohammad Reisi-Nafchi. The split heterogeneous vehicle routing problem with three-dimensional loading constraints on a large scale. *European Journal of Operational Research*, 299(2):706–721, 2022.

[77] Sebastian Reil, Andreas Bortfeldt, and Lars Mönch. Heuristics for vehicle routing problems with backhauls, time windows, and 3D loading constraints. *European Journal of Operational Research*, 266(3):877–894, 2018.

[78] Jidong Ren, Yajie Tian, and Tetsuo Sawaragi. A relaxation method for the three-dimensional loading capacitated vehicle routing problem. In *2011 IEEE/SICE International Symposium on System Integration (SII)*, pages 750–755. IEEE, 2011.

[79] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472, 2006.

[80] Gan Ruan, Leandro L Minku, Stefan Menzel, Bernhard Sendhoff, and Xin Yao. When and how to transfer knowledge in dynamic multi-objective optimization. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 2034–2041. IEEE, 2019.

[81] Gan Ruan, Leandro L Minku, Stefan Menzel, Bernhard Sendhoff, and Xin Yao. Computational study on effectiveness of knowledge transfer in dynamic multi-objective optimization. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2020.

[82] Gan Ruan, Leandro L Minku, Stefan Menzel, Bernhard Sendhoff, and Xin Yao. Knowledge transfer for dynamic multi-objective optimization with a changing number of objectives. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2024.

[83] Gan Ruan, Leandro L. Minku, Stefan Menzel, Bernhard Sendhoff, and Xin Yao. Learning to expand/contract pareto sets in dynamic multi-objective optimization with a changing number of objectives. *IEEE Transactions on Evolutionary Computation*, pages 1–1, 2024.

[84] Qingfang Ruan, Zhengqian Zhang, Lixin Miao, and Haitao Shen. A hybrid approach for the vehicle routing problem with three-dimensional loading constraints. *Computers & Operations Research*, 40(6):1579–1589, 2013.

[85] Hendrik Schaap, Maximilian Schiffer, Michael Schneider, and Grit Walther. A large neighborhood search for the vehicle routing problem with multiple time windows. *Transportation Science*, 56(5):1369–1392, 2022.

[86] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 417–431. Springer, 1998.

[87] Kay Chen Tan, Liang Feng, and Min Jiang. Evolutionary transfer optimization-a new frontier in evolutionary computation research. *IEEE Computational Intelligence Magazine*, 16(1):22–33, 2021.

[88] Kay Chen Tan, Loo Hay Lee, QL Zhu, and Ke Ou. Heuristic methods for vehicle routing problem with time windows. *Artificial Intelligence in Engineering*, 15(3):281–295, 2001.

[89] Ke Tang, Yi Mei, and Xin Yao. Memetic algorithm with extended neighborhood search for capacitated arc routing problems. *IEEE Transactions on Evolutionary Computation*, 13(5):1151–1166, 2009.

[90] Ke Tang and Xin Yao. Learn to optimize—a brief overview. *National Science Review*, page nwae132, 04 2024.

[91] Yi Tao and Fan Wang. An effective tabu search approach with improved loading algorithms for the 3l-cvrp. *Computers & Operations Research*, 55:127–140, 2015.

[92] Christos D Tarantilis, Emmanouil E Zachariadis, and Chris T Kiranoudis. A hybrid metaheuristic algorithm for the integrated vehicle routing and three-dimensional container-loading problem. *IEEE Transactions on Intelligent Transportation Systems*, 10(2):255–271, 2009.

[93] Chuan-Kang Ting, Chien-Hao Su, and Chung-Nan Lee. Multi-parent extension of partially mapped crossover for combinatorial optimization problems. *Expert Systems with Applications*, 37(3):1879–1886, 2010.

[94] Renato Tinós, Keld Helsgaun, and Darrell Whitley. Efficient recombination in the Lin-Kernighan-Helsgaun traveling salesman heuristic. In Anne Auger, Carlos M. Fonseca, Nuno Lourenço, Penousal Machado, Luís Paquete, and Darrell Whitley, editors, *Parallel Problem Solving from Nature – PPSN XV*, pages 95–107, Cham, 2018. Springer International Publishing.

[95] Hao Tong, Leandro Minku, Stefan Menzel, Bernhard Sendhoff, and Xin Yao. A novel generalized metaheuristic framework for dynamic capacitated arc routing problems. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, GECCO '23 Companion, page 45–46, New York, NY, USA, 2023. Association for Computing Machinery.

[96] Hao Tong, Leandro L. Minku, Stefan Menzel, Bernhard Sendhoff, and Xin Yao. Benchmarking dynamic capacitated arc routing algorithms using real-world traffic simulation. In *2022 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2022.

[97] Paolo Toth and Daniele Vigo. *The Vehicle Routing Problem*. SIAM, 2002.

[98] Paolo Toth and Daniele Vigo. *Vehicle Routing: Problems, Methods, and Applications*. SIAM, 2014.

[99] Shigeyoshi Tsutsui and Ashish Ghosh. A study on the effect of multi-parent recombination in real coded genetic algorithms. In *1998 IEEE international conference on evolutionary computation proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360)*, pages 828–833. IEEE, 1998.

[100] Ayad Turky, Irene Moser, and Aldeida Aleti. An iterated local search with guided perturbation for the heterogeneous fleet vehicle routing problem with time windows and three-dimensional loading constraints. In *Australasian Conference on Artificial Life and Computational Intelligence*, pages 279–290. Springer, 2017.

[101] Gündüz Ulusoy. The fleet size and mix problem for capacitated arc routing. *European Journal of Operational Research*, 22(3):329–337, 1985.

[102] Lei Wang, Songshan Guo, Shi Chen, Wenbin Zhu, and Andrew Lim. Two natural heuristics for 3D packing with practical loading constraints. In *Pacific Rim International Conference on Artificial Intelligence*, pages 256–267. Springer, 2010.

[103] Lijun Wei, Zhenzhen Zhang, and Andrew Lim. An adaptive variable neighborhood search for a heterogeneous fleet vehicle routing problem with three-dimensional loading constraints. *IEEE Computational Intelligence Magazine*, 9(4):18–30, 2014.

[104] Xin Yao. Simulated annealing with extended neighbourhood. *International journal of computer mathematics*, 40(3-4):169–189, 1991.

[105] Xin Yao. Dynamic neighbourhood size in simulated annealing. In *Proc. of Int'l Joint Conf. on Neural Networks (IJCNN'92)*, volume 1, pages 411–416, 1992.

[106] Xin Yao. An overview of evolutionary computation. *Chinese Journal of Advanced Software Research*, 3:12–29, 1996.

[107] Junmin Yi and Andreas Bortfeldt. The capacitated vehicle routing problem with three-dimensional loading constraints and split delivery—a case study. In Andreas Fink, Armin Fügenschuh, and Martin Josef Geiger, editors, *Operations Research Proceedings 2016*, pages 351–356, Cham, 2018. Springer International Publishing.

[108] Yusuf Yilmaz and Can B Kalayci. Variable neighborhood search algorithms to solve the electric vehicle routing problem with simultaneous pickup and delivery. *Mathematics*, 10(17):3108, 2022.

[109] Emmanouil E Zachariadis, Christos D Tarantilis, and Chris T Kiranoudis. The pallet-packing vehicle routing problem. *Transportation Science*, 46(3):341–358, 2012.

[110] Emmanouil E Zachariadis, Christos D Tarantilis, and Chris T Kiranoudis. Designing vehicle routes for a mix of different request types, under time windows and loading constraints. *European Journal of Operational Research*, 229(2):303–317, 2013.

[111] Han Zhang, Qing Li, and Xin Yao. An Adaptive Interactive Routing-Packing Strategy for Split Delivery Vehicle Routing Problem with 3D Constraints: Supplementary Material. Zenodo. June 2024. https://doi.org/10.5281/zenodo.12049152.

[112] Han Zhang, Qing Li, and Xin Yao. Dataset of PEAC-HNF algorithm. Zenodo. May 2024. https://doi.org/10.5281/zenodo.11232377.

[113] Han Zhang, Qing Li, and Xin Yao. Detailed experimental results of PEAC-HNF algorithm. Zenodo. May 2024. https://doi.org/10.5281/zenodo.11231220.

[114] Han Zhang, Qing Li, and Xin Yao. Knowledge-Guided Optimization for Complex Vehicle Routing with 3D Loading Constraints: Supplementary Material. Zenodo. April 2024. https://doi.org/10.5281/zenodo.10988571.

[115] Han Zhang, Qing Li, and Xin Yao. An adaptive interactive routing-packing strategy for split delivery vehicle routing problem with 3d loading constraints. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '24, New York, NY, USA, 2024. Association for Computing Machinery. https://doi.org/10.1145/3638529.3653991. In press.

[116] Han Zhang, Qing Li, and Xin Yao. An efficient algorithm for split delivery vehicle routing problem with three-dimensional loading. *Memetic Computing*, 2024. Submitted and under review.

[117] Han Zhang, Qing Li, and Xin Yao. Knowledge-guided optimization for complex vehicle routing with 3d loading constraints. In *International Conference on Parallel Problem Solving from Nature – PPSN XVIII*. Springer, 2024. In press.

[118] Han Zhang, Qing Li, and Xin Yao. PEAC-HNF:a novel multi-objective evolutionary algorithm for split delivery vehicle routing problem with three-dimensional loading constraints. *IEEE Transaction on Emerging Topics on Computational Intelligence*, 2024. Submitted and under review.

[119] Han Zhang, Qing Li, and Xin Yao. Peach: A multi-objective evolutionary algorithm for complex vehicle routing with three-dimensional loading constraints. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '24 Companion, New York, NY, USA, 2024. Association for Computing Machinery. https://doi.org/10.1145/3638530.3654333. In press.

[120] Zhenzhen Zhang, Lijun Wei, and Andrew Lim. An evolutionary local search for the capacitated vehicle routing problem minimizing fuel consumption under three-dimensional loading constraints. *Transportation Research Part B: Methodological*, 82:20–35, 2015.

[121] Wenbin Zhu, Hu Qin, Andrew Lim, and Lei Wang. A two-stage tabu search algorithm with enhanced packing heuristics for the 3L-CVRP and M3L-CVRP. *Computers & Operations Research*, 39(9):2178–2195, 2012.

[122] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, 3(4):257–271, 1999.