COLLEGE ED MATHEMATIQUES
DOCTORAL STIC
PAYS DE LA LOIRE

université angers

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ D'ANGERS

ÉCOLE DOCTORALE N° 641
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par
Zhenyu LEI

Memetic Algorithms and GPU Acceleration for Multi-Attribute Routing Problems

Thèse présentée et soutenue à Angers, le 19 Mai 2026
Unité de recherche : Laboratoire d'Étude et de Recherche en Informatique d'Angers (LERIA)
Thèse N° : « si pertinent »

Rapporteurs avant soutenance :

M. Jean-Charles BILLAUT	Professeur à Université de Tours
M. Marc SEVAUX	Professeur à Université de Bretagne-Sud

Composition du Jury :

Président :	M. David LESAINT	Professeur à Université d'Angers
Examinateurs :	M. Jean-Charles BILLAUT	Professeur à Université de Tours
	M. Jin-Kao HAO	Professeur à Université d'Angers
	M. David LESAINT	Professeur à Université d'Angers
	M. Marc SEVAUX	Professeur à Université de Bretagne-Sud
	Mme. Yang WANG	Professeur à Northwestern Polytechnical University
Directeur de thèse :	M. Jin-Kao HAO	Professeur à Université d'Angers

Acknowledgement

Time has passed quickly, and I have now completed nearly four years of academic life at Université d'Angers. These years have been invaluable for my academic growth, personal development, and exploration of challenging research problems. The experiences, knowledge, and friendships I have gained here will continue to guide and inspire me in my future endeavors.

First and foremost, I would like to express my deepest gratitude to my supervisor, Professor Jin-Kao Hao, for his guidance, continuous support, and insightful advice throughout my PhD journey. He has always been patient and approachable during our discussions on academic topics, offering valuable insights and guidance with his extensive knowledge and expert perspective. His mentorship has greatly shaped this research work and my development as a researcher, providing me with the skills and confidence to pursue future challenges in the field.

I would also like to sincerely thank Professor David Lesaint for serving as the president of my thesis jury. I am equally grateful to Professor Jean-Charles Billaut and Professor Marc Sevaux for reviewing my thesis and providing constructive feedback and thoughtful suggestions, which have significantly improved the quality of this work. I also wish to thank Professor Yang Wang for serving both as a member of my thesis jury and my thesis committee. My sincere appreciation also goes to Professor Qinghua Wu, a member of my thesis committee, for his valuable support during my studies.

I am also grateful to my colleagues at LERIA for fostering a stimulating and collaborative research environment. Special thanks go to the laboratory's technical and administrative staff for their continuous support and assistance, which have made my research work smoother and more productive.

Finally, I am profoundly thankful to my family, especially my parents, for their unwavering love, encouragement, and support throughout my academic journey. I would also like to thank all my friends for their companionship, encouragement, and help during my PhD studies. Their presence has made this journey both enjoyable and memorable.

This research has been financially supported by the China Scholarship Council (CSC, Grant No. 202206330014).

Table of Contents

General Introduction 9

1 Introduction 15

1.1 Routing problems . . . 16

1.1.1 Traveling salesman problem . . . 16

1.1.2 Vehicle routing problem . . . 17

1.1.3 Multi-attribute vehicle routing problem . . . 18

1.2 Vehicle routing problem with simultaneous pickup and delivery and time windows . . . 19

1.2.1 Problem introduction . . . 19

1.2.2 Related works . . . 21

1.2.3 Benchmark instances . . . 23

1.3 Multi-depot vehicle routing problems . . . 24

1.3.1 Problem introduction . . . 24

1.3.2 Related works . . . 25

1.3.3 Benchmark instances . . . 28

1.4 Chapter conclusion . . . 28

2 A review of GPU-based metaheuristics for routing problems 31

2.1 Introduction . . . 32

2.2 Background . . . 33

2.2.1 GPU and parallel computing . . . 33

2.2.2 Parallel performance metrics . . . 36

2.3 Metaheuristics for routing problems and parallel strategies . . . 37

2.3.1 Single-solution-based metaheuristics . . . 37

2.3.2 Population-based metaheuristics . . . 41

2.3.3 Hybrid metaheuristics . . . 44

2.4 Review of GPU-based metaheuristics for routing problems . . . 45

2.4.1 GPU-based single-solution-based metaheuristics . . . 46

2.4.2 GPU-based population-based metaheuristics . . . 51
2.4.3 GPU-based hybrid metaheuristics . . . 54
2.5 Discussion . . . 55
2.6 Chapter conclusion . . . 58

3 Memetic algorithm for vehicle routing with simultaneous pickup and delivery and time windows 59
3.1 Introduction . . . 60
3.2 Memetic algorithm with feasible and infeasible route descent search for VRPSPDTW . . . 61
3.2.1 Preprocessing . . . 62
3.2.2 Initialization . . . 63
3.2.3 Feasible and infeasible route descent search . . . 63
3.2.4 Adaptive route-inheritance crossover . . . 69
3.2.5 Learning-based adaptive strategy for the ARIX crossover . . . 71
3.2.6 Population management . . . 73
3.2.7 Discussion . . . 74
3.3 Computational results . . . 74
3.3.1 Experimental protocol and parameters tuning . . . 75
3.3.2 Reference algorithms . . . 75
3.3.3 Computational results and comparisons . . . 76
3.4 Performance analysis . . . 79
3.4.1 Analysis on the feasible and infeasible route descent search . . . 79
3.4.2 Analysis on the adaptive route-inheritance crossover . . . 82
3.5 Chapter conclusion . . . 83

4 Speeding up local optimization in vehicle routing with tensor-based GPU acceleration 85
4.1 Introduction . . . 86
4.2 GPU and tensor computation . . . 87
4.2.1 GPU architecture . . . 87
4.2.2 CUDA programming . . . 88
4.2.3 Tensor computation . . . 88
4.3 Move evaluation and attribute matrix . . . 89
4.3.1 Sequence concatenation-based move evaluation . . . 89

4.3.2 Attribute matrices . . . 90
4.3.3 Calculations of concatenation operations . . . 91
4.4 Tensor-based GPU acceleration for local search operators . . . 93
4.4.1 Workflow of the TGA framework between CPU and GPU . . . 93
4.4.2 Attribute-based solution tensor representation . . . 94
4.4.3 Tensor-based implementation of local search operators . . . 95
4.4.4 Complexity analysis . . . 104
4.5 Computational results . . . 106
4.5.1 Algorithms for experimentation . . . 106
4.5.2 Benchmark instances . . . 109
4.5.3 Experimental protocol . . . 109
4.5.4 Computational results and comparisons . . . 110
4.5.5 Analysis of overall speedup ratio . . . 112
4.5.6 Acceleration of move evaluation across operators . . . 113
4.5.7 Time analysis of key steps in TGA . . . 115
4.5.8 Comparative analysis with algorithms in the literature . . . 116
4.6 Discussion . . . 118
4.7 Chapter conclusion . . . 119

5 A GPU-accelerated hybrid method for a class of multi-depot vehicle routing problems 121
5.1 Introduction . . . 122
5.2 Multi-depot feasible-and-infeasible hybrid algorithm . . . 123
5.2.1 Initialization . . . 124
5.2.2 Multi-depot-supported feasible-and-infeasible search . . . 125
5.2.3 Diversity-controlled route-exchange crossover . . . 131
5.2.4 Population management . . . 135
5.2.5 Discussion . . . 136
5.3 Computational results . . . 138
5.3.1 Experimental protocol and parameter tuning . . . 138
5.3.2 Reference algorithms . . . 139
5.3.3 Computational results and comparisons . . . 140
5.4 Performance analysis . . . 141
5.4.1 Analysis on diversity-controlled route-exchange crossover . . . 142

5.4.2 Analysis on multi-depot-supported feasible-and-infeasible search . . 144
5.4.3 Analysis of GPU parallelism benefits 145
5.5 Chapter conclusion 147

Conclusion **149**

Bibliography **153**

Appendix **169**
A Detailed computational results on VRPSPDTW 169
B Detailed computational results for TGA 171
C Detailed computational results on MDVRPs 172

List of Figures **183**

List of Tables **187**

List of Publications **189**

General Introduction

Context

Routing problems constitute a fundamental branch of combinatorial optimization problems. They encompass a number of classic problems such as the Traveling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP). The Multi-Attribute VRPs (MAVRPs) extend the classical VRP by incorporating additional objectives and diverse operational constraints, aiming to capture the complexity of real-world decision-making in logistics, transportation, and distribution systems. Through these extensions, MAVRPs provide a more comprehensive and realistic representation of practical routing challenges faced in modern supply chain and service operations.

In this thesis, we focus on several representative and complex multi-attribute variants of the Vehicle Routing Problem, including the Vehicle Routing Problem with Simultaneous Pickup and Delivery and Time Windows (VRPSPDTW) [AM02], and a family of Multi-Depot Vehicle Routing Problems (MDVRPs) [Til69], such as the classical MDVRP, the MDVRP with Time Windows (MDVRPTW), and the Multi-Depot Open VRP (MDOVRP).

The VRPSPDTW is a single-depot variant, yet it is particularly challenging due to its hierarchical objective structure and multiple complex constraints, such as variable vehicle loads and time windows. Moreover, the VRPSPDTW is of significant importance as it generalizes several classical VRP variants, such as the Capacitated VRP (CVRP) and the VRP with Time Windows (VRPTW). Meanwhile, the MDVRP family also introduces multiple constraints, including vehicle capacity, route duration, time windows, and depot capacity. The presence of multiple depots makes these problems especially relevant for modeling realistic logistics systems in practical applications.

Given their theoretical and practical importance, numerous approaches have been proposed to solve these NP-hard problems, ranging from exact methods to heuristics and metaheuristics. However, due to their high complexity, exact methods rapidly become computationally intractable as problem size increases. Even state-of-the-art heuristics and metaheuristics, which are highly effective in many cases, still leave room for improvement

in solution quality when addressing these highly intricate problems with multiple complex constraints. In addition, they face significant computational challenges, particularly when handling large-scale instances.

With the rapid advancement of hardware and parallel computing technologies, Graphics Processing Units (GPUs) have emerged as a powerful platform for accelerating the intensive computations required in combinatorial optimization. Their massive parallelism provides new opportunities for enhancing the efficiency of solution methods, particularly the computationally demanding local search procedures, which often constitute the core search component of many high-performing heuristics and metaheuristics. Although numerous studies have investigated GPU-based metaheuristics for routing problems, a comprehensive and systematic review of this rapidly evolving field remains lacking.

Motivated by these challenges and opportunities, this thesis aims to advance the state-of-the-art by developing advanced memetic algorithms, which are among the most efficient metaheuristic approaches for solving these multi-attribute VRP variants with complex constraints. This thesis also conducts a comprehensive review of GPU-based metaheuristics for routing problems and explores GPU-based acceleration techniques for local optimization to further enhance the computational efficiency and scalability of the proposed algorithms.

Objectives

This thesis focuses on developing effective memetic algorithms for solving multi-attribute vehicle routing problems with multiple complex constraints, along with efficient GPU-based acceleration framework to enhance computational performance, particularly in local optimization. The main objectives of this research are as follows:

- To conduct a comprehensive review of GPU-based metaheuristics for routing problems, summarizing key developments, identifying strengths and limitations, and discussing emerging trends and future research directions in this evolving field.
- To analyze the characteristics of vehicle routing problems with complex constraints and establish an appropriate memetic algorithm framework.
- To design advanced local search operators and efficient neighborhood evaluation methods to improve solution quality.
- To develop GPU-based acceleration framework that significantly enhance the efficiency of neighborhood evaluation in local search.

- To propose dedicated crossover operators capable of handling solution structures with complex constraints.
- To incorporate learning-based mechanisms into the algorithms to support decision-making and improve search performance.
- To evaluate the proposed algorithms on benchmark instances and compare their performance with state-of-the-art methods.
- To conduct in-depth analyses of key algorithmic components, providing insights into design choices and their impact on performance.

Contributions

The main contributions of this thesis are summarized as follows.

- We conducted a comprehensive review of GPU-based metaheuristics for routing problems, including both TSP and various VRP variants. Based on 48 publications on GPU-based metaheuristics for routing problems published since 2010, we categorize existing methods based on the underlying metaheuristic paradigms, analyze their parallelization strategies, evaluate their strengths and limitations, and discuss common challenges such as scalability, reproducibility, and benchmark availability. Furthermore, we identify promising directions for future research. By offering a comprehensive review centered on GPU acceleration for routing problems, this work aims to serve as a useful reference for researchers and practitioners interested in high-performance optimization and the next generation of metaheuristic solvers. This work forms the basis of a review paper currently under preparation.
- For the single-depot VRPSPDTW with hierarchical objectives and multiple complex constraints, we proposed a novel memetic algorithm, MA-FIRD, which integrates several innovative strategies: (1) a feasible-infeasible route descent search with penalty management and a constant-time move evaluation for multiple complex constraints, (2) a reinforcement-learning-enhanced multi-parent route-inheritance crossover, and (3) a max-min normalization based fitness-distance population management scheme. Extensive experiments on the WC [WC12] and JD [LTY21] benchmarks demonstrated the algorithm's superiority, attaining the best solutions for all benchmark instances, including 57 best-known solutions and 8 new best upper bounds on the WC set, and new best upper bounds for all 20 large real-world JD instances. These results confirm both its effectiveness and its practical applicability.

The innovative strategies embedded in MA-FIRD also establish a solid foundation for tackling other multi-attribute VRPs. This work has been published in *IEEE Transactions on Evolutionary Computation* [LH25].

- Building on the observation that MA-FIRD still faces computational challenges on large-scale instances, we reformulated its constant-time move evaluation into a standardized sequence concatenation-based evaluation framework using attribute representations, forming the basis of a tensor-based GPU acceleration (TGA) framework for local optimization in VRPs. The main contributions are as follows: (1) We develop the first fine-grained tensor-based GPU acceleration framework, together with two concrete implementations, which accelerate move evaluation by tensorizing local search operators based on an attribute-based solution tensor representation, thereby fully exploiting GPU parallelism. (2) TGA is highly extensible, supporting a wide range of VRP variants and enabling seamless integration into different local search solvers. (3) Unlike low-level CUDA-based approaches, TGA can be implemented using high-level tensor libraries such as PyTorch, substantially lowering the barrier to GPU acceleration for researchers in operations research. (4) Comprehensive theoretical and experimental analyses provide deeper insights into the performance characteristics and computational bottlenecks of the proposed framework, offering valuable guidance for future research. This work has been submitted to *Transportation Science*.
- We further extended our study on single-depot VRPs to the multi-depot setting and proposed a hybrid algorithm, MDFIHA, within memetic algorithm framework, for solving a class of MDVRPs, including the MDVRP, MDVRPTW, and MDOVRP. MDFIHA integrates the global exploration capability of population-based search with the strong exploitation ability of local search. Its main components include: (1) a multi-depot feasible-and-infeasible search framework guided by a multi-penalty evaluation function and enhanced by two depot-specific operators; (2) a diversity-controlled route-exchange crossover equipped with a learning mechanism that dynamically adjusts offspring diversity; and (3) a fitness-and-distance-based population management strategy designed to maintain both solution quality and population diversity. To further address the computational challenges posed by large-scale instances, we develop an enhanced variant, MDFIHA-ETGA, featuring two major improvements. First, it incorporates an efficient edge-based tensor GPU acceleration (ETGA) framework to speed up neighborhood evaluations in lo-

cal search by fully exploiting GPU parallelism. Second, it adopts a leader-follower multi-move update strategy that applies multiple moves within a single local search iteration, thereby better utilizing the parallel evaluation capability of GPUs and further improving search efficiency. Experiments on four benchmark sets (125 instances across three MDVRP variants) showed that proposed algorithms achieve competitive or superior performance, setting 45 new best-known upper bounds while significantly accelerating large-scale problem solving. This work has been submitted to *IEEE Transactions on Evolutionary Computation*.

Furthermore, at the beginning of my PhD research, I conducted another study on developing an effective memetic algorithm for the Close-Enough Traveling Salesman Problem (CETSP), a variant of the Traveling Salesman Problem. This work, although outside the scope of the present thesis, was published in *Applied Soft Computing* [LH24].

Organization

The thesis is organized in the following way.

- Chapter 1 introduces routing problems, covering the classic TSP, VRP, and multi-attribute VRPs, and formally defines the complex VRP variants studied in this thesis. These include the VRPSPDTW, along with its reductions to the classic CVRP and VRPTW, as well as a class of multi-depot VRPs, including the MDVRP, MDVRPTW, and MDOVRP. The chapter also discusses real-world applications and provides a comprehensive review of existing methods and benchmark instances for these problem classes.
- Chapter 2 provides a comprehensive review of 48 publications on GPU-based metaheuristics for routing problems, covering research progress since 2010. We summarize key developments, highlight the strengths and limitations of existing approaches, and discuss emerging trends and future research directions in this evolving field.
- Chapter 3 presents the memetic algorithm MA-FIRD for solving the Vehicle Routing Problem with Simultaneous Pickup and Delivery and Time Windows. Experimental results demonstrate its superiority over state-of-the-art methods, particularly on large-scale real-world instances. The chapter also includes an analysis of key algorithmic components.
- Chapter 4 introduces an innovative Tensor-based GPU Acceleration framework

for accelerating time-consuming neighborhood evaluation of local search operators commonly used in vehicle routing. Comparative experiments highlight its efficiency over conventional CPU implementations, and in-depth analyses provide valuable insights and identify critical bottlenecks.

- Chapter 5 proposes novel hybrid algorithms, MDFIHA and its enhanced version MDFIHA-ETGA, for solving a class of Multi-Depot Vehicle Routing Problems. Extensive experiments on benchmark instances across different MDVRP variants demonstrate its competitiveness against leading algorithms. Additional analyses are conducted to assess the impact of its key components.
- The last chapter concludes the thesis by summarizing the main contributions and discussing perspectives and promising directions for future research.

Chapter 1

Introduction

1.1 Routing problems

Combinatorial Optimization Problems (COPs) are a fundamental class of discrete optimization problems where the objective is to find the best solution from a finite set of feasible solutions defined by a set of constraints. Among COPs, routing problems constitute a significant and widely studied subcategory. These problems are critical in logistics and transportation planning, where the goal is to determine one or more efficient routes for vehicles to deliver goods or services to a set of locations. The optimization often involves minimizing total travel distance, time, or cost, while satisfying a variety of constraints such as vehicle capacity, delivery time windows, and service-level requirements. Due to their direct impact on operational efficiency and cost savings, routing problems have gained considerable attention in both academia and industry for decades.

1.1.1 Traveling salesman problem

The Traveling Salesman Problem (TSP) [DFJ54] is one of the most well-known and extensively studied routing problems with wide-ranging applications in logistics, manufacturing, robotics, circuit design, and genomics. It is also one of the most popular NP-hard combinatorial optimization problems. Given a set of n cities and a cost matrix $\{c_{ij}\}$ representing the distance or cost of travel between each pair of cities i and j, the TSP involves finding the shortest Hamiltonian cycle that visits each city exactly once and returns to the starting city. An illustration of the TSP is shown in Figure 1.1.

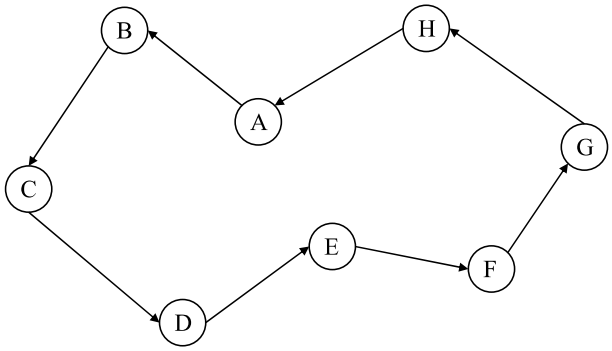


Figure 1.1 – Illustration of the Traveling Salesman Problem.

Due to its fundamental role in the field, the TSP has led to the development of numerous benchmark instances. The most widely used benchmark is TSPLIB [Rei91], introduced by Gerhard Reinelt in 1991 and hosted by the University of Heidelberg. TSPLIB contains

a large collection of real-world and synthetic instances, with sizes ranging from tens to tens of thousands of nodes, including the large-scale instances up to 85,900 nodes.

1.1.2 Vehicle routing problem

The Vehicle Routing Problem (VRP) [DR59; CW64] is also a foundational problem in combinatorial optimization that generalizes the Traveling Salesman Problem by considering multiple vehicles and capacity constraints. It arises in a wide range of practical contexts, particularly in transportation, logistics, and supply chain management, where efficient delivery and service routing are essential. Formally, let $V = \{v_0, v_1, \ldots, v_n\}$ be a set of nodes, where v_0 denotes a depot and $\{v_1, \ldots, v_n\}$ represent customers. Each customer v_i has a positive demand $d_i > 0$, and a cost matrix $\{c_{ij}\}$ specifies the travel cost or distance between nodes v_i and v_j. A fleet of M identical vehicles, each with a capacity Q, is available to serve the customers. As illustrated in Figure 1.2, the objective of the VRP is to determine a set of M routes that minimizes the total travel cost, subject to the following constraints: each route starts and ends at the depot v_0, each customer is visited exactly once, and the total demand served by each vehicle does not exceed its capacity Q.

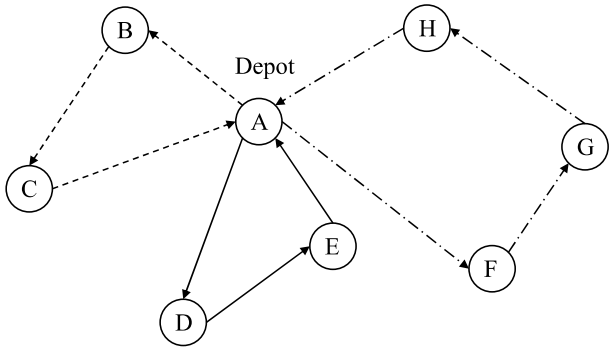


Figure 1.2 – Illustration of the Vehicle Routing Problem.

Since its introduction, the Vehicle Routing Problem and its numerous variants have been extensively investigated over the past decades, including classical forms such as the Capacitated VRP (CVRP), the VRP with Time Windows (VRPTW), and the Split-Delivery VRP (SDVRP), among many others. As research has evolved to better reflect real-world logistics and transportation systems, both the scale and the complexity of the studied problems have significantly increased.

For the VRP and its variants, standard benchmark instances are typically domain-specific. For the Capacitated Vehicle Routing Problem (CVRP), a widely used benchmark repository is CVRPLIB [1], which provides a collection of classical CVRP instances. Among them, the X instance set proposed by [Uch+17] is one of the most widely adopted. This modern benchmark suite combines both structured and randomly generated instances, with problem sizes scaling up to 1000 customers, thereby offering a comprehensive and challenging testbed for evaluating state-of-the-art algorithms.

CVRPLIB also includes two widely used benchmark sets for the VRP with Time Windows (VRPTW): The Solomon instances [Sol87], consisting of 56 problems with 100 customers each. And the Gehring and Homberger (GH) instances [GH99], comprising 300 instances with 200 to 1000 customers. Both sets include three categories of customer distribution: C-type (clustered), R-type (random), and RC-type (mixed), with variations in time window tightness and vehicle capacity.

These benchmark sets have been widely adopted in the literature, either directly or as the basis for generating new instance sets, to evaluate and compare the performance of algorithms for the VRP and its variants, providing a common ground for assessing solution quality and computational efficiency across different approaches.

1.1.3 Multi-attribute vehicle routing problem

Over time, the continuous introduction of new VRP variants has led to an increasingly fragmented research landscape, where each formulation targets a specific set of operational constraints or application scenarios. Although these variants share common structural features, they often differ in objectives, constraints, and modeling assumptions. This diversity highlights the need for a unified perspective capable of representing and analyzing routing problems with multiple interacting characteristics.

In this context, the Multi-Attribute Vehicle Routing Problem (MAVRP) has emerged as a generalized and unifying framework that integrates a wide range of VRP variants within a single modeling paradigm. The MAVRP explicitly accounts for multiple, potentially interdependent attributes, such as time windows, heterogeneous fleets, multiple depots, service types, or network structures, that jointly define the operational characteristics of a routing problem [Vid+14]. This unified view facilitates the analysis, comparison, and design of algorithms capable of handling the increasing heterogeneity of modern rout-

1. `http://vrp.galgos.inf.puc-rio.br/index.php/`

ing scenarios.

This thesis investigates several complex multi-attribute VRP variants. One of the most challenging problems considered is the Vehicle Routing Problem with Simultaneous Pickup and Delivery and Time Windows (VRPSPDTW). It is a single-depot problem with a hierarchical objective, where the primary goal is to minimize the number of vehicles, and the secondary goal is to minimize the total travel distance. It also integrates dynamically changing vehicle loads and strict time window constraints. Moreover, the VRPSPDTW is particularly significant as it generalizes several classical VRP variants, including the Capacitated VRP (CVRP) and the VRP with Time Windows (VRPTW).

Another class of problems studied is based on the classic Multi-Depot Vehicle Routing Problem (MDVRP), including its variants such as the Multi-Depot VRP with Time Windows (MDVRPTW) and the Multi-Depot Open VRP (MDOVRP). These variants combine multiple real-world features, including depot allocation, time windows, route duration limits, and open routing conditions.

Detailed introduction of these problems are provided in the following sections.

1.2 Vehicle routing problem with simultaneous pickup and delivery and time windows

1.2.1 Problem introduction

The Vehicle Routing Problem with Simultaneous Pickup and Delivery and Time Windows (VRPSPDTW) [AM02] is a member of the large family of Vehicle Routing Problems (VRPs) [BRV16]. Specifically, VRPSPDTW is a variant of the Vehicle Routing Problem with Simultaneous Pickup and Delivery (VRPSPD) [Min89] that incorporates time window constraints. This is a computationally challenging problem because it can be trivially reduced to the NP-hard VRPSPD problem [AM02]. VRPSPDTW has a wide range of applications in real world scenarios, especially in the field of logistics. Many logistics companies now offer pickup services in addition to delivery services to improve transportation efficiency and reduce costs, a concept commonly referred to as reverse logistics [Det01].

VRPSPDTW can be described on a complete graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the vertices or nodes $\mathcal{V} = \{v_0, v_1, \ldots, v_\mathcal{N}\}$ represent the depot v_0 and the $\mathcal{N}$ customers $\{v_1, \ldots, v_\mathcal{N}\}$. The edges $\mathcal{E} = \{e_{ij} | v_i, v_j \in \mathcal{V}\}$ represent the connections between these nodes. Each customer node $v_i \in \mathcal{V}$ $(i \neq 0)$ is associated with a delivery demand d_i and a pickup demand p_i.

This means that the vehicle must deliver d_i units of goods from the depot v_0 to v_i and pick up p_i units from v_i back to the depot. Furthermore, each node v_i has a time window $[e_i, l_i]$ and a service time s_i, which specifies the earliest and latest time for the service, as well as the time spent at node v_i for the service. The depot has a time window $[e_0, l_0]$, specifying the earliest departure and latest return time, with its service time s_0 set to 0. Additionally, each edge $e_{ij} \in \mathcal{E}$ is associated with a travel distance c_{ij} and a travel time t_{ij}.

A fleet of M homogeneous vehicles, each with a capacity $\mathcal{Q}$, serves all customers, starting and ending at the depot v_0, while satisfying various constraints, including capacity and time windows. A solution S is represented as a set of closed routes $S = \{R_1, \ldots, R_M\}$, where each route R_i consists of a sequence of nodes $\{n_{i,0}, n_{i,1}, \ldots, n_{i,\mathcal{L}_i}, n_{i,\mathcal{L}_i+1}\}$ visited by the i-th vehicle. Here, $n_{i,j}$ denotes the j-th node in route R_i and $\mathcal{L}_i$ is the number of served customers in that route. Thus, the length of route R_i is $|R_i| = \mathcal{L}_i + 2$. Note that both the first and last nodes in route R_i are the depot v_0 (i.e., $n_{i,0} = v_0$ and $n_{i,\mathcal{L}_i+1} = v_0$). To ensure that the load of each vehicle does not exceed its capacity $\mathcal{Q}$, the condition $q_{n_{i,j}} < \mathcal{Q}$ must hold for all $n_{i,j} \in R_i$, where $q_{n_{i,j}}$ represents the load of the i-th vehicle after visiting node $n_{i,j}$. Additionally, let $a_{n_{i,j}}$ denote the arrival time at node $n_{i,j}$. Arriving before $e_{n_{i,j}}$ results in a waiting time of $w_{n_{i,j}} = \max\{e_{n_{i,j}} - a_{n_{i,j}}, 0\}$, while arriving after $l_{n_{i,j}}$ renders the route infeasible.

The VRPSPDTW model is highly extensible, as it encompasses multiple complex constraints and can be reduced to several classical VRP variants through appropriate parameter settings. For example, the CVRP and VRPTW can be obtained by setting the pickup demand $p_i = 0$ for all customers. Moreover, the CVRP does not incorporate time-related attributes such as travel time, service time, or time windows.

$$
\begin{aligned}
\text{Minimize} \quad & f(S) = \mu_1 \cdot M + \mu_2 \cdot D(S) \\
\text{Subject to} \quad & D(S) = \sum_{i=1}^{M} \sum_{j=0}^{\mathcal{L}} c_{n_{i,j} n_{i,j+1}} \\
& S = \{R_1, \ldots, R_M\} \\
& R_i = \{n_{i,0}, n_{i,1}, \ldots, n_{i,\mathcal{L}}, n_{i,\mathcal{L}+1}\}, \quad i = 1, \ldots, M
\end{aligned}
\tag{1.1}
$$

Let μ_1 and μ_2 be the costs or weights assigned to the vehicles and the travel distance, respectively. A typical objective function of VRPSPDTW is then to minimize the total

cost, which is the weighted sum of the number of vehicles and the travel distance, as shown in Equation (1.1). Another typical hierarchical objective function aims to minimize first the number of vehicles used (primary objective) and then the travel distance of the vehicles (secondary objective). This can be achieved by assigning a very high cost μ_1 to vehicles and a relatively low cost μ_2 to travel distances.

1.2.2 Related works

Table 1.1 – Summary of the algorithms for VRPSPDTW in the literature.

Literature	Year	Method	Objective
Angelelli and Mansini [AM02]	2002	Branch-and-Price	travel cost
Liang et al. [LZZ09]	2009	GA	total cost with penalty
Lai and Cao [LC10]	2010	DE	travel cost
Wang and Chen [WC12]	2012	co-GA	hierarchical objective
Kassem and Chen [KC13]	2013	Insertion heuristic, SA	travel cost
Wang et al. [Wan+13]	2013	SA	hierarchical objective
Wang et al. [Wan+15]	2015	parallel SA	hierarchical objective
Hof and Schneider [HS19]	2019	ALNS, Path-relinking	hierarchical objective
Shi et al. [Shi+20]	2020	VNS, TS	hierarchical objective
Tang et al. [Tan+21]	2021	Co-evolution of parameterized search	weighted total cost
Liu et al. [LTY21]	2021	MA	hierarchical objective
Wu and Gao [WG23]	2023	ACO	weighted total cost
Praxedes et al. [Pra+24]	2024	Branch-Cut-and-Price	travel cost

As a variant of the well-studied VRPSPD problem [Min89; KLT20], the first study of VRPSPDTW can be traced back to the work of Angelelli and Mansini [AM02]. Their pioneering study introduced a Branch-and-Price algorithm based on a set covering formulation to address the problem, with the objective of minimizing the total travel cost of the vehicles. Table 1.1 summarizes the most important solution approaches that have been carried out in the literature on VRPSPDTW since then.

Regarding the objective of minimizing the total travel cost only, Lai and Cao [LC10] presented an improved differential evolution algorithm (IDE), where they devised a novel decimal encoding method to represent solutions and introduced a penalty function to deal with illegal solutions. Kassem and Chen [KC13] presented the INST-SA heuristic, using an insertion heuristic for solution initialization, and simulated annealing and neighborhood search for solution improvement. Tang et al. [Tan+21] studied VRPSPDTW with the objective of minimizing the weighted total cost of the vehicle dispatching and the travel distance. They proposed a co-evolution of parameterized search (CEPS) to achieve generalizable parallel algorithm portfolio (PAP) based on some training instances. Liang et

al. [LZZ09] studied a variant of VRPSPDTW with soft time window constraints, allowing delayed arrivals beyond the time windows with associated penalty costs.

Until the work of Wang and Chen [WC12], there was a lack of benchmark instances for VRPSPDTW, making it difficult to compare the solution algorithms for the problem. Wang and Chen filled the gap by introducing a set of 65 benchmark instances (denoted as WC) derived from the Solomon VRPTW benchmark [Sol87], previously introduced in Section 1.1.2. They also proposed a co-evolution genetic algorithm (co-GA), which maintains two populations for diversification and intensification, to minimize the hierarchical objective of the number of vehicles first and the travel distance second. Following their study on the hierarchical objective, a succession of research efforts using metaheuristics [Wan+13; Wan+15; HS19; Shi+20; LTY21] have contributed to advancing the state of the art in solving VRPSPDTW, particularly in the context of the WC benchmark.

Wang et al. [Wan+13] introduced a simulated annealing approach and later developed a parallel simulated annealing (p-SA) algorithm [Wan+15], utilizing multi-processor or multi-thread capabilities to enhance and accelerate the search process. Shi et al. [Shi+20] presented a two-stage algorithm VNS-BSTS. In the first stage, a Variable Neighborhood Search (VNS) was used to minimize the number of vehicles, featuring a novel learning-based evaluation function to assess each move. The second stage employed a Bi-Structure based Tabu Search (BSTS) to intensify the optimization. Recently, Wu and Gao [WG23] proposed an ant-colony optimization algorithm with destroy and repair strategies (ACO-DR) for VRPSPDTW, focusing on minimizing the weighted total cost of vehicle dispatching and travel distance.

Hof and Schneider [HS19] proposed an Adaptive Large Neighborhood Search with Path Relinking (ALNS-PR) algorithm for a class of VRPSPD, including VRPSPDTW. The algorithm involves 7 removal operators and 4 insertion operators, dynamically selected through an adaptive mechanism to iteratively destroy and repair the solution. Moreover, it incorporates a path-relinking component to explore promising search spaces between elite solutions. The algorithm temporarily accepts infeasible solutions by incorporating penalty terms into the objective function. Results showed the robustness of the ALNS-PR algorithm in solving this group of problems with competitive results on VRPSPDTW.

Liu et al. [LTY21] proposed the Memetic Algorithm with Extended Neighborhoods (MATE). This innovative approach employs local search for small-step exploration and removal-reinsertion of large neighborhoods for large-step exploration, effectively navigating the search space. To generate promising solutions, MATE incorporates a RARI

crossover combined with regret insertion. Experiments on the WC instances showed the efficacy of the MATE algorithm by discovering 12 new best solutions. The authors also generated a new set of 20 large-scale benchmark instances, denoted as JD benchmark, derived from the real-world JD logistics network. Testing on the JD instances confirmed MATE's effectiveness in solving VRPSPDTW in real-world scenarios.

Finally, Praxedes et al. [Pra+24] introduced the exact Branch Cut and Price (BCP) algorithm to solve a broad class of VRPSPD, including VRPSPDTW. With a 12-hour time limit, BCP achieved 45 optimal solutions out of 65 WC instances for minimizing travel cost only.

Given their superior performance on popular benchmarks compared to other approaches, ALNS-PR [HS19] and MATE [LTY21] are considered state-of-the-art methods for solving VRPSPDTW.

1.2.3 Benchmark instances

Two benchmark sets are available for the VRPSPDTW in the literature and are also considered in our study.

The first benchmark set consists of the popular 65 WC instances [2] from [WC12], which was derived from Solomon instances [Sol87] (Section 1.1.2). Among these instances, 9 are of small size with 10 to 50 customers, while the remaining 56 are of medium size with 100 customers. This benchmark is divided into three types: C instances with clustered customers, R instances with customer locations generated uniformly and randomly on a square, and RC instances with a combination of randomly placed and clustered customers. These instances are further classified to two categories: instances with narrow time windows and small vehicle capacity, and instances with large time windows and large vehicle capacity. Following [WC12; Wan+13; Wan+15; HS19; Shi+20; LTY21], we adopt the hierarchical objective for the WC benchmark with the number of vehicles and travel distance serving as the primary and secondary objectives, respectively. The instances of the WC benchmark have been extensively tested in the literature [WC12; Wan+13; Wan+15; HS19; Shi+20; LTY21; WG23; Pra+24]

The second benchmark set consists of the 20 large JD instances [3] from [LTY21], which was derived from real distribution system of JD Logistics company. This benchmark contains 4 groups of 5 instances with 200, 400, 600, 800, and 1000 customers. Unlike the

2. `https://oz.nthu.edu.tw/~d933810/test.htm`
3. `https://github.com/senshineL/VRPenstein`

WC benchmark designed for the hierarchical objective (first the number of vehicles and then the travel distance), the JD benchmark was initially designed to minimize the total cost, which is the weighted sum of the number of vehicles and the travel distance with predefined weights. This benchmark allows us to verify the usefulness of the proposed algorithm in real-world applications. Since this benchmark set is very recent, it has only been tested in [LTY21].

1.3 Multi-depot vehicle routing problems

1.3.1 Problem introduction

The Multi-Depot Vehicle Routing Problem (MDVRP) [Til69] is a well-known combinatorial optimization problem that extends the classical Vehicle Routing Problem (VRP) [DR59; CW64] by considering multiple depots from which vehicles are dispatched. The MDVRP is computationally challenging because it can be reduced to the classical single-depot VRP, which is itself NP-hard [LR78].

The MDVRP has many applications in transportation, logistics, and supply chain management. Over the years, various problem variants have been developed to address different operational constraints and objectives. In this thesis, we focus primarily on the classical MDVRP and the Multi-Depot Vehicle Routing Problem with Time Windows (MDVRPTW), which introduces time window constraints and is one of the most widely studied variants. We also consider the Multi-Depot Open Vehicle Routing Problem (MDOVRP) in our computational experiments. In this variant, vehicles are not required to return to their depots after serving customers.

The MDVRP and its variants can be described on a complete graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the set of nodes $\mathcal{V}$ consists of N_D depot nodes $\mathcal{V}_D$ and N_C customer nodes $\mathcal{V}_C$. The edges $\mathcal{E}$ represent all connections between nodes. Each customer node $v_i \in \mathcal{V}_C$ is associated with a delivery demand q_i that is served by an assigned vehicle, requiring a service time s_i. In the MDVRPTW, each node v_i has a time window $[e_i, l_i]$ denoting the earliest and latest time for the service. For depot nodes, the time windows define the earliest departure and latest return times, but their service times are set to zero. Each edge $e_{ij} \in \mathcal{E}$ is associated with a travel distance c_{ij} and a travel time t_{ij}, both of which are assumed to be symmetric and satisfy the triangle inequality. Each depot in $\mathcal{V}_D$ operates a fleet of up to N_V homogeneous vehicles, each with a limited capacity $\mathcal{Q}$. The objective of MDVRPs is

to determine a set of vehicle routes that collectively serve all customers while minimizing the total cost, typically defined as the total distance traveled by all vehicles, subject to various problem-specific constraints, as shown in Equation (1.2).

$$\text{Minimize} \quad f(S) = \sum_{i=1}^{M} \sum_{j=0}^{L_i} c_{n_{i,j} n_{i,j+1}} \tag{1.2}$$

In this formulation, the solution S is represented as a set of routes $S = \{R_1, \ldots, R_M\}$, where each route R_i is a sequence of nodes $\{n_{i,0}, n_{i,1}, \ldots, n_{i,L_i}, n_{i,L_i+1}\}$ visited by the i-th vehicle, with length $|R_i| = L_i + 2$. Here, $n_{i,0}$ and n_{i,L_i+1} denote the departure and arrival depot, and L_i is the number of customers served. A feasible solution requires that each route starts and ends at the same depot, i.e., $n_{i,0} = n_{i,L_i+1} \in \mathcal{V}_D$. Note that in the case of the MDOVRP, vehicles are not required to return to their depots after completing service, so the arrival depot node n_{i,L_i+1} can be neglected. Additional feasibility conditions include that the number of vehicles used per depot does not exceed the available fleet size N_V, the total load on each vehicle does not exceed capacity $\mathcal{Q}$, and each route respects a maximum tour duration $\mathcal{D}$, defined as the total time from departure to return. For the MDVRPTW, time window constraints must be satisfied. Let $a_{n_{i,j}}$ denote the arrival time at node $n_{i,j}$. If the vehicle arrives before the earliest time $e_{n_{i,j}}$, it must wait for $w_{n_{i,j}} = \max\{e_{n_{i,j}} - a_{n_{i,j}}, 0\}$. If it arrives after the latest time $l_{n_{i,j}}$, the route is considered infeasible.

1.3.2 Related works

The Vehicle Routing Problem is a fundamental combinatorial optimization problem that has been widely studied in the literature. It was first introduced by Dantzig and Ramser [DR59] as the Truck Dispatching Problem and then was formalized as a linear optimization problem by Clarke and Wright [CW64]. Of its many variants, the multi-depot variant (MDVRP) [Til69] stands out for its practical relevance, which better reflects real-world logistics. Because of its importance, extensive research has produced a wide range of solution methods, belonging to exact algorithms, heuristics, and metaheuristics, as surveyed in [Mon+15; SS20]. This section reviews key contributions to the MDVRP and its time-constrained variant, the MDVRPTW, which is another widely studied problem, due to its practical applicability in logistics.

Laporte [Lap84] proposed the first integer linear programming model for the MDVRP and developed a branch-and-bound exact algorithm. Other representative exact methods for the MDVRP were reported in [LNT88; BM09; CM14]. However, due to the problem's NP-hard nature, exact algorithms become impractical for large-scale instances. For instance, Contardo and Martinelli [CM14] reported that their exact method struggled with MDVRP instances with over 200 customers and could not solve those with over 300 customers in a reasonable amount of time. Consequently, heuristic and metaheuristic methods have become more common for solving the MDVRP.

The "cluster-first, route-second" heuristic strategy is widely used in MDVRP research. This approach first assigns customers to depots, and then solves the routing and scheduling problems for each depot. Giosa et al. [GTV02] focused on the assignment phase by integrating several heuristics with a modified Clarke and Wright heuristic to address the MDVRPTW. Tansini and Viera [TV06] introduced new proximity measures accounting for time windows and distances. More recently, Torres-Pérez et al. [Tor+22] proposed two new assignment heuristics showing competitive performance. Lim and Wang [LW05] compared two MDVRP approaches: a two-stage method based on the "cluster-first, route-second" strategy, and a one-stage method that integrates assignment and routing simultaneously. They concluded that the one-stage algorithm is more effective.

Among the heuristic and metaheuristic methods for solving the MDVRP, tabu search stands out as one of the most popular methods. Renaud et al. [RLB96] proposed a tabu search using the Improved Petal heuristic [RBL96] to generate initial solutions. Cordeau et al. [CGL97] later developed a unified tabu search heuristic applicable to multiple VRP variants, including the MDVRP. They evaluated their method using 33 benchmark instances and reported competitive results. This benchmark set remains the most widely used for the MDVRP.

Cordeau et al. [CLM01] extended their approach to include time window constraints and proposed a tabu search algorithm for VRP variants with time windows, including the MDVRPTW. They introduced 20 new benchmark instances for the MDVRPTW and showed that their algorithm produced competitive results. These 20 instances remain a standard benchmark for evaluating MDVRPTW algorithms. A later refinement with forward time slack strategy further improved performance [CLM04]. Finally, a parallel iterated tabu search [CM12] produced new best-known solutions with a simple and efficient implementation.

Another notable metaheuristic for MDVRPs is Variable Neighborhood Search. Polacek

et al. [Pol+04] proposed the first VNS-based algorithm for the MDVRPTW, achieving competitive performance against existing tabu search methods. Later, they incorporated parallel cooperative schemes [Pol+08], producing new best-known solutions. Sadati et al [SÇA21] presented a competitive VNS-based algorithm to solve a class of multi-depot vehicle routing problems, including the MDVRP, MDVRPTW, and MDOVRP. To enhance diversification and prevent premature convergence, the algorithm incorporates a tabu-based shaking mechanism during the diversification phase. The results on standard benchmark instances demonstrate the algorithm's effectiveness and competitiveness in solving the MDVRP and its multiple variants.

Adaptive Large Neighborhood Search has also proven to be another effective approach. In particular, Pisinger and Ropke [PR07] developed a generic ALNS algorithm to solve five VRP variants, including the MDVRP. ALNS dynamically selects among a set of insertion and removal operators based on their past performance, allowing the algorithm to adaptively explore the solution space. The algorithm was tested on various benchmark instances, demonstrating strong performance and proved competitive with state-of-the-art methods.

Hybrid genetic algorithms have also been widely applied in MDVRP research. Vidal et al. [Vid+12] proposed a Hybrid Genetic Search combining powerful neighborhood-based local search with effective population diversity control. This approach achieved new best-known solutions on benchmark instances of multiple VRP variants. Additionally, Vidal et al. [Vid+13] introduced an enhanced algorithm called Hybrid Genetic Search with Advanced Diversity Control for solving a broad class of time-constrained VRPs, including the MDVRPTW. Improved move evaluation strategies and robust diversity management contributed to its strong performance on existing benchmarks and on a set of new large-scale MDVRPTW instances proposed by the authors. Both studies [Vid+12; Vid+13] adopted the giant-tour approach [Pri04]. This approach first transforms a multiple routing solution into a giant-tour by concatenating all routes without delimiters, and then performs the simple order crossover, followed by an efficient *Split* algorithm to obtain the offspring routing solution.

Some studies have focused on improving algorithm efficiency through neighborhood reduction. [Esc+14] proposed a Hybrid Granular Tabu Search using five granular neighborhoods and multiple diversification strategies to balance solution quality and computational efficiency. Experiments on benchmark instances demonstrate that the proposed method is both competitive and effective compared to state-of-the-art algorithms. Another

notable study is by [Tu+14], who introduced a bi-level Voronoi diagram metaheuristic to efficiently reduce the search space for solving large-scale real-world MDVRPs (up to 20,000 customers), though it was less competitive on standard benchmarks.

1.3.3 Benchmark instances

In this thesis, we consider several benchmark sets for the classic Multi-Depot Vehicle Routing Problem (MDVRP) and two MDVRP variants: the Multi-Depot Vehicle Routing Problem with Time Windows (MDVRPTW) which introduces time window constraints, and the Multi-Depot Open Vehicle Routing Problem (MDOVRP), in which vehicles are not required to return to their depots after serving customers, which are widely used in the literature. These benchmark sets are summarized as follows:

C97: 33 widely studied MDVRP instances from [CGL97] with 48 to 360 customers and 2 to 9 depots. This set includes 23 instances from the literature and 10 *pr* instances generated with relaxed fleet-size limits. As some studies applied tighter limits, potentially causing result inconsistencies, we follow the original settings by default and also test the tighter version, denoted as *C97-T*.

C01: 20 MDVRPTW instances from [CLM01], based on the 10 *pr* instances from [CGL97] with added narrow or wide time windows. They include 48 to 288 customers and 4 to 6 depots. The last 10 instances were reported with relaxed fleet-size limits in [Vid+13], possibly due to an error. We account for this by also testing these 10 instances under relaxed constraints, denoted as *C01-R*.

V13: 28 large-scale MDVRPTW instances with 360 to 960 customers and 4 to 12 depots from [Vid+13]. To the best of our knowledge, these instances have only been tested in the original study [Vid+13].

L14: 24 MDOVRP instances from [LJG14], derived from *C97*, with 48 to 288 customers and 4 to 6 depots. Following common practice in the literature, fleet size limits N_V and tour duration constraints $\mathcal{D}$ are relaxed in these instances.

1.4 Chapter conclusion

In this chapter, we introduced the classical routing problems, including the Traveling Salesman Problem (TSP), the Vehicle Routing Problem (VRP), and its numerous variants, along with the concept of the Multi-Attribute VRP (MAVRP). We also presented the

specific VRP variants addressed in this thesis, reviewed relevant literature, and discussed commonly used benchmark instances for performance evaluation.

Chapter 2

A review of GPU-based metaheuristics for routing problems

Combinatorial optimization problems, particularly routing problems, are well known for their NP-hard nature and have attracted extensive research attention for decades. Metaheuristics have proven to be powerful algorithmic frameworks for addressing these complex problems. However, their computational cost remains a significant challenge. In recent years, the rise of General-Purpose computing on Graphics Processing Units (GPGPU) has made GPUs an attractive platform for accelerating such computations. As a major category of combinatorial optimization, routing problems, including the Traveling Salesman Problem (TSP), the Vehicle Routing Problem (VRP), and their numerous variants, have become a central focus in the advancement of GPU-accelerated algorithms. However, a comprehensive review of GPU-based metaheuristics specifically tailored for routing problems is lacking in the literature. To fill this gap, this chapter provides a systematic review of 48 publications on GPU-based metaheuristics for routing problems, covering research progress since 2010. This review summarizes key developments, highlights the strengths and limitations of existing approaches, and discusses emerging trends and future research directions in this evolving field. The content of this chapter serves as the foundation for a review paper currently under preparation.

2.1 Introduction

Most combinatorial optimization problems (COPs), including routing problems, are NP-hard, making the computation of optimal solutions intractable for large-scale instances. To address this, researchers have developed a wide range of heuristic and metaheuristic methods, which are capable of finding high-quality solutions within reasonable time limits. Metaheuristics such as Genetic Algorithms (GA), Ant Colony Optimization (ACO), Simulated Annealing (SA), and Tabu Search (TS) have proven especially effective for large, complex instances. However, these approaches still face challenges related to computational efficiency, especially when scaling to real-world, high-dimensional, or dynamic problem settings.

In recent years, the development of Graphics Processing Units (GPUs) and the advent of General-Purpose computing on Graphics Processing Units (GPGPU) have introduced new opportunities for accelerating metaheuristic algorithms. GPUs offer massive parallel processing capabilities with hundreds or thousands of cores, making them well-suited for the computational demands of metaheuristic search processes. As a result, an increasing number of studies have explored GPU-accelerated metaheuristics to enhance performance, scalability, and responsiveness, particularly for large-scale or real-time optimization tasks. Several survey studies have broadly reviewed GPU acceleration in combinatorial optimization and metaheuristics. Tan and Ding [TD15] focused on swarm intelligence algorithms, while Cheng and Gen [CG19] provided a selective overview of genetic algorithms. Essaid et al. [Ess+19] examined GPU-based metaheuristics in both combinatorial and continuous problem settings. Schryen [Sch20] offered a comprehensive review of parallel computational optimization in operations research, covering both exact methods and metaheuristics. However, these surveys either adopt a broad view of GPU acceleration in combinatorial optimization or focus on specific algorithmic families, leaving a lack of dedicated and up-to-date synthesis of GPU-based metaheuristics for routing problems.

To address this gap, this chapter provides a focused and up-to-date review of GPU-based metaheuristics for routing problems, including both TSP and various VRP variants. We categorize existing methods based on the underlying metaheuristic paradigms, analyze their parallelization strategies, evaluate their strengths and limitations, and discuss common challenges such as scalability, reproducibility, and benchmark availability. Furthermore, we identify promising directions for future research. By offering a comprehensive review centered on GPU acceleration for routing problems, this chapter aims to

serve as a useful reference for researchers and practitioners interested in high-performance optimization and the next generation of metaheuristic solvers.

The remainder of this chapter is organized as follows. Section 2.2 introduces the fundamentals of GPU computing and performance evaluation metrics. Section 2.3 provides a taxonomy of metaheuristics for routing problems and analyzes their potential for GPU parallelization. Section 2.4 offers a comprehensive review of developments since 2010, covering 48 representative studies on GPU-based metaheuristics for routing problems. Section 2.5 discusses the strengths and limitations of existing methods and outlines promising directions for future research. Finally, Section 2.6 summarizes the key findings and concludes the chapter.

2.2 Background

This section introduces the fundamentals of GPU computing, covering modern GPU architectures, parallel programming concepts, and performance evaluation metrics such as speedup and efficiency.

2.2.1 GPU and parallel computing

Graphics Processing Units (GPUs) have undergone a transformative evolution from their initial role in graphics rendering to becoming indispensable tools for high-performance computing (HPC). Over the past two decades, GPUs have emerged as powerful parallel processors well-suited for general-purpose computing, and have been widely adopted in fields such as machine learning, scientific simulations, data analytics, and combinatorial optimization.

GPU architecture

Unlike Central Processing Units (CPUs), which are optimized for low-latency execution of sequential instructions, Graphics Processing Units (GPUs) are designed to execute a vast number of lightweight threads in parallel. In NVIDIA's GPU architecture [1], thousands of physical cores, known as CUDA cores, are organized into units called Streaming Multiprocessors (SMs). Each SM contains a fixed number of CUDA cores. For instance,

1. https://www.nvidia.com/

the NVIDIA A100 GPU [2] (based on the Ampere architecture) features 108 SMs, each with 64 cores, totaling 6,912 CUDA cores. However, the number of physical cores does not directly equate to the number of threads a GPU can execute concurrently.

Threads in GPUs are logically grouped into units called *warps*, where 32 threads execute instructions in lockstep under the "Single Instruction, Multiple Threads" (SIMT) model. On the NVIDIA A100 GPU, each SM can manage up to 64 warps (i.e., 2,048 active threads), leading to a total of 221,184 simultaneously resident threads across the A100's 108 SMs. This ability to maintain a high number of concurrent threads is what allows GPUs to hide memory latency and sustain high throughput.

Moreover, parallel execution is structured hierarchically: individual threads are organized into thread blocks, which are further grouped into a grid. Each thread and block has a unique index, allowing developers to assign different data elements to different threads. This design enables efficient implementation of data-parallel operations, which are common in optimization problems.

In terms of memory access, NVIDIA GPU architecture includes a multi-level memory hierarchy to balance latency and accessibility. The key types of memory include:

- **Global memory**: A large, high-latency memory accessible by all threads on the device. Efficient access requires coalescing.
- **Shared memory**: Fast, low-latency memory shared among threads within the same block. Often used for cooperative computation and reducing global memory accesses.
- **Local memory and registers**: Private, very fast memory for each thread, used for temporary variables and intermediate computations.
- **Constant memory**: Read-only memory cached on-chip, optimized for broadcast access when all threads in a warp read the same location.
- **Texture memory**: Read-only memory cached on-chip, optimized for spatial locality of access.

This hierarchical design allows for high memory bandwidth and fine-grained control over data access, contributing to the GPU's ability to deliver substantial speedups over traditional CPU-based systems.

2. `https://www.nvidia.com/en-us/data-center/a100/`

GPU programming

Programming GPUs effectively presents several challenges due to their complex architecture. Highly efficient implementations often require careful thread synchronization, minimizing control flow divergence, efficient memory management, and other issues.

To address these challenges, NVIDIA introduced the CUDA (Compute Unified Device Architecture) programming model. CUDA allows developers to write GPU programs in familiar languages such as C/C++ or Python, while exposing parallelism through an abstraction of threads, blocks, and grids. It provides explicit control over memory access, synchronization, and thread scheduling, making it a powerful tool for high-performance GPU-accelerated applications.

In addition to low-level CUDA, high-level tensor-based frameworks such as PyTorch, TensorFlow, JAX, and CuPy have emerged. These frameworks abstract the details of GPU programming, mapping tensor (multi-dimensional array) operations to GPU kernels using CUDA internally. They greatly simplify development and enhance code portability, but they may not fully exploit the GPU's capabilities because they sacrifice fine-grained control over threads, memory hierarchy, and low-level hardware optimizations.

Additionally, profiling and performance analysis are crucial for optimizing GPU applications. For CUDA-based implementations, NVIDIA provides robust tools that allow detailed inspection of memory usage, kernel execution, instruction throughput, and latency. For tensor-based frameworks, built-in profilers offer insights into GPU memory consumption, kernel launch patterns, execution timelines, and operator-level performance, enabling effective optimization even in high-level implementations.

Parallelism paradigms

In parallel computing, two fundamental paradigms are commonly distinguished: *data parallelism* and *task parallelism*. *Data parallelism* involves applying the same operation simultaneously to multiple independent data elements, making it particularly well-suited for GPU architectures that feature thousands of lightweight cores and high memory bandwidth. On the other hand, *task parallelism* refers to executing different operations or tasks concurrently, potentially on different data, and typically requires more careful synchronization and scheduling.

The notions of *fine-grained* and *coarse-grained* parallelism offer an additional perspective on the scale and structure of parallel computation. *Fine-grained* parallelism refers

to the concurrent execution of numerous small, independent computations, and is well-suited to GPU thread-level execution. *Coarse-grained* parallelism, by contrast, involves the concurrent execution of larger, often independent tasks. In general, data parallelism tends to be fine-grained, while task parallelism is typically coarse-grained.

2.2.2 Parallel performance metrics

The computational efficiency of GPU-based algorithms is typically assessed using two categories of metrics: *speedup* and *scalability*.

The *speedup ratio* quantifies the performance gain of a GPU implementation relative to a reference baseline (usually a sequential CPU version) under identical stopping criteria:

$$\gamma(p) = \frac{T(n,1)}{T(n,p)} \tag{2.1}$$

where $T(n,1)$ is the runtime of the sequential implementation for problem size n, and $T(n,p)$ is the runtime of the parallel implementation using p processing units (e.g., GPU threads or cores).

$$E_s(p) = \frac{T(n,1)}{p \cdot T(n,p)} \tag{2.2}$$

Scalability reflects how efficiently performance improves as more processors are used. Two standard metrics are *strong scaling* and *weak scaling*.

$$E_w(p) = \frac{T(n,1)}{T(p \cdot n,p)} \tag{2.3}$$

Strong scaling (Equation (2.2)) evaluates runtime reduction for a fixed problem size as p increases. *Weak scaling* (Equation (2.3)) evaluates performance stability when both problem size and processors grow proportionally. In both cases, $E_s(p) = 1$ and $E_w(p) = 1$ represent ideal efficiency.

$$\text{Gap}(\%) = 100 \times \frac{f - f_{\text{ref}}}{f_{\text{ref}}} \tag{2.4}$$

Additionally, for evaluating solution quality, a standard metric is the relative gap as shown in Equation (2.4), which quantifies how close the obtained objective value (f) is to a reference value (f_{ref}), typically the known optimum or the best-known solution from

the literature. A negative gap indicates that the obtained solution improves upon the reference value.

2.3 Metaheuristics for routing problems and parallel strategies

To set the stage for GPU-based metaheuristics in routing problems, we first revisit the classical metaheuristics commonly applied to these problems and highlight their computational bottlenecks, which are potential opportunities for GPU acceleration.

Metaheuristics are powerful frameworks for addressing a wide range of combinatorial optimization problems (COPs), including routing problems [Lab+16]. They are generally classified into three categories [Tal09]: single-solution-based metaheuristics, which explore the search space through iterative local modifications of a single candidate solution; population-based metaheuristics, which maintain and evolve a set of solutions through evolutionary mechanism or swarm intelligence; hybrid metaheuristics, which integrate components or strategies from different paradigms to balance exploration and exploitation. Each class poses different computational demands and degrees of parallelism, leading to different opportunities and challenges when implemented on GPU architectures.

In general, the choice of parallelism paradigms and parallelization strategies depends on the characteristics of the metaheuristic. For instance, population-based algorithms naturally lend themselves to parallel execution, as the evaluation or evolution of multiple individuals can be performed concurrently. In contrast, single-solution-based methods often exploit parallelism within the neighborhood evaluation process to accelerate local search operations. Figure 2.1 illustrates the taxonomy, showing the classification of metaheuristics along with their potential opportunities for GPU-based parallelism. The detailed mechanisms and GPU adaptations of each metaheuristic class are discussed in the following sections.

2.3.1 Single-solution-based metaheuristics

Single-solution-based metaheuristics operate by maintaining and iteratively refining a single candidate solution throughout the search process. In contrast to population-based methods, which explore the solution space through the simultaneous evolution of multiple individuals, these algorithms rely on neighborhood structures and heuristic move

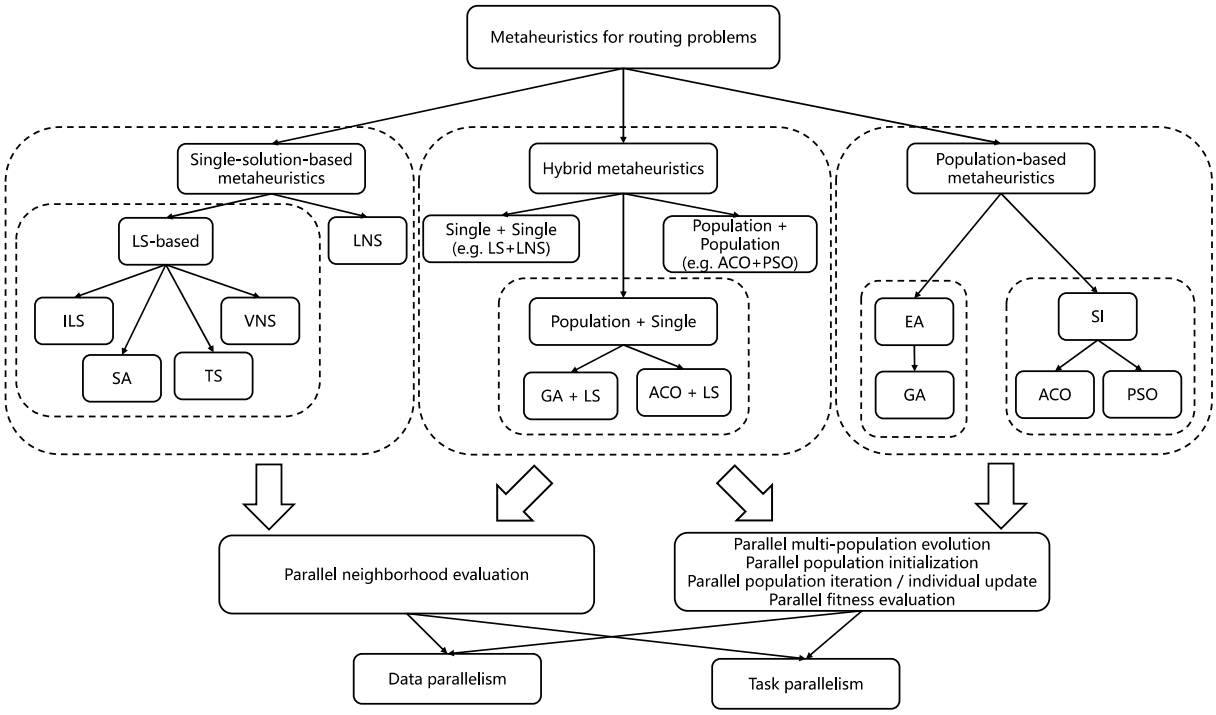


Figure 2.1 – Taxonomy of metaheuristics for routing problems and their corresponding GPU parallelism paradigms.

operators to guide the search. This class encompasses Local Search (LS) based methods such as Iterated Local Search (ILS), Simulated Annealing (SA), Tabu Search (TS), and Variable Neighborhood Search (VNS), as well as more advanced approaches like Large Neighborhood Search (LNS).

Local Search (LS) is a foundational metaheuristic widely applied to routing problems [FGI05] and has also emerged as an important research topic for GPU parallelization. Its key mechanism involves generating and evaluating neighboring solutions and moving to an improved solution according to predefined acceptance criteria. In routing problems, the neighborhood can be defined by a variety of move operators. For example, 2-opt, 3-opt, and Or-opt for TSP or intra-route improvements in VRPs, and Relocate, Swap, Cross-exchange, and 2-opt* for inter-route improvements in VRPs. These operators collectively provide a flexible framework for systematically exploring the solution space and improving solution quality in routing problems.

The simplest form of LS, often referred to as Hill Climbing (HC) in the literature, iteratively explores the neighborhood of a solution and accepts only improving moves. This basic principle forms the core of many LS-based metaheuristics, such as Iterated Local

Search (ILS), Simulated Annealing (SA), Tabu Search (TS), and Variable Neighborhood Search (VNS). These methods extend classical LS with additional strategies to overcome premature convergence to local optima: diversification in ILS, probabilistic acceptance of worse solutions in SA, memory-based restrictions in TS, and systematic changes of neighborhood structures in VNS. Despite their differences, they all share the same underlying nature: iterative improvement through local neighborhood exploration, which often represents a major computational bottleneck and a key factor influencing algorithmic efficiency. At the same time, this characteristic provides a natural opportunity to leverage GPU parallelism to accelerate computation.

Local Search typically employs two main search strategies: best-improvement and first-improvement. Among them, best-improvement is naturally more suitable for GPU parallelization. In this strategy, all candidate moves within the neighborhood are evaluated, and the move that provides the greatest improvement is selected. Since move evaluations are independent, each candidate move can be mapped to an individual GPU thread, enabling fine-grained data parallelism where thousands of moves are evaluated simultaneously. For instance, in the TSP, the cost variation of a 2-opt move replacing edges $(u, u+1)$ and $(v, v+1)$ with (u, v) and $(u+1, v+1)$ can be calculated using Equation 2.5, and this computation can be efficiently performed by a single GPU thread.

$$\Delta = c_{u,v} + c_{u+1,v+1} - c_{u,u+1} - c_{v,v+1} \tag{2.5}$$

However, there are also practical concerns. First, if a strict best move must be selected, a global reduction operation is required to identify it, which introduces synchronization overhead and can become a computational barrier, especially in very large neighborhoods. Second, move evaluation often requires reading edge distances from the cost matrix $c_{i,j}$. These memory accesses are not always consecutive, leading to non-coalesced (scatter) memory reads that reduce memory throughput and may limit parallel efficiency. Memory consumption also becomes a concern when handling very large neighborhoods.

Beyond these memory and synchronization issues, some operators are inherently difficult to parallelize due to their sequential or complex structure. For example, the SWAP* operator [Vid22] has high computational complexity, and Lin-Kernighan (LK) neighborhood [LK73; Hel00] moves involve sequential decision steps that are challenging to map efficiently to GPU threads. Furthermore, GPUs' SIMT architecture favors simple, uniform operations, therefore, sophisticated optimization strategies in CPU architectures such as

move pruning often cause control-flow divergence, reducing efficiency.

In contrast, first-improvement evaluates neighbors sequentially and stops at the first improving move, which makes it inherently less suitable for direct GPU parallelization. To better exploit GPU capabilities, several adaptations can be considered. One approach is asynchronous move application, where threads tentatively apply moves in shared or private memory and then commit the first valid improvement. Another is the batch first-improvement, in which a subset of neighbors is assigned to a GPU thread block and evaluated in parallel within a single kernel launch, thereby approximating the sequential nature of classical first-improvement while exploiting parallelism at the data level. A further alternative is a more coarse-grained multi-start first-improvement strategy, where multiple threads or thread blocks explore different solutions simultaneously, enhancing overall search diversification.

Simulated Annealing (SA) enables exploration by probabilistically accepting worse solutions according to a temperature parameter. Its inherently sequential nature, which resembles the first-improvement strategy, makes parallelization challenging. Nevertheless, many studies have investigated parallel variants of SA, particularly synchronous and asynchronous strategies [LL96; OÖ01]. Ferreiro et al. [Fer+13] provided implementations of such two SA approaches on GPU. In the asynchronous approach, each thread independently executes its own SA run, and the best solution is selected at the end. In the synchronous approach, after a fixed number of iterations, all threads synchronize with the current best solution and then continue the search from that shared state.

Additionally, SA variants incorporating an equilibrium loop are particularly amenable to GPU parallelization. The algorithm features two nested loops: an outer loop controlling the temperature schedule and an inner equilibrium loop that evaluates multiple candidate moves at a fixed temperature. This inner loop offers natural fine-grained parallelism, as the evaluation of different moves can be distributed across GPU threads.

Tabu Search (TS) typically employs the best-improvement strategy, selecting the best move while incorporating memory-based mechanisms to prevent cycling and promote diversification. Since the tabu mechanism is generally less time-consuming than move evaluation, a straightforward approach is to adopt a hybrid CPU-GPU framework: move evaluations are parallelized on the GPU, while tabu management remains on the CPU. In this case, TS shares the same opportunities and challenges as best-improvement LS, although tabu checks may still need to be implemented in parallel for efficiency. An early example is the work of Janiak et al. [JJL08], who implemented a GPU-based TS for

the TSP and Job Scheduling Problem (JSP), parallelizing neighborhood evaluation on the GPU while handling tabu operations on the CPU. On the other hand, fully GPU-based parallelism requires careful design of the tabu data structures and their storage. For instance, Malleypally [Mal18] proposed both global and block-level tabu lists, stored in global or shared memory, to exploit different levels of parallelism and memory access efficiency.

Furthermore, Iterated Local Search (ILS) and Variable Neighborhood Search (VNS) can employ either first-improvement or best-improvement strategies. Aside from the computation intensive neighborhood evaluation, other components, such as the perturbation step in ILS or the systematic neighborhood changes in VNS, are relatively lightweight, making them suitable for a hybrid CPU-GPU framework [ZHQ16]. As with Tabu Search, achieving fully GPU-based parallelism requires additional design efforts. For example, in ILS, efficient parallel perturbation strategies can be implemented to exploit GPU resources. In VNS, integrating multiple move operators opens up further possibilities for parallelism: one approach is to execute different operators sequentially, while another is to explore either the union of move operators or multiple instances of single-move operators in parallel. Ultimately, designing an effective parallel search strategy involves a trade-off between computational cost, parallel efficiency, and solution quality.

Finally, Large Neighborhood Search (LNS) is another effective metaheuristic for tackling routing problems [PR18]. Unlike LS-based methods, which rely on incremental improvements within a local neighborhood, LNS explores much larger neighborhoods by repeatedly applying destroy and repair operators to partially reconstruct solutions. Typical destroy operators for routing problems include random removal, related removal, and worst removal, while common repair operators include greedy insertion and regret-k insertion. The inherently sequential nature of many repair heuristics, where each insertion changes the solution context for the next, makes fine-grained GPU parallelization challenging. A practical alternative is coarse-grained task parallelism, where multiple independent LNS instances are executed in parallel [BHS19].

2.3.2 Population-based metaheuristics

Population-based metaheuristics are inspired by natural processes such as evolution and swarm behavior. Representative examples include Genetic Algorithms (GA) from Evolutionary Algorithms (EA) and Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) from Swarm Intelligence (SI). They maintain a population of can-

didate solutions that evolve or cooperate to improve quality over time. Due to the independence among individuals, these methods are inherently well suited for parallelization, making them a natural fit for GPU architectures.

Genetic Algorithms (GAs) are among the most widely used population-based metaheuristics, inspired by the principles of natural selection and genetic inheritance. A GA typically begins with a population of candidate solutions, each encoded as a chromosome, and iteratively evolves this population through three key operators: selection, crossover, and mutation. GAs have been successfully applied to a wide range of routing problems [Lar+99; KP15].

In terms of parallelization, GAs offer opportunities at multiple levels. First, fitness evaluation is typically the main bottleneck in routing problems, and substantial speedups can be achieved by distributing cost calculations across thousands of GPU threads. Second, the evolutionary operators, such as selection, crossover, and mutation, which are applied at each iteration can also be parallelized to improve efficiency. In GA encodings for routing problems, complete solutions are represented as *chromosomes*, while cities or customers correspond to *genes*. From this perspective, GPU parallelism can be exploited at the gene-level (data parallelism), where independent operations are performed on genes, and at the chromosome-level (solution-level), where operators are applied across multiple chromosomes concurrently.

Beyond these operator-level strategies, GAs can also be parallelized through population models originally studied on CPU architectures [GW93]: (1) the Master-Slave model, where the GPU accelerates fitness evaluation under a centralized GA controller; (2) the Island model, where multiple subpopulations evolve in parallel with occasional migration; and (3) the Cellular model, where individuals interact only with local neighbors on a grid. Each of these models requires tailored adaptations for GPU implementation, as summarized by Cheng and Gen [CG19].

Despite these opportunities, several challenges remain. Fitness evaluation involves non-coalesced memory accesses to distance matrices, crossover operators often require sequential steps, and population management must balance GPU utilization against convergence efficiency. Moreover, control flow divergence (e.g., handling infeasible VRP solutions) and synchronization overhead (e.g., migration in island models) can limit performance. Designing efficient GPU-based GAs thus requires careful consideration of trade-offs between algorithmic complexity, memory efficiency, and parallel scalability.

Ant Colony Optimization (ACO) is also well-established population-based metaheuris-

tic inspired by the foraging behavior of ants, where pheromone trails are used as a form of indirect communication to collectively discover high-quality paths. In ACO, a population of ants constructs solutions by traversing a problem graph (e.g., cities in the TSP), guided by pheromone levels and heuristic information. Once all ants complete their tours, pheromone trails are updated: good solutions reinforce their edges with higher pheromone deposits, while evaporation prevents premature convergence by weakening less promising paths. Depending on the specific strategies and implementation details, several ACO variants have been developed for routing problems, with representative examples being the Ant System (AS), the Ant Colony System (ACS) and the Max-Min Ant System (MMAS) [SD99].

Because of its decentralized nature and independence among ants, ACO is highly suitable for GPU parallelization and is one of the most studied metaheuristics in this context [PNC11; MLG22]. Since ants construct their solutions independently, a straightforward strategy is to map each ant to a GPU thread or block, enabling hundreds or thousands of tours to be built simultaneously. Within solution construction, probabilities of moving to different nodes can also be computed in parallel, offering node-level parallelism. Fitness evaluation, which is typically a computational bottleneck in routing problems, can likewise be distributed across threads to achieve fine-grained parallellism. Finally, pheromone evaporation is inherently parallelizable because each edge update is independent, while pheromone deposit can also be parallelized across ants, enabling large-scale concurrent updates. Moreover, multi-colony models provide an additional coarse-grained parallelism strategy, where independent colonies evolve in parallel with occasional exchanges of information.

However, several challenges limit efficient GPU implementation of ACO. Solution construction is sequential in nature, as each ant must select its next city step by step, which restricts fine-grained parallelism. The roulette-wheel selection used for probabilistic city choice introduces control flow divergence and often requires expensive cumulative probability calculations. Pheromone updates create additional difficulties: while evaporation is straightforward, deposits generally require atomic operations to prevent race conditions, which can significantly degrade performance. Moreover, the heavy use of random number generation in ACO demands efficient, independent streams on GPU, and divergence among ants further reduces throughput. As a result, practical implementations often rely on hybrid CPU-GPU models, probabilistic selection optimizations, or hierarchical pheromone update schemes to balance speed and solution quality.

Particle Swarm Optimization (PSO) [KE95; ZWJ15] is another population-based metaheuristic inspired by the collective swarm behavior. In PSO, a set of candidate solutions, called *particles*, move through the search space by updating their positions based on both their own past experiences (personal best *pbest*) and the knowledge of the swarm (global best *gbest*). Each particle balances exploration, driven by randomness and social interaction, with exploitation of known good regions of the search space. PSO was originally developed for continuous optimization, therefore, adapting it to routing problems introduces additional complexities due to the discrete nature of these problems. Specifically, routing problems often require permutation-based solution encoding, which necessitates modifications to the standard PSO algorithm to handle discrete search spaces effectively [BMS17].

PSO is naturally parallel at the particle level, where each particle represents a candidate solution, making it well suited for GPU execution. This allows operations such as updating positions and velocities, as well as fitness evaluation to be distributed across thousands of threads. In routing problems, edge-cost computations within a solution can also be parallelized at a finer granularity, such as the node level. However, note that global best synchronization introduces overhead, since every iteration requires all particles to access the current best solution, creating barriers and communication costs. Additionally, as with other metaheuristics, challenges such as non-coalesced memory accesses and control flow divergence can also significantly reduce GPU efficiency. Finally, the most fundamental challenge still lies in adapting PSO to the discrete search space of routing problems while still exploiting GPU parallelism effectively.

2.3.3 Hybrid metaheuristics

Hybrid metaheuristics are powerful approaches that combine two or more metaheuristic strategies to exploit their complementary strengths. The concept is broad and can include hybrids of single-solution-based methods (e.g., LS + LNS), population-based methods (e.g., ACO + PSO), or, most commonly, a combination of population-based and single-solution-based methods (e.g., GA + LS), which is the focus of this discussion. In routing problems, hybrid metaheuristics have proven particularly effective and represent some of the most competitive state-of-the-art methods [Vid+14; Vid22], as they can effectively balance diversification and intensification by combining global search from population-based strategies with local improvement from single-solution-based methods.

Due to their algorithmic richness and computational demands, the parallelization of

hybrid metaheuristics has attracted significant research interest. Given that hybrids combine single-solution-based and population-based components, the parallel strategies discussed earlier can often be applied independently to different algorithmic components. A common approach is to adopt a hybrid CPU-GPU architecture, where part of the computation is executed sequentially on the CPU and part in parallel on the GPU. For instance, population management can remain on the CPU while local search is parallelized on the GPU, or, conversely, population-level operations can be performed on the GPU while more sophisticated local optimizations are handled on the CPU.

Additionally, hybridization introduces new opportunities and challenges for GPU parallelization. Fully parallelizing the entire framework can improve integration and efficiency but is more complex due to several factors. Synchronization overhead arises from coordinating population-based and local-search phases. Sequential dependencies exist because some local search or LNS operators are inherently sequential. Memory and data management become critical, as hybrid metaheuristics often maintain large or multiple populations, increasing GPU memory demands. Finally, control flow divergence can be intensified by combining heterogeneous operators. These challenges require careful design and trade-off analysis to achieve both high solution quality and GPU efficiency.

2.4 Review of GPU-based metaheuristics for routing problems

Table 2.1 – Publications on GPU-based metaheuristics for routing problems by year (2010-2024)

Metaheuristics	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	Sum
IHC / ILS		1	3	2		1	1	1	3	1	1				1	15
SA						1	1				1					3
TS				1					1							2
VNS							1				2	1				4
LNS										1						1
GA							1		1		2					4
ACO	1			2	1		1	1	1	1	1				1	10
PSO						1		1								2
Hybrid		1		1					1	2		1			1	8
Sum	1	2	3	6	1	3	5	3	7	5	8	1	1	0	2	48

In this section, we provide a comprehensive review of GPU-based metaheuristics for routing problems, covering 48 research papers published since 2010, with particular emphasis on how parallelism has been exploited to accelerate these optimization methods. Table 2.1 provides an overview of the reviewed studies, while Table 2.2 offers a comprehen-

sive summary of 44 key papers selected from the total set. We organize and analyze existing GPU implementations according to their metaheuristic category, following the taxonomy introduced in Section 2.3: single-solution-based metaheuristics, population-based metaheuristics, and hybrid metaheuristics.

2.4.1 GPU-based single-solution-based metaheuristics

This section reviews GPU-based single-solution metaheuristics for routing problems. Most research has concentrated on Local Search (LS) methods, particularly Iterated LS (ILS) and Iterated Hill Climbing (IHC) with 2-opt and 3-opt for the TSP. In contrast, other single-solution metaheuristics have been less studied, with existing works mainly targeting more complex VRP variants.

GPU-based iterated local search

GPU-based Iterated Local Search (ILS) or Iterative Hill Climbing (IHC) for routing problems, particularly the TSP, has received significant attention and forms the foundation for many high-performance methods and GPU-accelerated implementations.

O'Neil et al. [OTB11] pioneered a GPU-based Iterative Hill Climbing (IHC) algorithm for the TSP using 2-opt moves and random restarts. By launching thousands of independent climbers under task parallelism, they achieved over 20 billion tour modifications per second and a 62× speedup compared to CPU implementations on 100-city problems.

Building on this work, Rocki and Suda [RS12b] proposed a more scalable GPU-based IHC solver capable of handling TSP instances with up to 6000 cities. Their key insight was to recompute distances on demand rather than storing the full distance matrix in limited GPU shared memory. They later extended this line of research by developing highly parallel 2-opt and 3-opt implementations using fine-grained data parallelism, where each thread evaluates a city pair [RS12a; RS13]. These algorithms were tested on both TSPLIB benchmarks and massive TSP instances up to 744,710 cities, showing substantial speedups over CPU implementations. Beyond shared-memory reuse for distance computation, they also improved memory efficiency by reordering coordinates to optimize memory access patterns. Nevertheless, they noted that the limited size of GPU shared memory remains a key bottleneck for scaling to very large problem instances.

O'Neil et al. [OB15] further improved their earlier approach by introducing hierarchical parallelism, combining task (across climbers) and partial data parallelism (across the outer

Table 2.2 – Comprehensive summary of GPU-based metaheuristics for routing problems (2010-2025).

Year	Literature	Problem	Algorithm	Parallelism level	Hardware	Benchmark	Acceleration	Solution quality
2010	[FLZ10]	TSP	MMAS	data	GPU: Tesla C1060; CPU: i7 3.3GHz	TSPLIB	-	Close to CPU solution, much less computation time
2011	[OTB11]	TSP	IHC with 2-opt	task	GPU: NVIDIA Tesla C2050; CPU: 8-core Xeon X7550 2.0 GHz	five 100-instances from TSPLIB [Rei91]	20B 2-opt moves/s; 62× faster than x86 core	Optimal in all but one instance (0.07% gap)
2011	[FT11]	TSP	GA + 2-opt	data	GPU: GTX480; CPU: Core2Duo 3.0GHz	TSPLIB (120-512 cities)	40.8× faster	-
2012	[RS12b]	TSP	IHC with 2-opt	task	GPU: NVIDIA Tesla C2050	TSPLIB instances [Rei91]	Slower than [OTB11] but handles >6000 nodes	-
2012	[RS12a]	TSP	ILS with 2-opt, 3-opt	data	GPU: Tesla C2075, GTX 680; CPU: i7 SS70, i7-3960X, Opteron 6276	13 TSPLIB instances (100-4461 cities)	Up to 26× vs 32-core CPU; 500× vs sequential	-
2012	[DDK12]	TSP	ILS with 3-opt	task + data	GPU: NVIDIA Tesla C2050	TSPLIB (100-3038 cities)	Up to 6.02× faster; similar quality as sequential	Gap 0-1.11% depending on instance size
2013	[RS13]	TSP	ILS with 2-opt	data	GPU: GTX 680; CPU: i7-3960X, Radeon HD 7970	TSPLIB (up to 744,710 cities)	5-45× vs 6-core CPU; up to 300× faster convergence	-
2013	[Sch13]	DCVRP	LS with 2-opt, 3-opt	data	GPU: NVIDIA GTX 480	10 CVRP and DCVRP instances (57-2401 nodes)	≈ 10× faster than baseline GPU version	-
2013	[SD13]	DCVRP	TS	task	GPU: Tesla S2050; GTX 480	10 instances (10-500 nodes)	14× vs sequential	-
2013	[Cec+13]	TSP	ACO	data	GPU: Tesla C2050; CPU: Xeon E5620 2.4GHz	TSPLIB (up to 2392 cities)	20× faster than CPU	Equal or better tour quality
2013	[DS13]	TSP	AS	data	GPU: GTX 580; CPU: i7 950	TSPLIB (up to 2392 cities)	Up to 82× faster vs CPU; 8.5× vs GPU baseline	Same or better tour quality
2013	[Del+13]	TSP	MMAS + 3-opt	task + data	GPU: Fermi C2050; CPU: Xeon E5640 2.67GHz	TSPLIB (51-2103 cities)	Up to 23.6× faster	-
2014	[UIN14]	TSP	AS	data	GPU: GTX 580	TSPLIB (up to 3038 cities)	22.11× faster vs CPU	Similar to optimal
2015	[OB15]	TSP	IHC with 2-opt	task + data	GPU: NVIDIA K40	TSPLIB (up to 14,767 cities)	60B 2-opt moves/s; 3× faster than other GPU, 8× faster than CPU	Quality improves as climber count increases
2015	[WWY15]	TSP	SA with swap	data	GPU: GTX 680; CPU: AMD A8-3870K	TSPLIB (280-2392 cities)	Up to 14.84× faster than sequential SA	Up to 27% quality improvement
2016	[ZHQ16]	TSP	ILS with 2-opt	data	GPU: GTX780; CPU: i7-4770 3.4GHz	TSPLIB (198-4461 cities)	279× faster than sequential	Similar solution quality to CPU
2016	[Cor+16]	SVRPDSP	VNS	data	-	68 instances (16-484 customers)	-	Improved results in 51; equal in 7 instances
2016	[Kan+16]	TSP	GA	task	GPU: GTX 980; CPU: Xeon 3.25GHz	TSP (up to 100,000 cities)	22.1× faster	-
2016	[Ski16]	TSP	ACS	data	GPU: Kepler GK104; CPU: Xeon E5-2670	TSPLIB (198-2392 cities)	24.29× vs sequential	Better quality solutions than sequential ACO
2017	[QC17]	TSP	LS with 2-opt	data	GPU: GTX 850M; CPU: i7-4710HQ 2.5GHz	TSPLIB (up to 10,639 cities)	Reduced total runtime	Similar or better solutions than sequential
2017	[ZHQ17]	TSP	ACO	data	GPU: GTX780; CPU: i7-4770 3.4GHz	TSPLIB (198-4461 cities)	44× faster than CPU	Deviation <0.4%
2018	[Rob+18]	TSP	IHC with 2-opt	data	GPU: GTX 780/650 Ti; CPU: Intel i3, Phenom II X4	TSPLIB (up to 33,810 cities)	4.5-22.9× faster; larger graphs gain most	Shorter tours for larger datasets
2018	[YT18]	TSP	IHC with 2-opt	data	GPU: NVIDIA Tesla K40m	TSPLIB (105-85,900 cities)	181× vs sequential; 251× vs GPU-TSP [OB15]	Gap 0.72-8.06%
2018	[Pan+18]	DARP	TS	-	GPU: Tesla P100; CPU: Xeon E5-2620v4	DARP [CL06]	31× faster vs CPU TS	Better quality in less time
2018	[BB18]	DVRP	GA	task + data	GPU: Quadro; CPU: -	DVRP (up to 3000 nodes)	-	Outperforms CPU implementation
2018	[Cec+18]	TSP	ACO	data	GPU: Tesla K40c, GTX580, TITAN X	TSPLIB (up to 3795 cities)	Up to 8× vs baseline	-
2018	[Rey+18]	VRP	MMAS+SPLIT+LS	data	GPU: Tesla T10, M2070	2 VRP benchmark sets	Linear speedup with problem size	Comparable or better quality
2019	[YT19]	TSP	IHC with 2-opt	task + data	GPU: NVIDIA Tesla K40m	TSPLIB (105-85,900 cities)	193× vs CPU; 979.96× vs GPU baseline	Gap 0.72-8.06%
2019	[BHS19]	DCVRP	ALNS	task + data	GPU: GeForce TITAN; CPU: i7-4820K	DCVRP (up to 4500 requests)	6.9× faster vs PRA-8 (8 threads)	Competitive with SOTA
2019	[YKT19]	TSP	MMAS + 2-opt	task + data	GPU: Tesla P100; CPU: Xeon Silver 4114	TSPLIB (up to 33,810 cities)	Up to 60× faster vs sequential	Error 0.52-4.97%
2019	[Men+19]	TSP	ACO	task + data	GPU: Tesla K20 (2496 cores)	TSPLIB (up to 1002 cities)	-	-
2019	[Zha+19]	VRPRB	GMMA variants	task + data	GPU: GTX 950M; CPU: i5-4200 2.8GHz	7 VRPRB instances (up to 200 nodes)	Speedup increases with vehicle count	R-GMMA achieves best performance
2020	[QC20]	TSP	LS with 2-opt, 3-opt	data	GPU: GTX 850M; CPU: i7-4710HQ 2.5GHz	22 national TSP instances (up to 71,009 cities)	362.54s vs >2 days for sequential	Slightly degraded quality
2020	[CAO20]	Multi-UAV (TSP)	SA	data	GPU: GTX 750 Ti; CPU: AMD Phenom II X6	TSPLIB	1.6× faster than CPU	Lower cost than CPU version
2020	[Pan+20]	DARP	VNS, TS	data	GPU: Tesla P100; CPU: Xeon E5-2620v4	DARP [CL06]	55.1× (TS), 22.94× (VNS)	Better solutions in shorter time
2020	[YT20]	CVRP	VNS	data	GPU: Tesla P100; CPU: i7 3.6GHz	CVRPLIB (up to 20,000 customers)	Up to 176.12× faster	-
2020	[Wan+20]	TSP	MPGA (island model)	task	GPU: GTX 960M; CPU: i7-4720HQ	TSPLIB (48-195 cities)	200× faster (pop 20k)	MPGA stable, small deviation from optimum
2020	[Alb+20]	TSP	GA	task	GPU: GTX1060; CPU: i5-7600 3.5GHz	VLSI TSP (up to 984 cities)	Speedup up to 13.73×	-
2020	[Sli20]	TSP	MMAS	data	GPU: Tesla P100, V100; CPU: Xeon 6138 2GHz	TSPLIB (198-3795 cities)	Up to 7.18× vs GPU; 21.79× vs CPU	Finds high-quality solutions up to 18,512 nodes
2020	[AS20]	CVRP	GA + 2-opt	data	GPU: RTX 2080	14 CVRP (16-76 nodes)	Up to 454.4× faster on 76-node instance	Matches or improves best-known values
2021	[YT21]	CVRP	VNS	task + data	GPU: Tesla P100; CPU: i7 3.6GHz	CVRPLIB (up to 30,000 customers)	147.19× speedup	Slightly lower quality than SOTA
2022	[ASS22]	CVRP	GA + 2-opt	data	GPU: DGX-1 (8× Tesla V100)	Large-scale CVRP (up to 20,000 nodes)	1700× faster vs CPU	Gap 4.98% vs best-known
2024	[YUY24]	TSP	ILS with 2-opt, LKH	data	-	7 TSPLIB instances (up to 18,512 cities)	Compression reduces runtime by 15%+	29% faster than SOTA GPU
2024	[YJC24]	TSP	ACO	data	GPU: NVIDIA A100; CPU: i7-10750H 2.60GHz	TSPLIB	Up to 1921× faster than standard ACO	AdaIR improves convergence by 80%, quality +2%

loop of 2-opt) strategies. They also optimized memory access using tiling techniques to reduce global memory loads. On TSP instances with up to 14,767 cities, this method achieved 60 billion tour modifications per second and a 3× speedup over other GPU implementations.

Delévacq et al. [DDK12] developed a GPU-based Iterated Local Search incorporating a decomposition of the 3-opt operator and investigated multiple parallelization strategies across different levels of parallelism. Experiments on TSP instances with 100-3080 cities achieved speedups of up to 6.02×, while maintaining solution quality close to the optimum. However, scalability declined for larger instances.

Schulz [Sch13] integrated high-performance 2-opt and 3-opt operators with an exhaustive best-improvement strategy to tackle the Distance-Constrained Capacitated VRP (DCVRP) on instances of up to 2401 nodes. Using profiling tools, he identified major performance bottlenecks and carefully optimized key aspects of the implementation, including memory hierarchy usage, block size and kernel configuration, data structure selection, and GPU-CPU coordination. The final implementation achieved nearly an order of magnitude speedup compared to baseline GPU implementations.

Zhou et al. [ZHQ16] presented a GPU-based Iterated Local Search (ILS) that exploits data parallelism by mapping neighborhood evaluations to GPU threads and applying a reduction operation [Har+07] to identify the best move. To further enhance memory efficiency, they used texture memory instead of global memory. Experiments on TSPLIB instances with up to 4461 cities showed speedups of up to 279× over the sequential version and 1.7× compared to the GPU-based solver of Rocki and Suda [RS13].

Qiao and Créput [QC17] proposed a simple partial data parallelism scheme for 2-opt where only a set of edges are evaluated simultaneously. Later, they developed the "multiple k-opt evaluation, multiple k-opt moves" approach [QC18; QC20], which evaluates multiple k-opt moves (2-/3-opt) in parallel and applies multiple non-overlapping improvements in a single iteration. This innovation significantly accelerated convergence on 22 large national TSP instances (up to 71,009 cities), outperforming classical sequential and GPU-based 2-opt methods.

Similarly, Robinson et al. [Rob+18] adopted a multi-update strategy in their 2-opt-based hill climbing algorithm, applying multiple improvements per generation. Their method achieved speedups of 4.5× to 22.9× over the existing algorihtm on instances ranging from 1,400 to 33,810 cities.

Yelmewad and Talawar [YT18; YT19] introduced further refinements to GPU-based

IHC with 2-opt, investigating a range of parallelization strategies. Their work explored different GPU memory hierarchies [YT18], CUDA thread-mapping strategies, data-transfer techniques, data structures, and even initial-solution construction methods [YT19]. They reported speedups of up to 251× [YT18] and 979.66× [YT19] over earlier GPU-based solvers [OB15], and demonstrated effectiveness on large TSPLIB instances with up to 85,900 cities.

Recently, Yalcin et al. [YUY24] tackled memory and data transfer bottlenecks in GPU-based methods by introducing data compression techniques. To better exploit shared memory and reduce reliance on the less efficient global memory, they proposed a three-step compression scheme that compresses coordinates of TSP instances to fit within shared memory. Applied to Iterated Local Search (ILS) with 2-opt and the LKH algorithm, their method improved performance by 29% over the state-of-the-art GPU-based TSP solver on instances with up to 18,512 cities.

Other GPU-based single-solution-based metaheuristics

While Iterated Local Search has been extensively parallelized on GPUs, other single-solution metaheuristics such as Simulated Annealing (SA), Tabu Search (TS), Variable Neighborhood Search (VNS) have received comparatively limited attention in the context of GPU parallelism for routing problems.

Wei et al. [WWY15] developed a GPU-based Simulated Annealing (SA) algorithm that parallelizes neighborhood evaluation while partially preserving the first-improvement nature of SA. Instead of selecting the best move, their method randomly chooses a feasible neighbor generated by the *swap* operator. To support this, they employed a modified parallel reduction technique to efficiently identify a candidate move. Experiments showed up to a 14.84× speedup and a 27% improvement in solution quality compared to the sequential implementation.

Additional applications of GPU-accelerated SA have been explored in route planning for multi-UAV systems [TYS16; CAO20], where the canonical SA with an equilibrium loop was adopted, and the equilibrium phase was parallelized on the GPU to better exploit parallelism.

Szymon and Dominik [SD13] introduced a multi-objective Tabu Search variant Parallel Pareto Archive Tabu Search (PPATS) for multi-criteria VRP, using parallel neighborhood exploration with different starting points. Pandi et al. [Pan+18] designed GPU kernels for key TS components: construction, neighborhood generation, and neighborhood evaluation

to reduce CPU-GPU communication latency and improve computational efficiency in solving Dial-A-Ride Problems (DARP). Their approach achieved up to 31× speedup on standard DARP benchmark instances [CL03] compared to a sequential CPU-based Tabu Search algorithm.

Variable Neighborhood Search (VNS) is more suitable to solve more complex VRPs because it systematically changes the neighborhood structures during the search process to enhance both diversification and intensification. By switching between multiple neighborhood operators, VNS increases the likelihood of escaping local optima and exploring new regions of the solution space more effectively.

Coelho et al. [Coe+16] applied VNS to a selective pickup and delivery VRP variant, using GPU-accelerated implementations of four local search operators. Their hybrid CPU-GPU VNS achieved up to 14.49× speedup across large instances.

Pandi et al. [Pan+20] enhanced their earlier GPU-based approach [Pan+18] with device-oriented optimizations, such as hierarchical memory management, adaptive kernel grid configuration, and data transfer optimization, and integrated VNS and TS to solve DARP benchmark instances [CL03]. Their method achieved higher average solution quality within shorter runtimes compared to state-of-the-art sequential solvers.

Yelmewad and Talawar [YT20; YT21] extended their approach [YT18; YT19] to solve large-scale CVRP by combining five heuristics (2-opt, 3-opt, or-opt, relocate, swap). They introduced two GPU parallelization strategies: route-level and customer-level parallelism to speed up the local search phase, which often accounts for 99% of total runtime. Their customer-level design achieved a 147.19× speedup on CVRP instances with up to 30,000 customers.

Finally, due to the challenges of parallelization, GPU-based Large Neighborhood Search (LNS) for routing problems has received limited attention in the literature. To the best of our knowledge, only one study exists: Bach et al. [BHS19] proposed a GPU-based Adaptive Large Neighborhood Search (ALNS) for solving the DCVRP. Their approach adopted two levels of parallelism: (i) task parallelism, where multiple independent ALNS runs were launched from different starting solutions concurrently, and (ii) data parallelism, where the computation of destroy and repair operators, such as Greedy Insertion, Regret-2 Insertion, and Worst Removal, was parallelized using reduction operations. Experiments on benchmark instances with up to 4500 requests demonstrated that their parallel ALNS implementation was competitive in terms of both solution quality and runtime, particularly for larger problem sizes.

2.4.2 GPU-based population-based metaheuristics

This section reviewed studies on GPU-based population-based metaheuristics for routing problems. Among these methods, Genetic Algorithms (GA) and Ant Colony Optimization (ACO) have received the greatest attention for GPU-accelerated routing problem solvers.

GPU-based genetic algorithms

Although Genetic Algorithms (GAs) are among the most widely used metaheuristics for routing problems, only a few studies have explored their GPU-based parallelization for these problems.

Kang et al. [Kan+16] implemented a parallel GA for the TSP under task parallelism. They designed a constructive crossover operator (ARX), which is typically challenging to parallelize. To address this, they employed asynchronous parents pairing and launching multiple crossover operations concurrently to effectively exploit parallelism. Experiments on TSP instances with up to 100,000 cities demonstrated up to 22.1× speedup over CPU version, using different population sizes and grouping strategies.

Benaini and Berrajaa [BB18] proposed the first GPU-based GA for large-scale Dynamic VRP (DVRP). Their algorithm adopts a periodic reoptimization strategy: newly arriving customer requests are first inserted into the current solution using data parallelism to accelerate insertion evaluation, followed by reoptimization through GA operators (selection, crossover, mutation) parallelized at the task level. This hybrid parallel design enables efficient solution of instances with up to 3000 nodes, outperforming existing approaches on benchmark datasets.

Wang et al. [Wan+20] introduced a multi-population GA using an island model for the TSP. The algorithm initializes a large population and partitions it into multiple subpopulations, with genetic operations (selection, crossover, mutation) performed independently and in parallel within each subpopulation. A migration mechanism allows selected individuals to move between subpopulations. Leveraging GPU parallelism, the algorithm efficiently handles large-scale populations and achieves high-quality solutions in a short time, achieving up to 200× speedup compared to CPU baselines on selected TSPLIB instances with population sizes up to 20,000.

Abbasi et al. [Abb+20] proposed a highly parallel GA for the TSP targeting both multi-core CPUs and many-core GPUs. They implemented three dedicated GPU kernels

for genetic operations under chromosome-level parallelism and used an efficient synchronization mechanism to coordinate threads. Their algorithm handled populations up to 16,384 individuals and achieved up to 13.73× speedup on GPU. They also proposed integrating their GA into vehicular cloud computing systems for real-time traffic management applications.

GPU-based ant colony optimization

Due to its inherently parallel population-based nature, GPU-based ACO for routing problems has attracted significant research attention. However, many challenges remain as discussed in Section 2.3.2, prompting the development of various strategies in the literature to address these bottlenecks.

Several works focus on parallelizing tour construction, pheromone update and improving the efficiency of probabilistic node selection on GPU.

Fu et al. [FLZ10] proposed a hybrid CPU-GPU implementation of the MAX-MIN Ant System (MMAS) for the TSP, where all ants share global matrices (e.g., pheromone, probability, and taboo matrix). Their All-In roulette selection mechanism was tailored to maximize GPU data-parallel efficiency.

Cecilia et al. [Cec+13] proposed new parallelization mechanisms for ACO under data parallelism. They introduced I-Roulette (Independent Roulette) for tour construction, which improves control-flow regularity and reduces thread divergence, and employed atomic instructions for pheromone updates to ensure consistency in concurrent execution. Their implementation achieved up to 20× speedup on TSPLIB instances up to 2392 cities.

Then, Dawson and Stewart [DS13] proposed Double-Spin Roulette (DS-Roulette), a novel selection method for parallel Ant System (AS) and significantly enhanced parallel efficiency. Their approach achieved speedups of up to 82× over a sequential CPU baseline and up to 8.5× compared to earlier GPU-based implementations [Cec+13], while maintaining solution quality.

Zhou et al. [ZHQ17] followed a similar approach and proposed a data-parallel ACO. They designed two kernel strategies for the tour construction phase and introduced a new parallel roulette-wheel selection method called Tiling Roulette. Their implementation achieved speedups of up to 44.7× on TSPLIB instances with up to 4461 cities.

Later, Cecilia et al. [Cec+18] extended their work [Cec+13] by incorporating vectorization techniques and introducing SS-Roulette (SCAN and STENCIL Roulette), an

enhanced roulette method that further improves parallelism and efficiency on modern CUDA architectures, achieving up to an additional 8× speedup over their earlier implementation [Cec+13].

Skinderowicz [Ski16] proposed a GPU-based parallel implementation of the Ant Colony System (ACS) that leverages data parallelism to fully exploit GPU capabilities. Several parallel strategies were investigated, particularly focusing on pheromone memory update mechanisms to minimize synchronization barriers and reduce reliance on atomic instructions. Experiments on TSPLIB instances with 198 to 2392 cities reported speedups of up to 24.29× compared to the sequential CPU implementation.

In subsequent work, Skinderowicz proposed a GPU-based MAX-MIN Ant System (MMAS) implementation [Ski20] focusing on the solution construction phrase, which is the most time-consuming part of the algorithm. The main contributions include a novel parallel node selection method based on weighted reservoir sampling and a memory-efficient tabu list design. Extensive experiments on TSPLIB instances of up to 3795 cities demonstrated the competitiveness of this approach, achieving up to 21.79× speedup over a multi-core CPU parallel implementation. Furthermore, a GPU-parallelized 2-opt local search was incorporated, achieving high-quality solutions for large TSP instances with up to 18,512 cities.

Efficient memory access patterns are crucial for achieving high performance in GPU parallelization. Uchida et al. [UIN14] addressed key GPU programming challenges in their parallel AS algorithm, such as coalesced global memory access and shared memory bank conflicts. Their implementation carefully arranges data in global memory to maximize bandwidth utilization and exploits shared memory to minimize costly global memory accesses. As a result, they achieved a speedup of up to 22.11× over the CPU implementation.

Some studies compared parallelization strategies. Menezes et al. [Men+19] conducted a comprehensive analysis of coarse-grained and fine-grained ACO parallelism. Their study revealed that while coarse-grained methods may underutilize GPU resources, fine-grained designs may incur higher synchronization overheads. The optimal strategy depends on instance size and colony configuration.

Recent works have explored modern GPU features and tensor-friendly representations. Yang et al. [YJC24] introduced Tensorized ACO (TensorACO), in which pheromone matrices and ant paths are represented as tensors to fully leverage GPU parallelism. They proposed an Adaptive Independent Roulette (AdaIR) to accelerate convergence while maintaining solution quality. TensorACO demonstrated up to 1921× speedup over a stan-

dard Python implementation on TSPLIB instances.

Additionally, although Particle Swarm Optimization (PSO) is inherently parallel and well suited to GPU execution, its parallelization for routing problems has been relatively limited. Only a few studies have explored GPU-based PSO for routing, notably the work of Bali et al. [Bal+15; Bal+17], who conducted preliminary experiments on GPU-accelerated PSO for the TSP. However, their implementation remained fairly basic and was evaluated only on small-scale instances.

2.4.3 GPU-based hybrid metaheuristics

This section reviewed studies on GPU-based hybrid metaheuristics for routing problems, particularly those that combine population-based and single-solution-based methods. At this level, a common hybrid strategy involves integrating GA or ACO with local search techniques such as 2-opt, 3-opt, and related operators, achieving a balanced trade-off between exploration and exploitation.

Fujimoto and Tsutsui [FT11] proposed a highly parallel genetic algorithm based on Order Crossover (OX) and data-parallel 2-opt local search for solving the TSP on NVIDIA CUDA architecture. Their implementation achieved a speedup of 40.8× over the corresponding CPU version. However, the algorithm was limited to instances of up to 512 cities due to the restricted search capability of 2-opt on larger problems.

Delévacq et al. [Del+13] proposed a GPU-based hybrid method combining Max-Min Ant System (MMAS) with 3-opt local search for TSP, exploring multiple levels of parallelism. Their parallel GPU implementation achieved up to 23.6× speedup while maintaining solution quality comparable to the sequential version.

Yelmewad et al. [YKT19] implemented both task and data parallelism in MMAS to tackle very large TSP instances. In their design, task parallelism assigns one ant per thread, whereas data parallelism assigns one ant per thread block. A parallel 2-opt procedure was also integrated to enhance solution quality. Experiments on instances with up to 33,810 cities showed that the data parallelism approach outperformed task parallelism, with relative speedups ranging from 1.02× to 3.37×. When compared to the sequential counterpart, their best GPU implementation achieved overall speedups of up to 60×.

Rey et al. [Rey+18] developed an algorithm for solving the VRP under the hybrid CPU-GPU setting. The algorithm first applied a GPU-parallelized MMAS ACO [FLZ10] to generate TSP routes, followed by CPU-based operations using the Route First-Cluster Second method and Local Search to convert TSP routes into VRP solutions. Although

only the ACO phase was accelerated on GPU, the hybrid setup demonstrated potential for balancing workload between devices.

Zhang et al. [Zha+19] introduced a GPU-based multi-objective memetic algorithm (GMMA) for solving the VRP with route balancing (VRPRB). Their framework combined a multi-objective evolutionary algorithm with several local search operators. Two parallelization schemes were proposed: solution-level parallelism, where different individuals were processed in parallel, and route-level parallelism, offering finer granularity by parallelizing local search operations within each solution. However, they did not exploit node-level parallelism typically used in GPU-based neighborhood evaluations.

Abdelatti and Sodhi [AS20] proposed a fully GPU-accelerated genetic algorithm incorporating 2-opt local search to solve the Capacitated VRP (CVRP). Their implementation parallelized all major components including population initialization, reproduction, and refinement. They later extended their work using multi-GPU parallelization [ASS22] to handle very large CVRP instances with up to 20,000 nodes. Their approach mapped all algorithmic arrays to thread blocks, maximizing parallelism and ensuring high throughput with reduced latency, leading to speedups of up to 1700× over CPU implementations. They also observed that performance improved consistently as the number of GPUs increased within the tested range.

2.5 Discussion

This section summarizes the current state of GPU-based metaheuristics for routing problems, highlighting their achievements, identifying their main limitations, and outlining promising directions for future research.

As reviewed in Section 2.4, GPU-based metaheuristics have achieved notable progress. Significant performance gains, which are often expressed as strong speedups over sequential CPU implementations, have been reported, especially for classical routing problems such as the TSP. Due to its simple definition and the efficiency of local heuristics like 2-opt and 3-opt, the TSP has served as an ideal testbed for exploring parallelization strategies across various metaheuristic frameworks. Many studies have implemented sophisticated low-level CUDA designs, involving thread control, memory management, and kernel optimization. Furthermore, large, standardized benchmark suites such as TSPLIB [Rei91] have facilitated fair and consistent performance evaluation.

In contrast, GPU-based research on Vehicle Routing Problem (VRP) variants remains

relatively limited. Most studies focus on foundational variants such as the CVRP [AS20; ASS22] and DCVRP [Sch13; BHS19], while only a few address more complex extensions [BB18]. Very few works have investigated complex VRP variants in real-world settings, which often involve large-scale problems characterized by dynamic, uncertain, stochastic, or time-dependent constraints that require real-time decision-making.

This gap is largely due to the more complex problem modeling and solution space of such variants, which complicate parallelization. Nevertheless, domains such as dynamic vehicle routing, drone delivery, ride-sharing, and logistics represent promising application areas where high-throughput, adaptive GPU-based solutions could yield substantial benefits.

Another limitation lies in benchmarking. Unlike the TSP, which benefits from large and standardized datasets [Rei91] that support comprehensive scalability evaluations, complex VRP variants lack comparable benchmark resources. Most existing instances were originally designed for evaluation of CPU-based algorithms and typically involve fewer than 1000 customers, which insufficiently tests the scalability of modern GPU architectures. Consequently, there is a pressing need for new, large-scale, publicly available benchmark suites that better capture real-world problem complexity and enable fair assessment of GPU-based algorithms.

Furthermore, while significant advances have been made, most existing studies focus on a single problem type. The field lacks a unified, efficient, and scalable GPU-based framework that can address a wide variety of routing problems. Such a framework should integrate support for diverse problem structures, offer adaptable parallelization mechanisms, and be readily accessible to both researchers and practitioners in combinatorial optimization.

Beyond the problem scope, another major challenge in the literature is the inconsistency of evaluation methodologies. Although Section 2.2.2 presents common performance metrics, practical implementations vary widely. For example, while many studies report speedup ratios according to Equation 2.1, others adopt alternative metrics, such as the number of operations per second [OTB11] or the time required to perform a single local search step [RS12a]. Furthermore, only [ASS22] explicitly applies the strong scaling formulation in Equation 2.2 to evaluate scalability.

In terms of solution quality, most studies prioritize speedup performance, with limited attention to achieving or maintaining high-quality solutions. Consequently, few works perform rigorous comparisons against state-of-the-art CPU-based metaheuristics under stan-

dardized conditions. The trade-off between speed and solution quality therefore remains underexplored, leaving the true competitiveness of GPU-based approaches uncertain.

This issue is partly due to architectural differences: CPU-based methods often employ sophisticated search strategies to avoid redundant computations, whereas GPU-based algorithms typically apply uniform operations across all data to fully exploit massive parallelism. Although this uniformity increases throughput, it can lead to suboptimal algorithmic behavior and complicate fair comparisons between CPU and GPU-based algorithms.

Moreover, despite the availability of profiling tools that can identify performance bottlenecks, only a few studies have employed them [Sch13; ASS22]. Reproducibility is another concern, as variations in GPU hardware, software environments, and the lack of open-source implementations hinder consistent benchmarking and independent verification of results.

Additionally, most existing GPU-based approaches rely on CUDA, which offers fine-grained control but demands substantial programming effort. Developers must explicitly manage memory allocation, thread synchronization, and kernel scheduling, making GPU programming both time-consuming and error-prone. This complexity poses a steep learning curve for researchers without deep expertise in GPU computing.

To overcome these barriers, recent studies have explored tensorization as a high-level and scalable alternative. By formulating metaheuristic operations as tensor computations, frameworks such as PyTorch, TensorFlow, JAX, and CuPy enable efficient parallelism without requiring manual low-level optimization. Works like [YJC24] demonstrate that core algorithmic components can be implemented efficiently through high-level tensor abstractions.

This trend aligns with the broader integration of deep learning and combinatorial optimization, where tensor computation serves as a unifying paradigm. Tensorized implementations facilitate hybrid models that combine classical heuristics with machine learning techniques, such as neural networks, reinforcement learning, and large language models (LLMs). These hybrid approaches promise both high performance and adaptability, leveraging the rich tooling and scalability of modern ML ecosystems.

Finally, sustainability considerations should not be overlooked. While GPUs offer significant performance advantages, they also involve high energy consumption and cost. Balancing performance with environmental and economic factors will be increasingly important in the design of future large-scale GPU-based optimization systems.

In summary, while GPU-based methods for routing problems have achieved substantial progress, several limitations remain, particularly in problem scope, evaluation rigor, reproducibility, and framework accessibility. Overcoming these challenges through the development of standardized benchmarks, adoption of high-level tensor-based programming models, and integration with emerging technologies could pave the way for the next generation of scalable and effective optimization frameworks.

2.6 Chapter conclusion

This chapter has provided a comprehensive review of recent advances in GPU-based metaheuristics for solving routing problems since 2010, including the Traveling Salesman Problem (TSP), the Vehicle Routing Problem (VRP), and their numerous variants. We revisited foundational concepts in metaheuristics, routing problem formulations, and GPU parallel computing. Standard benchmark datasets and evaluation metrics were also reviewed. In particular, we analyzed opportunities for GPU-based metaheuristics and discussed recent developments in GPU-accelerated methods, emphasizing specific strategies for exploiting parallelism and common GPU programming practices.

While GPUs provide significant computational advantages through massive parallelism, this review highlights several key limitations. Research remains largely focused on classical problem variants, with limited attention to complex, real-world scenarios. Benchmarking and evaluation practices are inconsistent, and reproducibility is often constrained by differences in hardware and the absence of open-source implementations. Moreover, the field still lacks scalable, unified frameworks that are both efficient and broadly accessible to the research community.

Several promising directions for future research have been identified. First, greater attention should be given to dynamic, stochastic, and uncertain problem settings, where GPU-based solvers could have a transformative impact on real-world applications. Second, to enable rigorous and reproducible evaluation, there is a critical need for larger and more diverse benchmark datasets specifically designed for complex VRP variants. Finally, advances in tensorization and high-level GPU programming frameworks offer opportunities to develop modular, maintainable, and high-performance metaheuristics. Integrating these frameworks with emerging technologies, including machine learning, reinforcement learning, and large language models, provides another promising avenue to further enhance solution quality and adaptability.

Chapter 3

Memetic algorithm for vehicle routing with simultaneous pickup and delivery and time windows

---

The Vehicle Routing Problem with Simultaneous Pickup and Delivery and Time Windows (VRPSPDTW) has a number of real-world applications, especially in reverse logistics. In this chapter, we propose an effective memetic algorithm that integrates a lightweight feasible and infeasible route descent search and a learning-based adaptive route-inheritance crossover to solve this complex problem. We evaluate the effectiveness of the proposed algorithm on the set of 65 popular benchmark instances as well as 20 real-world large-scale benchmark instances. We provide a comprehensive analysis to better understand the design and performance of the proposed algorithm. The content of this chapter is based on an article published on *IEEE Transactions on Evolutionary Computation*.

3.1 Introduction

Since the introduction of VRPSPDTW [AM02], many algorithms have been developed to solve this problem. Section 1.2.2 provides a comprehensive review of related studies. The inherent complexity of the problem, as introduced in Section 1.2.1, makes it a challenging VRP variant with multiple complex characteristics, including dynamic load changes, time windows, and hierarchical objectives. The primary objective is to minimize the number of vehicles, while the secondary objective is to minimize the total travel distance. As a result, no single method consistently performs best across all benchmark instances. Additionally, research on large instances and real-world applications remains limited. Meanwhile, a notable trend in recent algorithm design is the use of large neighborhood search techniques with destroy and repair operators [HS19; LTY21; WG23]. While these strategies enhance search space exploration, they also increase algorithmic complexity.

To address these challenges, we propose a novel algorithm, MA-FIRD, which is based on the memetic algorithm framework. This framework has been proven effective for solving various VRPs, including VRPTW [Vid+13], split delivery VRP [HH23a], and multi-trip VRP [Cat+14]. One recent work [LTY21] has also showed the effectiveness of the memetic algorithm approach for VRPSPDTW. Following this framework, our proposed algorithm integrates several novel components and strategies. These include a lightweight feasible and infeasible route descent search where a dedicated penalty function is embedded to manage infeasible solutions, and a multi-parent route-inheritance crossover supported by reinforcement learning. Furthermore, we implement a max-min normalization-based fitness-distance population management strategy to maintain population diversity. The proposed algorithm is evaluated on the popular WC benchmark [WC12] and the large-scale real-world JD benchmark [LTY21], demonstrating its superiority and practicality compared to existing algorithms.

The rest of this chapter is organized as follows. Section 3.2 presents the proposed algorithm. Section 3.3 shows computational results on benchmark instances compared to state-of-the-art methods, highlighting its ability to solve real-world large-scale instances. Section 3.4 provides an in-depth analysis of the key components of the algorithm. Section 3.5 concludes this chapter and suggests possible avenues for future research.

3.2 Memetic algorithm with feasible and infeasible route descent search for VRPSPDTW

The memetic algorithm framework [MC10; NC12] combines the advantages of genetic algorithms and local search methods, providing an interesting way to balance search diversification and intensification. Leveraging this powerful framework, we design dedicated search operators and strategies to address the diverse features and constraints of VRPSPDTW. Specifically, we propose a lightweight feasible and infeasible route descent search, a learning-based adaptive route-inheritance crossover, and a max-min normalization-based fitness-distance population management strategy to enable efficient search.

Algorithm 1 Main framework of MA-FIRD

1: **Input**: Instance $\mathcal{I}$, Population size $\mathcal{N}_{\mathcal{P}}$, Maximum number of generations φ_{max}, Patience for stagnation generations ρ, Maximum running time τ.
2: **Output**: The best solution S^*.
3: $\varphi \leftarrow 0, \varphi_{st} \leftarrow 0$ /* Current generation and stagnation generation counter */
4: $\mathcal{I} \leftarrow Preprocessing(\mathcal{I})$ /* Section 3.2.1 */
5: $\mathcal{P} \leftarrow Initialization(\mathcal{I}, \mathcal{N}_{\mathcal{P}})$ /* Section 3.2.2 */
6: $S^* \leftarrow BestSolution(\mathcal{P})$ /* Record current best solution */
7: **while** $\varphi \leq \varphi_{max}$ **and** $\varphi_{st} \leq \rho$ **and** $time() \leq \tau$ **do**
8: $S' \leftarrow ARIX(\mathcal{P})$ /* Section 3.2.4 */
9: $S' \leftarrow FIRDSearch(S')$ /* Section 3.2.3 */
10: $\mathcal{P} \leftarrow UpdatePopulation(S', \mathcal{P})$ /* Section 3.2.6 */
11: $S_{best} \leftarrow BestSolution(\mathcal{P})$ /* The best solution in this generation */
12: **if** $f(S_{best}) < f(S^*)$ **then**
13: $S^* \leftarrow S_{best}$ /* Update the best solution */
14: $\varphi_{st} \leftarrow 0$
15: **else**
16: $\varphi_{st} \leftarrow \varphi_{st} + 1$ /* Stagnation counter is incremented */
17: **end if**
18: $\varphi \leftarrow \varphi + 1$
19: **end while**
20: **return** S^*

Algorithm 1 outlines the framework of MA-FIRD. For a given problem instance $\mathcal{I}$, the algorithm starts with a preprocessing step to reduce the instance (line 4). Then, $\mathcal{N}_{\mathcal{P}}$ candidate solutions are generated to form the initial population (line 5), and the best solution S^* is recorded (line 6). The population is updated iteratively by generating new candidate solutions (lines 7-19). In each generation, the adaptive route-inheritance crossover

creates a new solution from multiple parent solutions (line 8). This new solution S' is then improved by the feasible and infeasible route descent search (line 9). The improved solution S' is used to update the population according to the population management strategy (line 10). If a superior best solution is found, S^* is updated, and the stagnation counter is reset to 0 (lines 12-14). Otherwise, the stagnation generation counter φ_{st} is incremented (lines 15-16). The algorithm terminates and returns the best solution S^* found (line 20) when the termination condition is met (line 7), which includes reaching a maximum number of generations φ_{max}, surpassing a maximum patience threshold ρ for stagnation generations φ_{st}, or exceeding a predetermined maximum running time τ. Below, we provide a detailed description of each component of the algorithm.

3.2.1 Preprocessing

To improve computational efficiency, a preprocessing step is employed before executing the main algorithm to prune the search space. As outlined in Section 1.2.1, VRPSPDTW can be defined on a directed complete graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the vertices $\mathcal{V} = \{v_0, v_1, \ldots, v_N\}$ represent the depot and customer nodes. Each customer node is associated with a delivery demand d_i, a pickup demand p_i, a time window $[e_i, l_i]$, and a service time s_i. The edges $\mathcal{E} = \{e_{ij} | v_i, v_j \in \mathcal{V}\}$ represent the connections between the nodes, each with an associated travel cost c_{ij} and travel time t_{ij}. The vehicle capacity is denoted by $\mathcal{Q}$. The preprocessing procedure removes edges that violate the time window or vehicle capacity constraints of VRPSPDTW. For any pair of nodes v_i and v_j, the directed edge e_{ij} is eliminated if one of the following rules is true.

$$e_i + s_i + t_{ij} > l_j \tag{3.1}$$

$$d_i + d_j > \mathcal{Q} \tag{3.2}$$

$$p_i + d_j > \mathcal{Q} \tag{3.3}$$

$$p_i + p_j > \mathcal{Q} \tag{3.4}$$

Rule (3.1) allows the connection between v_i and v_j to be pruned if the arrival time at v_j after serving v_i is later than the allowed latest time l_j for v_j. Rules (3.2), (3.3), and (3.4) prohibit the connection between v_i and v_j if the capacity constraint is violated before, between, and after consecutive visits to v_i and v_j.

3.2.2 Initialization

The initialization stage generates a group of candidate solutions to form the initial population. A high-quality initial solution, characterized by its feasibility and a relatively lower number of vehicles, helps the algorithm in its subsequent search. We use the RCRS algorithm [Det01] to generate such initial solutions. RCRS is an insertion-based heuristic that starts with an empty solution and iteratively inserts customer nodes until all nodes are accommodated. Notably, RCRS only accepts feasible insertions, and a new route is introduced if no feasible insertion is possible with the existing routes. In addition to considering travel cost, RCRS incorporates two other metrics: Residual Capacity (RC), which represents the insertion's freedom in terms of capacity, and Radial Surcharge (RS), which is to prevent late and unfavorable insertions of remote customer nodes. Using these three metrics, the RCRS algorithm efficiently constructs a relatively high-quality solution in a short time. To generate a diverse initial population, the RCRS algorithm is executed multiple times with randomly assigned weights for the RC and RS metrics, resulting in a set of distinct solutions.

3.2.3 Feasible and infeasible route descent search

This section presents the feasible and infeasible route descent search (FIRD), which is one key component of the proposed algorithm.

General FIRD procedure

As shown in [GH11], allowing controlled exploration of infeasible solutions facilitates the transition between structurally different feasible solutions and thus improves the performance of local search algorithms. Indeed, this approach has proven successful in several difficult combinatorial optimization problems with complex constraints (see, e.g., [LHW22; Sun+18; ZHW24]).

Therefore, the proposed FIRD procedure adopts a feasible and infeasible search framework for solving the VRPSPDTW. In our design, FIRD consists of two main components: (i) a route descent (RouteDescent) procedure and (ii) a feasible and infeasible search (FIS) procedure. The route descent procedure explicitly addresses the primary objective of minimizing the number of vehicles by iteratively removing a route and reinserting its nodes into other routes. The FIS procedure further improves the solution obtained from the route descent procedure by introducing penalty terms for constraint violations in

the evaluation function (Section 3.2.3). Combined with a set of local search operators (Section 3.2.3), FIS can effectively transition between feasible and infeasible regions of the search space, allowing the search to explore and identify high-quality solutions that would otherwise be difficult to obtain.

Algorithm 2 Feasible and infeasible route descent search (FIRD)

```
1:  Input: The solution S for improvement.
2:  Output: Improved solution S'.
3:  S' ← S                                              /* Current best solution */
4:  if S is infeasible then
5:     S ← FIS(S, S')                                   /* Try to make S feasible */
6:     S' ← S
7:  end if
8:  while S is feasible do
9:     S ← RouteDescent(S)                              /* Remove a route of S */
10:    if S is better than S' then
11:       S' ← S
12:    else
13:       S ← FIS(S, S')      /* Optimize S and try to find a better solution than S' */
14:       if S is better than S' then
15:          S' ← S
16:       end if
17:    end if
18: end while
19: if S' is feasible then
20:    S' ← FIS(S', ∅)                /* Intensive search from the best-found solution S' */
21: end if
22: return S'
```

As shown in Algorithm 2, the FIRD procedure starts with an input solution S, which can be either feasible or infeasible. If S is infeasible, the FIS is applied to ensure that the solution entering the route descent procedure is feasible (lines 4-7). The route descent procedure removes the route with the fewest nodes from the solution S, and its nodes are reinserted into other routes using the infeasible regret insertion strategy presented in Section 3.2.5 (line 9), as illustrated in Figure 3.1. If the resulting solution of route descent is not feasible or not better than the current best solution S', the FIS is re-applied to try to find a better solution (line 13). The route descent procedure continues until it becomes impossible to find a feasible solution with the current number of vehicles (lines 8-18). Finally, an intensive search is performed on the best feasible solution S' (lines 19-21), and

the final solution S' is returned (line 22).

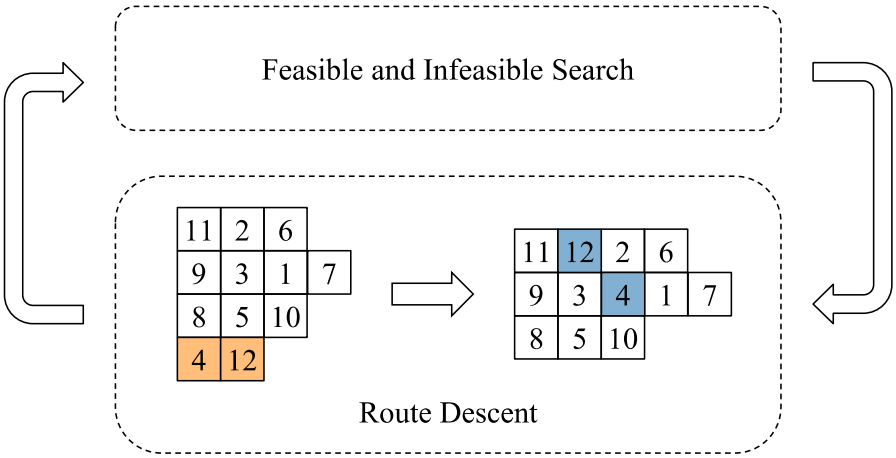


Figure 3.1 – Illustration of the route descent procedure. The solution has four routes. First, the route with the fewest nodes 4-12 is removed. Then, the nodes 4 and 12 are inserted into other routes using the infeasible regret insertion strategy.

The FIS takes two inputs: the solution to be improved and an optional target solution. If a target solution (e.g., the best-found solution S' in lines 5 and 13) is provided, the search terminates prematurely if a better solution is found. If no target solution is provided, the search intensively improves the input solution (e.g., the best-found solution S' in line 20).

Evaluation function

The feasible and infeasible search dynamically accepts feasible and infeasible solutions to explore diverse areas of the search space. To achieve this, we define an evaluation function that incorporates penalty terms of time window and capacity constraint violations, as shown in Equation (3.5).

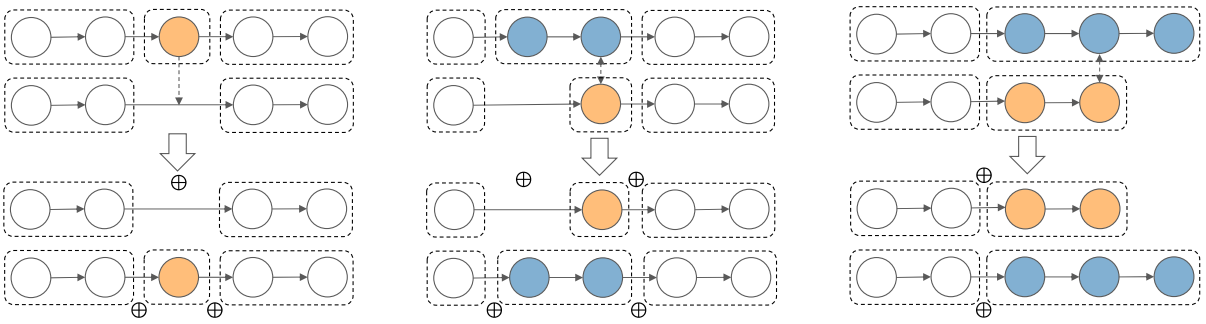
$$
\begin{aligned}
\text{Minimize} \quad & \mathcal{F}(S) = D(S) + \lambda_1^k \cdot V_1(S) + \lambda_2^k \cdot V_2(S) \\
\text{Subject to} \quad & D(S) = \sum_{i=1}^{M} \sum_{j=0}^{\mathcal{L}} c_{n_{i,j} n_{i,j+1}} \\
& V_1(S) = \frac{\sum c_k}{\sum t_k} \sum_{i \in R_r} \sum_{j=0}^{\mathcal{L}} \max(a_{i,j} - l_{i,j}, 0) \\
& V_2(S) = (2 \cdot \max c_k - \min c_k) \frac{\sum_{i \in R_r} \max(\max_{j=0}^{\mathcal{L}}(q_{n_{i,j}} - \mathcal{Q}), 0)}{\mathcal{Q}} \\
& S = \{R_1, \ldots, R_M\} \\
& R_i = \{n_{i,0}, n_{i,1}, \ldots, n_{i,\mathcal{L}}, n_{i,\mathcal{L}+1}\}, \quad i = 1, \ldots, M
\end{aligned}
\tag{3.5}
$$

$$
\lambda_i^k = \begin{cases} \lambda_i^k \cdot \kappa & \text{if } V_i(S) = 0 \\ \frac{\lambda_i^k}{\kappa} & \text{if } V_i(S) > 0 \end{cases} \quad i = 1, 2, \quad \kappa \in (0, 1) \tag{3.6}
$$

Since the number of vehicles (primary objective) is handled through the route descent strategy, the feasible and infeasible search only focuses on the travel distance objective $D(S)$. $V_1(S)$ and $V_2(S)$ represent the normalized violation values of time window and capacity constraints, respectively. The time-warp technique [NBD10; Vid+13] is adopted to address violation of time window constraints. During the search process, the coefficients λ_1^k and λ_2^k are dynamically adjusted at each move step based on the feasibility of the constraints. Here, k denotes the iteration count during the search. This adjustment is facilitated by the coefficient adjustment factor κ, as depicted in Equation (3.6). If the time window or capacity constraint is violated, the corresponding coefficient is increased to prioritize satisfying that constraint. If both constraints are violated, the coefficients λ_1^k and λ_2^k are both increased. Conversely, if there is no constraint violation, the coefficients λ_1^k and λ_2^k are decreased to encourage exploration of infeasible areas.

Move operators and move evaluation

The feasible and infeasible search employs a set of local search move operators to explore the search space, including *Relocate*, *Swap*, and *2-opt**. *Relocate* moves a sequence of nodes to the same or another route. *Swap* exchanges two sequences of nodes in the



(a) Relocate (b) Swap (c) 2-opt*

Figure 3.2 – Illustration of operators and concatenation operation.

same route or different routes. *2-opt** takes place between two routes. It first breaks one edge in two candidate routes, and then exchanges and reconnects the sequences of nodes. Here, the sequence length of nodes for *Relocate* and *Swap* operators ranges from 1 to 3. Considering their similar complexity of the employed operators, a random order is employed to execute these operators. The feasible and infeasible search stops under several conditions: when no improvement is detected after applying all operators in the sequence, or when the maximum iteration limit is reached, or when a better solution than the given target solution is found.

The search procedure adopts the best-improvement strategy to apply each local search operator. To speed up move evaluation and accommodate infeasible solutions, we introduce an infeasibility-allowed constant-time move evaluation method inspired by prior work [LTY21; Vid+13]. Since the used operators affect only one or two routes, the move evaluation process can be performed in constant time by computing the variation (called move gain) of the evaluation function between the original solution and a new solution. We conceptualize moves as rearrangements of subsequences of routes, using a concatenation operation denoted as $\oplus$ to symbolize the merging of two route subsequences. Illustrative examples of various operators are depicted in Figure 3.2, while detailed concatenation operation calculations are provided in the next section. With this method, the computation of the $D(\cdot)$, $V_1(\cdot)$, and $V_2(\cdot)$ values in the evaluation function can be done in constant time, which greatly speeds up the evaluation process.

Calculations of concatenation operation

We provide detailed calculations of the concatenation operation, denoted as $\oplus$, for constant-time move evaluation of the local search operators.

Since the move of the local search operators can be represented as a rearrangement of subsequences of routes, the evaluation of the move can be computed efficiently by maintaining a set of metrics for each route, including travel distance (D), duration time (T_d), earliest and latest arrival times (T_e and T_l), waiting time (T_w), violated time window value (T_v), load values before and after the sequence (L_{in} and L_{out}), and the maximum load (L_{max}).

For a subsequence containing a single node, denoted as $\sigma^0 = \{v_i\}$, these metrics can be defined:

$$D(\sigma^0) = 0 \tag{3.7}$$
$$T_d(\sigma^0) = s_i \tag{3.8}$$
$$T_e(\sigma^0) = e_i \tag{3.9}$$
$$T_l(\sigma^0) = l_i \tag{3.10}$$
$$T_w(\sigma^0) = 0 \tag{3.11}$$
$$T_v(\sigma^0) = 0 \tag{3.12}$$
$$L_{in}(\sigma^0) = d_i \tag{3.13}$$
$$L_{out}(\sigma^0) = p_i \tag{3.14}$$
$$L_{max}(\sigma^0) = \max(d_i, p_i) \tag{3.15}$$

For two subsequences of routes, $\sigma_1 = \{v_i, \ldots, v_j\}$ and $\sigma_2 = \{v_k, \ldots, v_l\}$, the corresponding metrics of the combined sequence of routes $\sigma_1 \oplus \sigma_2$ can be calculated by:

$$D(\sigma_1 \oplus \sigma_2) = D(\sigma_1) + c_{j,k} + D(\sigma_2) \tag{3.16}$$
$$\Delta_t = T_d(\sigma_1) + T_w(\sigma_1) + t_{j,k} - T_v(\sigma_1) \tag{3.17}$$
$$\Delta_w = \max(T_e(\sigma_2) - \Delta_t - T_l(\sigma_1), 0) \tag{3.18}$$
$$\Delta_v = \max(T_e(\sigma_1) + \Delta_t - T_l(\sigma_2), 0) \tag{3.19}$$
$$T_d(\sigma_1 \oplus \sigma_2) = T_d(\sigma_1) + t_{j,k} + T_d(\sigma_2) \tag{3.20}$$
$$T_e(\sigma_1 \oplus \sigma_2) = \max(T_e(\sigma_1), T_e(\sigma_2) - \Delta_t) - \Delta_w \tag{3.21}$$
$$T_l(\sigma_1 \oplus \sigma_2) = \min(T_l(\sigma_1), T_l(\sigma_2) - \Delta_t) + \Delta_v \tag{3.22}$$
$$T_w(\sigma_1 \oplus \sigma_2) = T_w(\sigma_1) + \Delta_w + T_w(\sigma_2) \tag{3.23}$$
$$T_v(\sigma_1 \oplus \sigma_2) = T_v(\sigma_1) + \Delta_v + T_v(\sigma_2) \tag{3.24}$$
$$L_{in}(\sigma_1 \oplus \sigma_2) = L_{in}(\sigma_1) + L_{in}(\sigma_2) \tag{3.25}$$
$$L_{out}(\sigma_1 \oplus \sigma_2) = L_{out}(\sigma_1) + L_{out}(\sigma_2) \tag{3.26}$$
$$L_{max}(\sigma_1 \oplus \sigma_2) = \max(L_{max}(\sigma_1) + L_{in}(\sigma_2), L_{out}(\sigma_1) + L_{max}(\sigma_2)) \tag{3.27}$$

The difference of $D(\cdot)$, $V_1(\cdot)$, and $V_2(\cdot)$ in evaluation function values between the original route R and modified route R' can be computed using the values of D, T_v, and L_{max} by Equation (3.28) in constant time. It's important to note that multiple matrices must be maintained for each route to store the values of these metrics. These matrices can be updated promptly whenever a move alters the corresponding route.

$$\begin{aligned}\Delta\mathcal{F}(R', R) = {} & D(R') - D(R) \\ & + \lambda_1 \frac{\sum c_k}{\sum t_k}(T_v(R') - T_v(R)) \\ & + \lambda_2 \frac{2 \cdot \max c_k - \min c_k}{\mathcal{Q}} \\ & (\max(L_{max}(R') - \mathcal{Q}, 0) - \max(L_{max}(R) - \mathcal{Q}, 0))\end{aligned} \tag{3.28}$$

3.2.4 Adaptive route-inheritance crossover

We propose an adaptive route-inheritance crossover (ARIX) for VRPSPDTW, which relies on inheriting routes from parent solutions and employs feasible and infeasible insertion operators to complete the construction of each new offspring solution. To decide the

number of parent solutions and the insertion operators to apply, a learning-based adaptive mechanism is embedded to make the best possible decisions (see Section 3.2.5).

Algorithm 3 Adaptive Route-Inheritance Crossover

1: **Input**: Population $\mathcal{P}$.
2: **Output**: Offspring solution S'.
3: $S' \leftarrow \emptyset$ /* Initialize offspring solution */
4: Determine the number of parents N_p, the insertion operator $\boxdot$ based on the adaptive strategy /* Section 3.2.5 */
5: Randomly select N_p solutions from the population $\mathcal{P}$ as parents P
6: $P_d \leftarrow P[0]$ /* Dominant parent */
7: $\mathcal{R}_d \leftarrow \forall R \in P[0]$ /* Dominant routes */
8: $\mathcal{R}_p \leftarrow \forall R \in P[1:]$ /* Routes from other parents form the route pool */
9: Determine the retention ratio γ_r
10: $N_r \leftarrow \lfloor \gamma_r \times |P_d| \rfloor$ /* Number of retained routes from the dominant routes */
11: $N_i \leftarrow |P_d| - N_r$ /* Number of introduced routes from the route pool */
12: **while** route number of S' is smaller than the route number of P_d **do**
13: **if** $N_r > 0$ **then**
14: Select the route with the lowest conflict rate from $\mathcal{R}_d$ and add it to S'
15: $N_r \leftarrow N_r - 1$
16: **end if**
17: **if** $N_i > 0$ **then**
18: Select the route with the lowest conflict rate from $\mathcal{R}_p$ and add it to S'
19: $N_i \leftarrow N_i - 1$
20: **end if**
21: **end while**
22: Remove redundant nodes in S'
23: $U \leftarrow \{n \in \mathcal{V} | n \notin S'\}$ /* Unrouted nodes */
24: Insert the unrouted nodes U into S' using the insertion operator $\boxdot$
25: **return** S'

The ARIX crossover (see Algorithm 3) starts by determining the number of parents N_p (≥ 2) and the insertion operator $\boxdot$ based on the adaptive strategy (line 4). Then, ARIX randomly selects N_p parent solutions from the population $\mathcal{P}$ (line 5), designating the first parent as the dominant parent P_d, and gathering its routes in $\mathcal{R}d$ (lines 6-7). Routes from the other parents are pooled in $\mathcal{R}p$ (line 8). The retention ratio γ_r is then determined (line 9), which is dynamically adjusted during the search process and influences the number of routes retained from P_d and the number of introduced routes N_i (lines 10-11). ARIX then iteratively constructs the offspring solution S' by selecting routes from $\mathcal{R}d$ and $\mathcal{R}p$ based on N_r and N_i (lines 12-21). Routes with the lowest conflict rate, representing the ratio

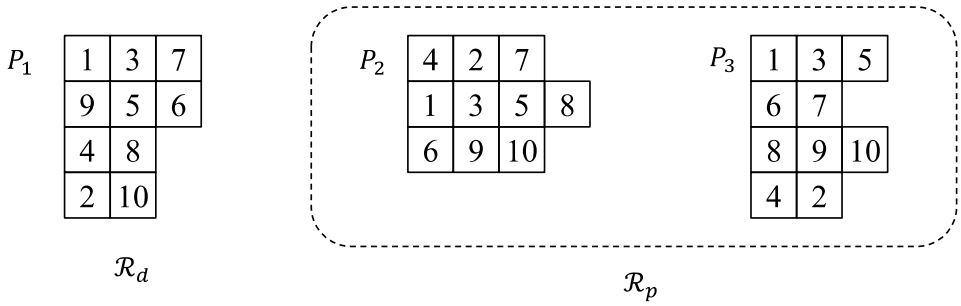
of nodes in S' to the total nodes in the route, are prioritized for selection. This process continues until the number of routes in S' reaches that of P_d (line 12). Redundant nodes in S' are subsequently removed based on selection order (line 22), and unrouted nodes are inserted using insertion operator $\boxdot$ to obtain the final offspring (line 24). If the insertion leads to infeasibility, FIRD is employed to establish the feasibility.

Five insertion operators are designed for the insertion of unrouted nodes. Feasible best insertion (FBI) and infeasible best insertion (IBI) prioritize the minimal cost increase, with FBI creating a new route if no feasible insertion is possible, while IBI accepts infeasible insertions without introducing new routes. Feasible regret insertion (FRI) and infeasible regret insertion (IRI) evaluate each insertion based on regret value, defined as the difference in travel cost between the best and second-best insertions. Random insertion (RI) inserts an unrouted customer randomly into a random route.

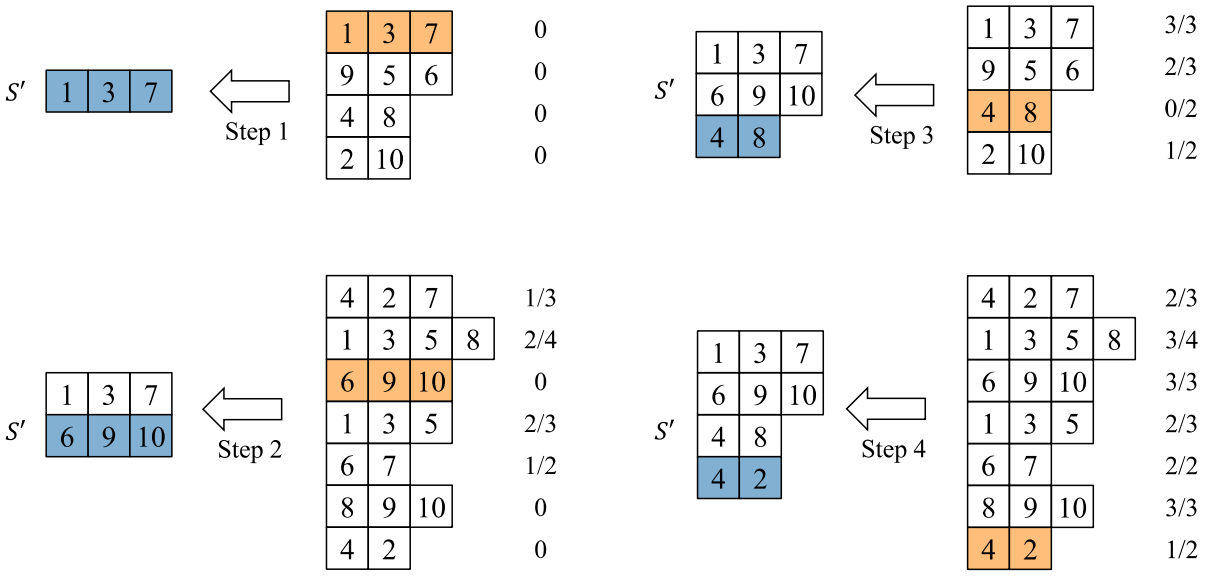
Figure 3.3 illustrates the ARIX process with three parent solutions, P_1, P_2, and P_3 (Figure 3.3a), where P_1 is the dominant parent P_d, whose routes form the dominant routes $\mathcal{R}_d$. The routes from P_2 and P_3 form the route pool $\mathcal{R}_p$. Given that P_d has four routes and the retention ratio γ_r is set to 0.5 in this example, ARIX retains two routes from $\mathcal{R}_d$ and selects two routes from $\mathcal{R}_p$. Next (Figure 3.3b), the inheritance process begins. Each route is marked with its conflict rate. The offspring solution S' is constructed by alternately selecting routes with the lowest conflict rates from $\mathcal{R}_d$ and $\mathcal{R}_p$ until the number of routes in S' matches that of the dominant parent P_d. Finally (Figure 3.3c), the redundant node 4 is removed from S', and the unrouted node 5 is inserted using the selected insertion operator, resulting in the final offspring solution S'.

3.2.5 Learning-based adaptive strategy for the ARIX crossover

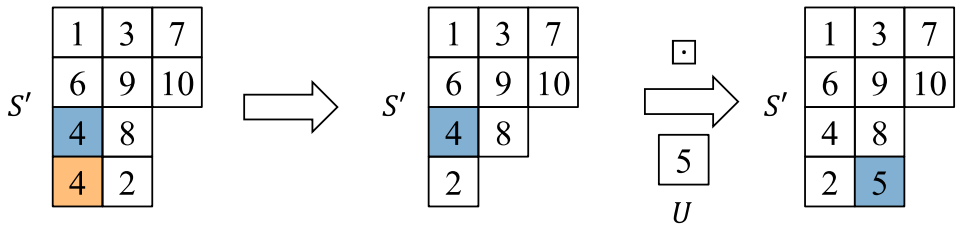
During the ARIX crossover, ARIX faces two main decision-making problems to decide the number of parents N_p used to generate offspring solutions and the insertion operator $\boxdot$ used to insert the unrouted nodes in the last step of the crossover process. To help the ARIX crossover to make the best possible decisions for these choices, we employ an adaptive strategy based on multi-armed bandit (MAB) [VM05]. Specifically, we use the UCB1 algorithm from the Upper Confidence Bound (UCB) family of algorithms [KP14]. For a set of actions $a_i \in \mathcal{A}$, UCB1 calculates the upper confidence bound of the estimated reward for each action a_i at step t using Equation (3.29). $\bar{R}_i^d$ is the empirical mean reward of action a_i, N_i is the number of times action a_i has been selected so far. The action a_i with the maximum value of $UCB1(i,t)$ is selected at each step.



(a) Parent solutions and their routes.



(b) Inheritance from the dominant routes $\mathcal{R}_d$ and the route pool $\mathcal{R}_p$.



(c) Redundant nodes removal and unrouted nodes insertion.

Figure 3.3 – Illustration of the ARIX crossover.

$$UCB1(i,t) = \bar{R}_i^d + \sqrt{\frac{2 \cdot \ln(t)}{N_i}} \tag{3.29}$$

In our context, we define the set of actions $\mathcal{A}$ as the set $\{2, ..., |\mathcal{P}|\}$ of all possible N_p parents, and the set $\{\text{FBI}, \text{IBI}, \text{FRI}, \text{IRI}, \text{RI}\}$ of all possible insertion operators I. The reward r of the action a is defined as the improvement of the objective function value of the offspring S' from the action a compared with the dominant parent P_d.

3.2.6 Population management

Following [SS06; PHK10], we use a fitness-distance based population management strategy, which relies on the extended fitness function ψ defined in Equation (3.30). This function is derived from the max-min normalization of the objective value and the distance within the population. The coefficient ξ adjusts the balance between the effects of the objective value and the distance, with higher fitness values indicating superior solution fitness. Furthermore, Equation (3.31) quantifies the distance of a given solution S from the population $\mathcal{P}$, while Equation (3.32) measures the dissimilarity between two solutions. $\mathcal{E}_{S_1}$ and $\mathcal{E}_{S_2}$ represent the sets of edges in solutions S_1 and S_2, respectively.

$$\psi(S) = \frac{\max f - f(S)}{\max f - \min f} + \xi \cdot \frac{D_{\mathcal{P}}(S) - \min D_{\mathcal{P}}}{\max D_{\mathcal{P}} - \min D_{\mathcal{P}}} \tag{3.30}$$

$$\begin{aligned} \text{Subject to} \quad \max f &= \max_{S_i \in \mathcal{P}} f(S_i), \quad \min f = \min_{S_i \in \mathcal{P}} f(S_i) \\ \max D_{\mathcal{P}} &= \max_{S_i \in \mathcal{P}} D_{\mathcal{P}}(S_i), \quad \min D_{\mathcal{P}} = \min_{S_i \in \mathcal{P}} D_{\mathcal{P}}(S_i) \end{aligned}$$

$$D_{\mathcal{P}}(S) = \min_{S_i \in \mathcal{P}} D_S(S, S_i), \quad S_i \neq S \tag{3.31}$$

$$D_S(S_1, S_2) = 100\% \left(1 - \frac{|\mathcal{E}_{S_1} \cap \mathcal{E}_{S_2}|}{\max(|\mathcal{E}_{S_1}|, |\mathcal{E}_{S_2}|)}\right), \quad \mathcal{E}_{S_1}, \mathcal{E}_{S_2} \in \mathcal{E} \tag{3.32}$$

During the execution of the algorithm, the population is expanded for $\mathcal{N}_{\mathcal{P}}/2$ generations before being reduced to its original size $\mathcal{N}_{\mathcal{P}}$. This resizing involves eliminating solutions with the worst ψ values, thereby maintaining an adaptive balance in the population composition.

3.2.7 Discussion

The proposed algorithm has a number of novel features compared to existing studies on VRPSPDTW, particularly in terms of local optimization and crossover.

For local optimization, unlike the complex large neighborhood search techniques used in recent studies [HS19; LTY21; WG23], we employ a lightweight feasible and infeasible search with simple local search operators. Key distinctions include the management of penalty terms, the infeasibility-allowed constant-time move evaluation, and the route descent strategy. Unlike penalty design in ALNS-PR [HS19], we normalize penalty terms and dynamically adjust coefficients to balance the exploration of feasible and infeasible solutions for two constraints. Compared to the evaluation method in MATE [LTY21], which focuses only on feasible solutions, our method accommodates both feasible and infeasible solutions and facilitates the calculation of violation values. Additionally, unlike VNS-BSTS [Shi+20], which handles optimization objectives in two stages, our algorithm simultaneously optimizes the number of vehicles and the travel distance by integrating the route descent strategy into the feasible and infeasible search.

For crossover, although our ARIX crossover follows the similar idea of inheriting routes from parent solutions, ARIX differs from those in [WC12; LTY21] by using multiple parents and insertion operators that are adaptively determined by a learning mechanism to create more diversified and promising offspring solutions. In addition, ARIX strategically evaluates and selects the inherited routes based on the conflict rate instead of the random selection to improve the offspring quality.

In summary, we present a novel and effective approach to solve VRPSPDTW. Its strategies and operators can also be extended to other VRPs. The route descent strategy could be useful for VRP variants with a flexible number of vehicles. The management of penalty terms and the constant-time move evaluation can be applied to VRPs with complex constraints. The idea of our adaptive route-inheritance crossover, leveraging a learning-based strategy, could be extended to various VRPs due to its general nature.

3.3 Computational results

In this section, we show computational experiments to evaluate the MA-FIRD algorithm and comparisons with the state-of-the-art algorithms based on two VRPSPDTW benchmarks, WC and JD, which are introduced in Section 1.2.3.

3.3.1 Experimental protocol and parameters tuning

The MA-FIRD algorithm was coded in C++. and run on a computer with an AMD EPYC 7282 2.8 GHz processor and 4 GB RAM, running Linux.

To balance solution quality and computational efficiency, we empirically determined several parameters. The population size $\mathcal{N}_{\mathcal{P}}$ was set to 10, the maximum number of generations φ_{max} was set to 5000, and the patience for stagnation generations ρ was set to 500. Additionally, the algorithm relies on three critical parameters: the coefficient of the fitness function ξ, the coefficient adjustment factor κ, and the initial retention ratio γ_r^0. To find a suitable setting for these three parameters, we conducted an exhaustive parameter sensitivity analysis to evaluate their impact on performance and calibrate them for best results. The parameters are varied within the ranges: $\xi \in \{0.1, 0.2, \ldots, 0.9\}$, $\kappa \in \{0.1, 0.2, \ldots, 0.9\}$, and $\gamma_r^0 \in \{0.1, 0.2, \ldots, 0.9\}$. The best parameter settings were summarized in Table 3.1. These values are consistently used in the experiments.

Table 3.1 – Parameters tuning results.

Parameter	Description	Value range	Final value
ξ	coefficient of the fitness function	(0, 1)	0.8
κ	coefficient adjustment factor	(0, 1)	0.5
γ_r^0	initial retention ratio	(0, 1)	0.9

To ensure a fair comparison, the maximum running time τ for the JD benchmark is set to 7200 seconds following [LTY21]. Each instance was solved with 30 independent runs of the algorithm.

3.3.2 Reference algorithms

For the WC instances, we compare our results with the best-known solutions (BKS) ever reported in the literature and with the top performing algorithms ALNS-PR [HS19] and MATE [LTY21]. These two algorithms outperformed the other algorithms, and together produced the current best-known solutions for the WC instances. For the JD instances, we focus on a comparative study with MATE [LTY21], which tested these 20 large-scale instances. Since the code of MATE is open source, we reran it on our computer with the same experimental protocol as for our algorithm and the results are indicated by MATE*.

The experimental environments of the reference algorithms are as follows. ALNS-PR [HS19] was programmed in Java and performed on a Windows 10 Professional desktop

computer with an Inter Core i5-6600 3.30 GHz processor and 16 GB RAM. MATE [LTY21] was programmed in C++ and executed on a machine with an Intel Xeon E5-2699A V4 2.40 GHz and 128 GB RAM, running Centos 7.5.

Table 3.2 – CPU information and time conversion ratio γ.

Algorithm	Processor	Base frequency	CPU mark	γ
MA-FIRD	AMD EPYC 7282	2.80 GHz	1829	1.00
ALNS-PR [HS19]	Intel Core i5-6600	3.30 GHz	2260	1.24
MATE [LTY21]	Intel Xeon E5-2699A	2.40 GHz	1945	1.06

To account for different experimental environments in the proposed algorithm and reference algorithms, we apply a time conversion ratio γ to standardize the computation time. This ratio is defined as the ratio of CPU scores obtained from PassMark [1], a recognized CPU benchmarking platform. Table 3.2 presents the detailed information about the CPU used in our experiments and those used by the reference algorithms, and the calculated time conversion ratio γ.

3.3.3 Computational results and comparisons

The comparative results on the two benchmark sets are summarized in Table 3.3, which provides a global view of the performance of the proposed algorithm across different benchmark groups. These groups are identified as WC-S, WC-C, WC-R, WC-RC, and JD, corresponding to the small WC, clustered WC, random WC, clustered-random WC instances, and JD instances, respectively. The row *#Instances* gives the number of instances within each group. The table contains information about the instances where the proposed algorithm shows *better-same-worse* results compared to the BKS and the reference algorithms. The '–' symbol indicates the unavailability of the results. In addition, the Wilcoxon signed-rank test [Wil92] with a confidence level of 0.05 was conducted to ascertain the statistical significance of the results, and the corresponding *p-values* are given in the table.

Table 3.3 shows that the MA-FIRD algorithm consistently provides comparable or superior results compared to the BKS and the reference algorithms. For the WC benchmark, MA-FIRD achieved 8 new best upper bounds and reached the BKS values for all remaining instances. For the JD benchmark, MA-FIRD established new best upper bounds for all 20 instances. The associated *p-values* ($<< 0.05$) indicates that the proposed algorithm statistically performs better than the BKS and reference algorithms.

1. `https://www.passmark.com/`

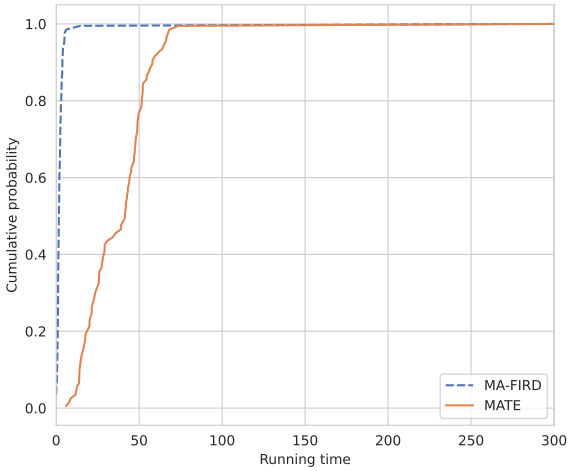
Table 3.3 – Summary of comparative results on benchmark instances between MA-FIRD and the references.

Group	WC-S	WC-C	WC-R	WC-RC	JD	Total	*p-value*
#Instances	9	17	23	16	20	85	
MA-FIRD vs BKS	0-9-0	0-17-0	5-18-0	3-13-0	20-0-0	28-57-0	3.79E-06
MA-FIRD vs ALNS-PR [HS19]	-	3-14-0	11-12-0	10-6-0	-	24-41-0	1.82E-05
MA-FIRD vs MATE [LTY21]	0-9-0	2-15-0	10-13-0	5-11-0	20-0-0	37-48-0	8.39E-08
MA-FIRD vs MATE*	0-9-0	2-15-0	11-12-0	6-10-0	20-0-0	39-46-0	9.56E-09

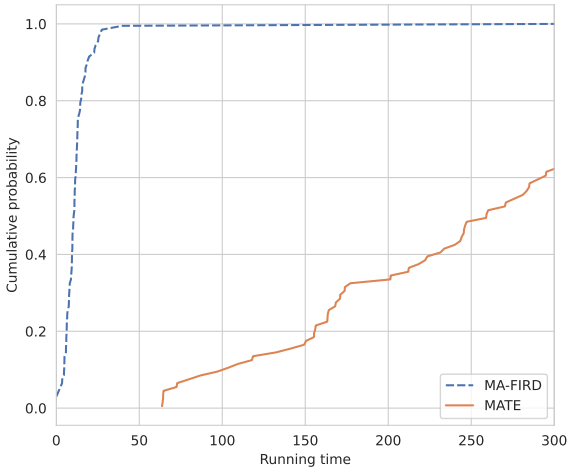
The detailed computational results are provided in Tables A.1, A.2 and A.3 in the appendix. For the 9 small WC instances, as shown in Table A.1, despite small differences in computation time, all methods solve these instances consistently and reliably within a short time. These instances do not represent a challenge for the compared algorithms.

Table A.2 shows that the MA-FIRD algorithm outperforms the reference algorithms for 56 medium WC instances. It reaches the BKS values for 48 instances and improves the best-known results for the 8 remaining instances (indicated in bold). Notably, within these 8 instances, MA-FIRD obtains a new best upper bound for the instance *Rdp104* with a smaller M value (with 9 vehicles instead of 10 for the BKS). MA-FIRD performs consistently and stably with small, and even zero, standard deviations for the number of vehicles and the travel distance for most instances. For the primary objective M (the number of vehicles), MA-FIRD has a better mean value of 7.27 against 7.30 for ALNS-PR, 7.36 for MATE, and 7.45 for MATE*. In terms of the gap to the BKS, MA-FIRD holds the best gap of -0.19% (a negative gap, indicating an improved result) against 0.14% for ALNS-PR, 2.00% for MATE, and 2.79% for MATE*. In terms of the running time, all algorithms perform similarly, except for MATE*, which requires a longer running time.

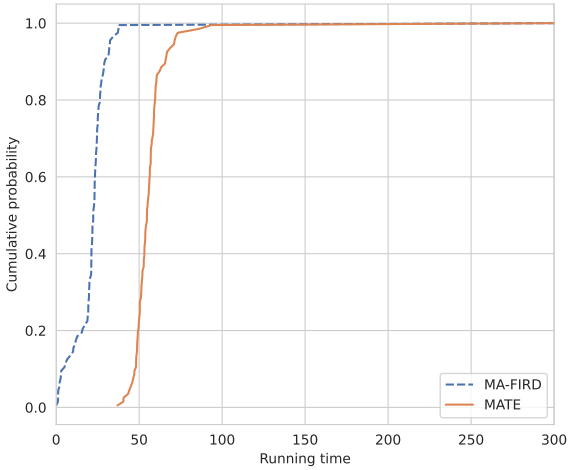
The results on the 20 large-scale JD benchmark instances in Table A.3 clearly demonstrate that the MA-FIRD algorithm consistently outperforms both MATE and MATE* with significant improvements in all instances in terms of the total cost, with a negative mean gap of -2.28% for average results. Even the average objective value is better than the best reference value (144677.53 against 179464.90). If we examine the number of vehicles and the travel distances separately, we observe that MA-FIRD dominates MATE and MATE* in both criteria. These results demonstrate the effectiveness of the proposed algorithm in solving large instances in real-world scenarios.



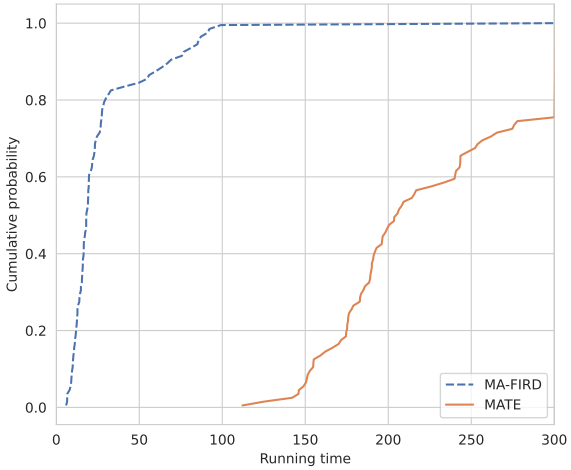
(a) TTT value of *Cdp101*.



(b) TTT value of *Rdp203*.



(c) TTT value of *RCdp101*.



(d) TTT value of *RCdp204*.

Figure 3.4 – Examples of time-to-target analysis between MA-FIRD and MATE. The x-axis represents the running time to reach the target value and the y-axis indicates the cumulative probability of reaching the given target value.

Time-to-target analysis

To further compare the computational efficiency between the proposed MA-FIRD algorithm and the MATE algorithm, we perform a time-to-target (TTT) analysis [ARR07; RRV12] on the medium WC instances. This analysis involves solving each instance 100 times for both algorithms, measuring the probability distribution of the required time to achieve a predefined target objective value. In this analysis, the maximum running time per run is set to 300 seconds.

Figure 3.4 visually depicts the results for four representative instances *Cdp101*, *Rdp203*, *RCdp101*, and *RCdp204*. The results confirm that the proposed algorithm consistently achieves the given targets within a short running time, indicating its remarkable convergence speed and computational efficiency.

3.4 Performance analysis

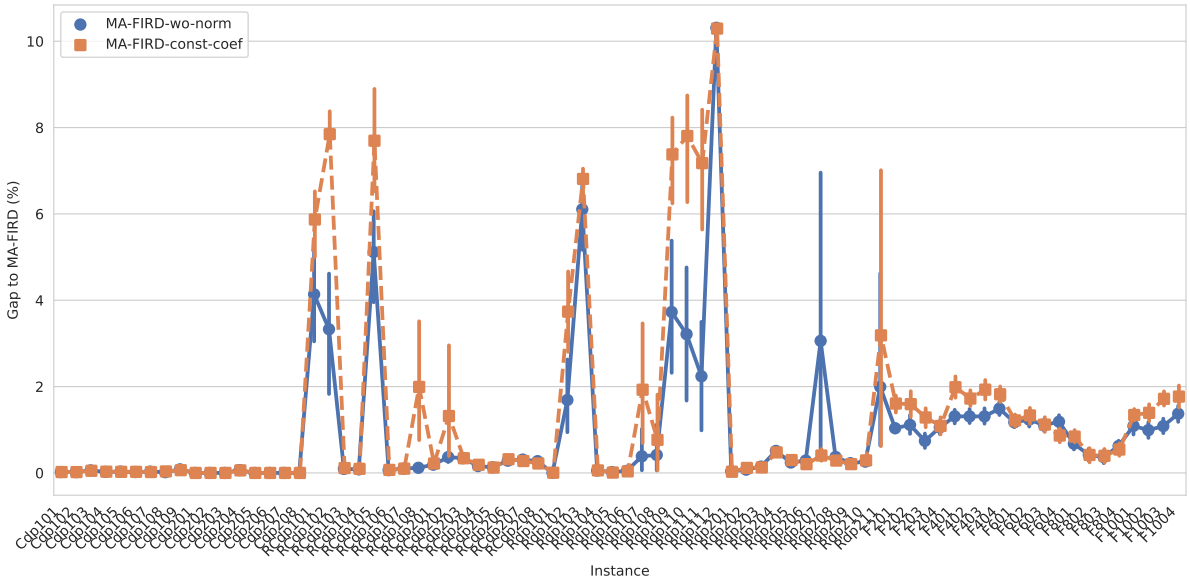
We now present experiments to study the influences of the algorithmic components based on the medium WC instances and the large JD instances.

3.4.1 Analysis on the feasible and infeasible route descent search

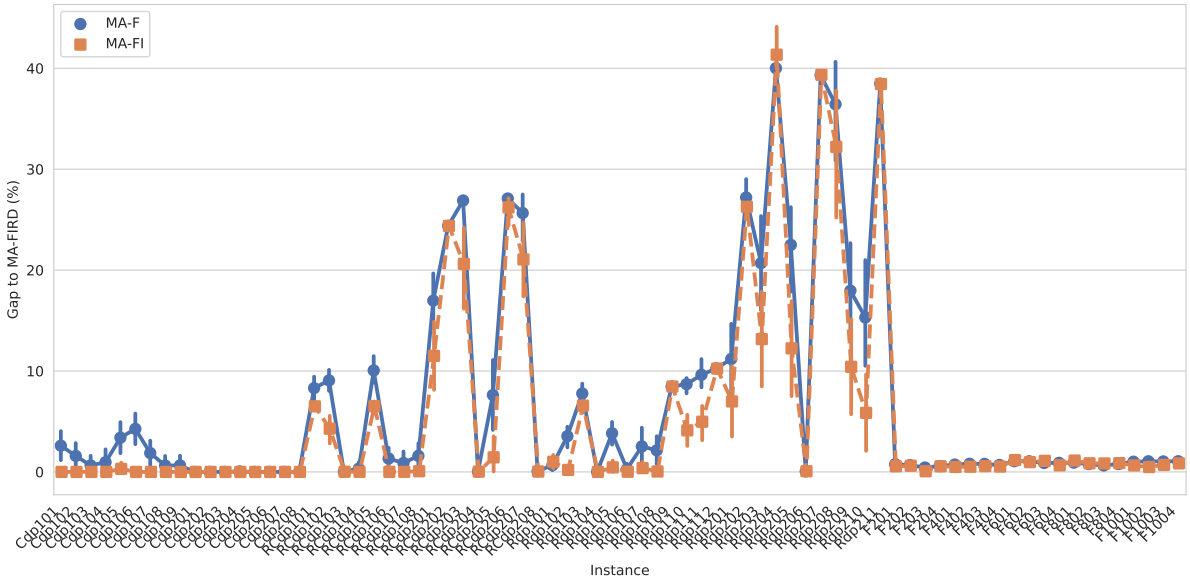
First, we investigate the impact of the penalty term management in the feasible and infeasible search, focusing on normalization and dynamic coefficient adjustment. We compare the MA-FIRD algorithm with two variants: MA-FIRD-wo-norm, which removes the penalty term normalization, and MA-FIRD-const-coef, which sets the penalty coefficients to a constant value ($\lambda_1 = \lambda_2 = 1.5$). Figure 3.5a shows the relative gap between the solutions obtained by these variants and those of MA-FIRD. The results indicate that normalization of penalty terms and dynamic coefficient adjustment significantly enhance performance as the variants perform much worse than the baseline MA-FIRD in most instances.

Next, we assess the role of the feasible and infeasible route descent search by comparing it with the variant MA-F, which eliminates the route descent strategy and accepts only feasible solutions, and the variant MA-FI, which also removes the route descent strategy but allows infeasible solutions. Figure 3.5b compares MA-F and MA-FI with the baseline MA-FIRD. MA-FI shows slightly better performance than MA-F, suggesting that accepting infeasible solutions can help escape feasible local optima and improve search

capability. However, both variants perform significantly worse than MA-FIRD in almost all instances, highlighting the importance of the route descent strategy.



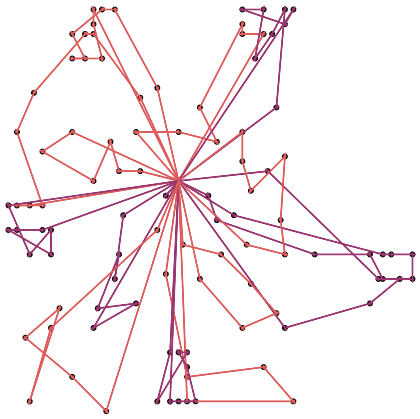
(a) Comparison of MA-FIRD with MA-FIRD-wo-norm and MA-FIRD-const-coef.



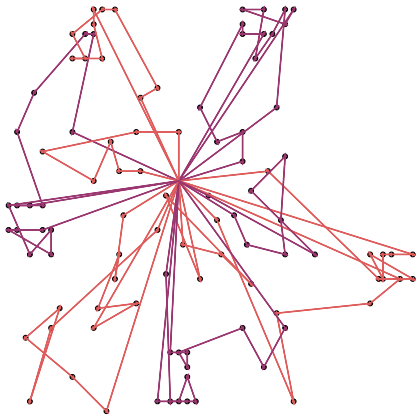
(b) Comparison of MA-FIRD with MA-F and MA-FI.

Figure 3.5 – Comparison of MA-FIRD with its variants. The x-axis denotes the instance names and the y-axis illustrates the relative gap between the solutions obtained by variants compared to those of MA-FIRD. A positive gap indicates that the variant performs worse than MA-FIRD, and vice versa.

To better understand the benefits of the route descent strategy, figures 3.6 and 3.7

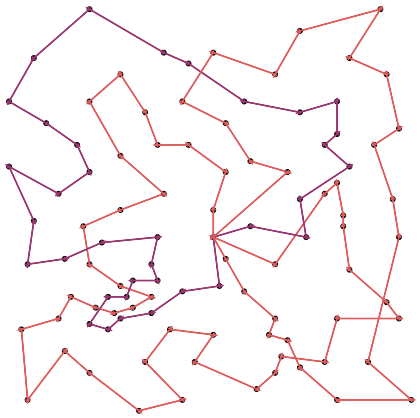


(a) $M = 15$ and $f(S) = 1634.85$.

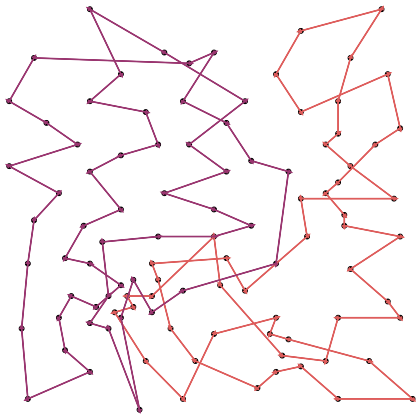


(b) $M = 14$ and $f(S) = 1708.21$.

Figure 3.6 – Two solutions on the instance *RCdp101* with the distance of 32.17.



(a) $M = 3$ and $f(S) = 767.82$.



(b) $M = 2$ and $f(S) = 885.71$.

Figure 3.7 – Two solutions on the instance *Rdp211* with the distance of 72.82.

illustrate high-quality solutions for adjacent numbers of vehicles in two representative instances. Different colors represent distinct routes. One notices that high-quality solutions for adjacent numbers of vehicles are not necessarily close and may show significant differences. For example, the distance between two solutions for *RCdp101* is 32.17, and for *Rdp211*, it is 72.82. This indicates significant dissimilarity between solutions with adjacent numbers of vehicles. This observation highlights that the distribution of solutions in the search space is not uniform, and high-quality solutions for different numbers of vehicles may not be proximate. The route descent strategy enables rapid traversal of the search space, helping to locate high-quality solutions for varying numbers of vehicles.

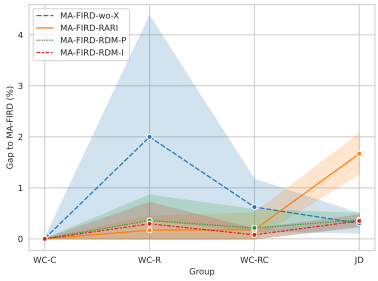
3.4.2 Analysis on the adaptive route-inheritance crossover

To study the impact of the ARIX crossover, we create the following variants of the MA-FIRD algorithm:

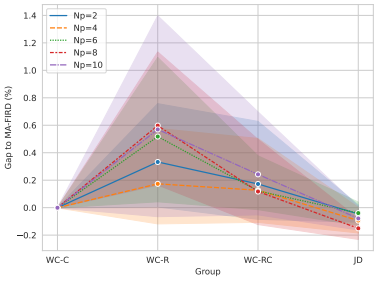
- MA-FIRD-wo-X: ARIX is removed, and only one solution is maintained in the population. At each generation, the correlated destroy and repair operator from [LTY21] is used to perturb the solution, which is then improved by the FIRD search to generate a new solution.
- MA-FIRD-RARI: ARIX is replaced by the RARI crossover from [LTY21].
- MA-FIRD-RDM-P: The number of parents is randomly chosen from $\{2, ..., |\mathcal{P}|\}$.
- MA-FIRD-RDM-I: The insertion operator is randomly selected from the set {FBI, IBI, FRI, IRI, RI} at each generation.

Figure 3.8a provides a visual comparison of the variants against MA-FIRD. The instances are grouped into WC-C, WC-R, WC-RC, and JD. Note that the WC-C instances are less sensitive to configuration variations, while instances in the other groups are more sensitive. In general, the baseline MA-FIRD algorithm outperforms MA-FIRD-wo-X and MA-FIRD-RARI in most cases, which confirms the importance of ARIX in the algorithm. Furthermore, the results of MA-FIRD-RDM-P and MA-FIRD-RDM-I lag behind MA-FIRD, suggesting that the learning-based adaptive mechanism helps ARIX to select the appropriate number of parents and insertion operators for different instances.

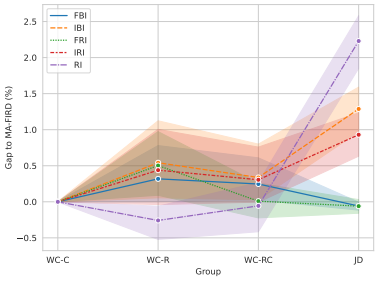
To gain deeper insights into the impact of different ARIX configurations, we conducted comprehensive experiments analyzing the influence of the number of parents and the insertion operations. Figures 3.8b and 3.8c present the visualized results. Figure 3.8b shows the results for different numbers of parents ($N_p = 2, 4, 6, 8, 10$), while Figure 3.8c presents the results for different insertion operators (FBI, IBI, FRI, IRI, and RI).



(a) Comparison of the MA-FIRD and its variants.



(b) Results with number of parents $N_p = 2, 4, 6, 8, 10$.



(c) Results with insertion operations FBI, IBI, FRI, IRI, and RI.

Figure 3.8 – Comparison of the MA-FIRD algorithm and its variants. The x-axis denotes the instance groups and the y-axis illustrates the relative gap between the solutions obtained by variants compared to those of MA-FIRD. The mean relative gap for each group is indicated by a solid line, while the 0.95 confidence interval is represented by the corresponding shadow.

Overall, instances in the WC-C group still demonstrate less sensitivity to variations in setups. However, performance across the other three groups varies with different configurations. Notably, no single configuration consistently performs well across all instances. For example, $N_p = 6, 8, 10$ perform well on JD but poorly on WC-R, while certain insertion operators (IBI, IRI, and RI) perform well on WC-R but poorly on JD. This suggests the necessity of an adaptive mechanism to dynamically determine configurations and values. Additionally, these configurations generally do not perform better than the baseline MA-FIRD, indicating the efficacy of the learning-based adaptive mechanism.

3.5 Chapter conclusion

We introduce an effective memetic algorithm to solve the challenging VRPSPDTW. Based on the memetic algorithm framework, our algorithm integrates several innovative strategies to effectively enhance the exploration of the solution space. These include a lightweight feasible and infeasible search mechanism featuring the route descent strategy, penalty term management, and infeasibility-allowed constant-time move evaluation. We also introduce a learning-based adaptive route-inheritance crossover to enhance generalization and robustness, and a fitness-distance population management strategy based on max-min normalization to maintain population diversity.

Extensive experiments have been conducted to evaluate the algorithm's performance. The results show that the proposed algorithm consistently outperforms the reference algorithms with high computational efficiency. Remarkably, The algorithm attains the best

solutions for all benchmark instances, reaching 57 best-known solutions (BKS) and achieving 8 new best upper bounds for the 65 popular benchmark instances. Furthermore, the algorithm achieves new best upper bounds for all 20 large real-world instances, demonstrating its effectiveness and practicality in real-world scenarios.

The flexibility and versatility of innovative strategies in the proposed algorithm lay the basis for future extensions to tackle other multi-attribute Vehicle Routing Problems. Future work aims to explore the adaptability and performance of the algorithm in a broader range of complex routing scenarios, thus contributing to the advancement of the field.

Chapter 4

Speeding up local optimization in vehicle routing with tensor-based GPU acceleration

Local search is central to many high-performing metaheuristic algorithms for the vehicle routing problem (VRP) and its variants. However, neighborhood exploration is computationally expensive, especially for large or highly constrained instances. This chapter addresses this challenge by proposing an original Tensor-based GPU Acceleration (TGA) framework with two implementations. TGA accelerates move evaluation for common local search operators through an attribute-based solution tensor representation and fully tensorized operator implementations that exploit GPU parallelism. The framework is extensible to a wide range of VRP variants and can be seamlessly integrated into local search-based algorithms. Extensive experiments on three routing problems demonstrate significant advantages over traditional CPU-based implementations. A detailed analysis further reveals performance characteristics, potential bottlenecks, and directions for future improvement. The content of this chapter is based on an article submitted to *Transportation Science*.

4.1 Introduction

The literature review in Chapter 2 has demonstrated the potential of GPU acceleration for solving VRPs, but several limitations remain. Most methods focus on basic VRP variants or simple operators, with limited support for complex constraints and diverse problem settings. Moreover, current approaches rely heavily on low-level CUDA implementations. While powerful, CUDA poses a high barrier to entry due to its steep learning curve and the need for manual management of data transfers, memory layouts, parallel execution, and thread synchronization. This makes widespread adoption challenging, particularly for researchers in operations research without specialized GPU expertise.

To address these gaps, we propose a fine-grained Tensor-based GPU Acceleration (TGA) framework to improve the computational efficiency of local search-based algorithms for complex VRPs. The main contributions are: (1) The TGA framework, along with two implementations, is proposed to accelerate move evaluation by tensorizing local search operators based on an attribute-based solution tensor representation, thereby fully leveraging GPU parallelism. (2) TGA is highly extensible, supporting a wide range of VRP variants and integration into different local search solvers. (3) Unlike CUDA-based approaches, TGA can be implemented using high-level tensor libraries such as PyTorch, substantially lowering the barrier to GPU acceleration for operations research researchers. (4) In-depth theoretical and experimental analyses provide insights into performance characteristics and potential bottlenecks, offering guidance for future research.

To demonstrate its performance and extensibility, we apply TGA to three representative VRPs: CVRP, VRP with Time Windows (VRPTW), and VRP with Simultaneous Pickup and Delivery and Time Windows (VRPSPDTW). Results on benchmark instances confirm that the TGA effectively handles diverse constraints and achieves substantial efficiency gains over traditional CPU-based implementations.

The rest of this chapter is organized as follows. Section 4.2 presents an overview of GPU architecture and tensor computation, forming the basis of the proposed framework. Section 4.3 introduces essential background on attribute matrices and sequence concatenation-based move evaluation. Section 4.4 presents the tensor-based GPU acceleration framework. Section 4.5 reports computational results on benchmark instances and provides an in-depth analysis of the framework's advantages and limitations. Section 4.6 discusses its extensibility and limitations, while Section 4.7 gives concluding comments and suggests possible directions for future research.

4.2 GPU and tensor computation

This section recalls background on GPU architecture, CUDA programming, and tensor computation, forming the basis of our acceleration method.

4.2.1 GPU architecture

Originally designed for graphics rendering, GPUs are specialized parallel processors that are now widely used for general-purpose computing in scientific simulations, data analytics, and artificial intelligence. Unlike CPUs, which prioritize low-latency execution of sequential instructions through complex control logic and extensive caching, GPUs dedicate most of resources to arithmetic units, enabling thousands of lightweight threads to execute concurrently.

NVIDIA GPUs dominate high-performance and scientific computing due to their performance and mature software ecosystem. The fundamental unit of NVIDIA GPUs, the *Streaming Multiprocessor* (SM), integrates numerous CUDA cores, registers, shared memory, and caches, enabling thousands of threads to cooperate efficiently. Execution on NVIDIA GPUs follows the SIMT (Single Instruction, Multiple Threads) model, where threads are organized into 32-thread *warps* that execute the same instruction on different data, simplifying scheduling and enabling scalable fine-grained parallelism. Additionally, the GPU memory hierarchy supports high-throughput workloads, with large high-latency global memory, fast per-SM shared memory for inter-thread communication, and low-latency per-thread registers.

Table 4.1 – Comparison of NVIDIA GPUs: V100, A100, H100.

GPU	Year	Arch.	SMs	Cores/SM	Total Cores	Memory	Memory Bandwidth	Compute Throughput (TFLOPS)
V100	2017	Volta	80	64	5,120	16/32 GB HBM2	∼900-1,134 GB/s	∼15.7 (FP32) / ∼7.8 (FP64)
A100	2020	Ampere	108	64	6,912	40/80 GB HBM2e	∼1.6-2.0 TB/s	∼19.5 (FP32) / ∼9.7 (FP64)
H100	2022	Hopper	132	128	16,896	80 GB HBM3	∼3.0-3.2 TB/s	∼67 (FP32) / ∼34 (FP64)

Table 4.1 summarizes three recent NVIDIA GPU generations, showing consistent increases in SMs, CUDA cores, memory capacity, memory bandwidth, and floating-point throughput. More SMs, cores, and higher compute throughput enhance parallelism and speed up compute-intensive workloads. Larger and faster memory supports larger datasets locally, while higher memory bandwidth enables faster data transfer between memory and compute units. Together, these improvements substantially boost modern GPUs' ability to handle large-scale, complex parallel workloads in scientific and industrial applications.

4.2.2 CUDA programming

NVIDIA's Compute Unified Device Architecture (CUDA) is the primary framework for general-purpose GPU programming on NVIDIA hardware. It provides a C/C++ based model in which developers write kernels executed in parallel by many lightweight threads. CUDA exposes hardware concepts such as threads, blocks, warps, and multiple levels of memory, offering fine-grained control over parallel execution and data movement.

Despite its flexibility, CUDA development is challenging. Programmers must explicitly manage device memory, data transfers between CPU and GPU, and thread synchronization. High performance requires deep knowledge of GPU architecture, including how to design coalesced memory accesses, avoid bank conflicts, minimize warp divergence, and tune occupancy. CUDA programming is thus inherently more time-consuming and error-prone than high-level CPU programming, limiting adoption among operations researchers who primarily focus on algorithm design or problem modeling and often lack these specialized skills.

4.2.3 Tensor computation

In scientific computing and machine learning, tensor computation has emerged as an effective paradigm for exploiting GPU acceleration. A *tensor* is a multi-dimensional array that generalizes scalars (0-D), vectors (1-D), and matrices (2-D) to higher dimensions. On GPUs, tensors are typically stored as contiguous memory blocks, enabling coalesced memory access and efficient parallel execution. Tensor operations are inherently parallel, as computations on different elements can be distributed across thousands of GPU cores, allowing tensor-based workloads to fully exploit the massive parallelism of modern GPU architectures.

High-level frameworks such as PyTorch and TensorFlow provide a rich set of tensor operations, including element-wise arithmetic, slicing, reshaping, and reduction. These frameworks automatically translate tensor operations into highly optimized GPU kernels, relieving developers from explicitly managing threads, memory transfers, and kernel launches, which are required in low-level CUDA programming. By abstracting away hardware-specific details, tensor computation enables researchers to concentrate on algorithmic design while still achieving high-performance execution.

Thus, in this study, we model the solution with tensor representation, and implement move evaluation of local search operators using tensor operations. This design enables

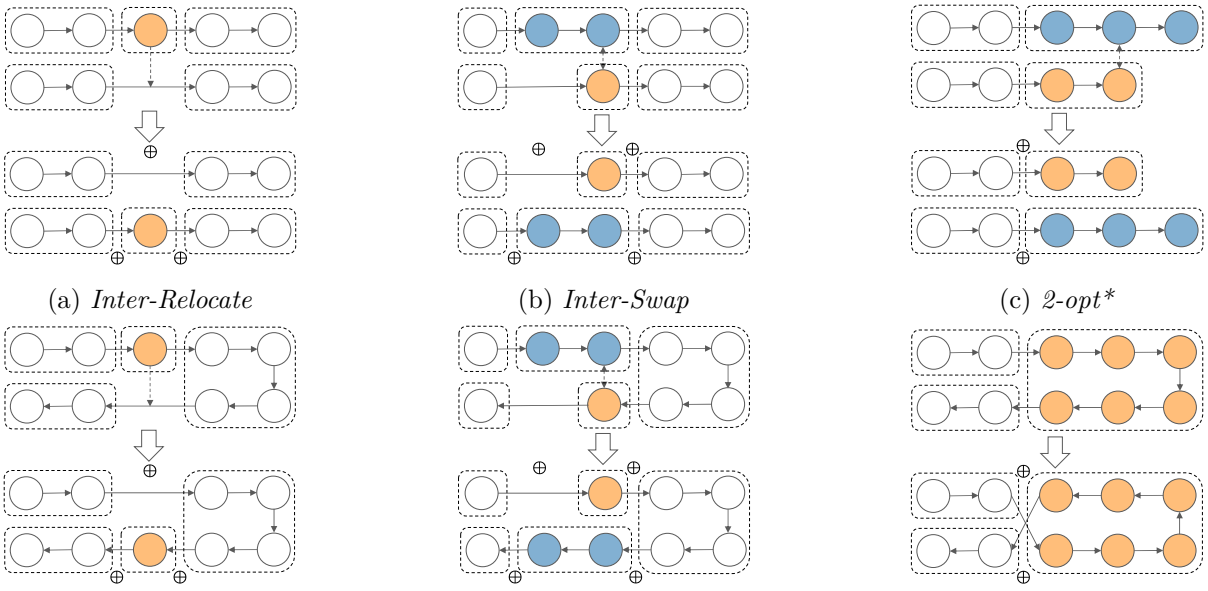
efficient exploitation of GPU parallelism through high-level tensor frameworks, making GPU acceleration more accessible and practical for researchers in operations research.

4.3 Move evaluation and attribute matrix

Efficient move evaluation is a critical step in the search process of local search algorithms and directly affects both solution quality and search efficiency. Our proposed tensor-based GPU acceleration (TGA) framework builds upon a constant-time move evaluation method [Vid+14; LTY21; LH25]. Although this method has not been explicitly named in prior works, We refer to it hereafter as *sequence concatenation-based move evaluation*.

4.3.1 Sequence concatenation-based move evaluation

Sequence concatenation-based move evaluation conceptualizes common local search move operators, such as *Relocate*, *Swap*, *2-opt**, and *2-opt*, as the concatenation of multiple route subsequences. This allows for efficient neighborhood evaluations with respect to both the objective function and the associated constraints.



(a) *Inter-Relocate* (b) *Inter-Swap* (c) *2-opt**

(d) *Intra-Relocate* (e) *Intra-Swap* (f) *2-opt*

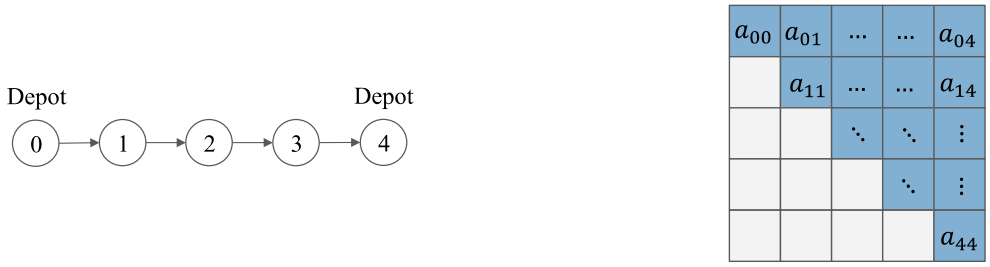
Figure 4.1 – Illustration of operators and their concatenation operations. Dotted boxes represent the route subsequences. Symbols $\oplus$ denote the concatenation operations.

The *Relocate* operator moves a sequence of nodes to a new position within the same route or a different route, while the *Swap* operator exchanges two sequences of nodes. The *2-opt** operator, applied between two routes, breaks one edge in each route, exchanges the subsequences, and reconnects them to form new routes. In contrast, the *2-opt* move operates within a single route by breaking two non-adjacent edges and reconnecting the reversed subsequence. Note that *2-opt* is only suitable for problems with symmetric distance matrices and without directed constraints, such as time windows. Based on whether the operators act within a single route or between two routes, these operators can be classified into two categories: *intra-route* operators, which act within a single route (*Intra-Relocate*, *Intra-Swap* and *2-opt*), and *inter-route* operators, which operate between two routes (*Inter-Relocate*, *Inter-Swap*, and *2-opt**). Figure 4.1 illustrates the concatenation operations for each operator. The concatenation operations can also be extended to other routing operators with similar characteristics, such as 3-opt, Or-opt, and Cross-exchange.

4.3.2 Attribute matrices

To enable efficient sequence concatenation-based move evaluation, we introduce *attribute matrices*, which store the attribute values of all feasible route subsequences. These values are precomputed during the update stage, enabling constant-time computation of concatenated sequences in the move evaluation stage. Specifically, for each route and for each relevant attribute, an *attribute matrix* is constructed to record the attribute values of all feasible subsequences. Figure 4.2 illustrates a route of length $|R| = 5$ along with its attribute matrix, where a_{ij} denotes the attribute value associated with the subsequence from the i-th node to the j-th node of the route. For example, for the travel distance attribute, a_{ij} corresponds to the cumulative travel distance between the i-th and j-th nodes. Note that only the upper triangular portion of the matrix contains valid entries. These attributes are tailored to specific problem settings and constraints, providing strong adaptability across various VRP variants.

Building upon these attribute matrices, constant-time move evaluation is achieved through sequence concatenation operations, which constitute a core component of the proposed tensor-based GPU acceleration framework.



(a) Route representation.

(b) Attribute matrix

Figure 4.2 – Illustration of a route of length $|R| = 5$ and its corresponding attribute matrix. The first (0) and last (4) nodes represent the depot.

4.3.3 Calculations of concatenation operations

This section details the calculations of concatenation operations using attribute matrices. We focus on commonly used attributes: *travel distance*, *load related attributes*, and *time window related attributes*, which are sufficient for the studied VRPs (CVRP, VRPTW, and VRPSPDTW) and widely adopted in vehicle routing. This concatenation operation can be extended to attributes required by other VRP variants, such as cumulative distance, load-dependent cost as discussed in [Vid+14].

Let σ denote a route subsequence. A single-node subsequence is denoted by $\sigma^0 = \{v_k\}$, while $\sigma_1 = \{v_i, \ldots, v_j\}$ and $\sigma_2 = \{v_k, \ldots, v_l\}$ represent two generic subsequences. The concatenation of σ_1 and σ_2 is denoted by $\sigma_1 \oplus \sigma_2$. The following subsections describe how attributes are updated under concatenation.

Travel distance

Travel distance is a fundamental attribute in routing problems. Let $D(\sigma)$ denote the travel distance of a subsequence σ. For a single-node subsequence σ^0 and the concatenation of two route subsequences σ_1 and σ_2, the travel distance is computed as follows.

$$D(\sigma^0) = 0 \tag{4.1a}$$

$$D(\sigma_1 \oplus \sigma_2) = D(\sigma_1) + c_{jk} + D(\sigma_2) \tag{4.1b}$$

where c_{jk} denotes the travel distance between the last node v_j of σ_1 and the first node v_k of σ_2.

Load and capacity

To evaluate load and capacity constraints, we maintain three load-related attributes for each route subsequence σ: the incoming load $L_I(\sigma)$, the outgoing load $L_O(\sigma)$, and the maximum load $L_M(\sigma)$. The computation of these attributes is defined as follows.

$$L_I(\sigma^0) = d_k, \quad L_O(\sigma^0) = p_k, \quad L_M(\sigma^0) = \max\{d_k, p_k\} \tag{4.2a}$$
$$L_I(\sigma_1 \oplus \sigma_2) = L_I(\sigma_1) + L_I(\sigma_2) \tag{4.2b}$$
$$L_O(\sigma_1 \oplus \sigma_2) = L_O(\sigma_1) + L_O(\sigma_2) \tag{4.2c}$$
$$L_M(\sigma_1 \oplus \sigma_2) = \max\{L_M(\sigma_1) + L_I(\sigma_2), L_O(\sigma_1) + L_M(\sigma_2)\} \tag{4.2d}$$

For CVRP and VRPTW, where pickup demands are zero ($p_k = 0$), the formulation simplifies and only the maximum load needs to be maintained:

$$L_M(\sigma^0) = d_k \tag{4.3a}$$
$$L_M(\sigma_1 \oplus \sigma_2) = L_M(\sigma_1) + L_M(\sigma_2) \tag{4.3b}$$

Time windows

Time window constraints can be handled using the time-warp technique [NBD10; Vid+13], which allows the evaluation of time window violations within subsequences. Each subsequence σ maintains the earliest and latest arrival times ($T_E(\sigma)$ and $T_L(\sigma)$), duration $T_D(\sigma)$, and time window violation $T_V(\sigma)$. Let t_{jk} denote the travel time between the last node of σ_1 and the first node of σ_2. The concatenation rules are given by:

$$T_D(\sigma^0) = s_k, \quad T_E(\sigma^0) = e_k, \quad T_L(\sigma^0) = l_k, \quad T_V(\sigma^0) = 0 \tag{4.4a}$$
$$\Delta_t = T_D(\sigma_1) + T_W(\sigma_1) + t_{jk} - T_V(\sigma_1) \tag{4.4b}$$
$$\Delta_w = \max\{T_E(\sigma_2) - \Delta_t - T_L(\sigma_1), 0\} \tag{4.4c}$$
$$\Delta_v = \max\{T_E(\sigma_1) + \Delta_t - T_L(\sigma_2), 0\} \tag{4.4d}$$
$$T_D(\sigma_1 \oplus \sigma_2) = T_D(\sigma_1) + t_{jk} + \Delta_w + T_D(\sigma_2) \tag{4.4e}$$
$$T_E(\sigma_1 \oplus \sigma_2) = \max\{T_E(\sigma_1), T_E(\sigma_2) - \Delta_t\} - \Delta_w \tag{4.4f}$$
$$T_L(\sigma_1 \oplus \sigma_2) = \min\{T_L(\sigma_1), T_L(\sigma_2) - \Delta_t\} + \Delta_v \tag{4.4g}$$
$$T_V(\sigma_1 \oplus \sigma_2) = T_V(\sigma_1) + \Delta_v + T_V(\sigma_2) \tag{4.4h}$$

By maintaining the attribute matrices, all concatenation operations can be computed in constant time, enabling efficient evaluation of attribute differences between the current solution and its neighbors within local search neighborhoods.

4.4 Tensor-based GPU acceleration for local search operators

In this section, we present the Tensor-based GPU Acceleration (TGA) framework, which is designed to improve computational efficiency of commonly used local search operators in vehicle routing problems. The core lies in the design of the attribute-based solution tensor representation, and the tensor-based implementation of local search operators. By extracting and concatenating the relevant attribute tensors from the solution tensor representation, we can efficiently implement the sequence concatenation-based move evaluation with the help of GPU parallelism.

4.4.1 Workflow of the TGA framework between CPU and GPU

The proposed TGA framework is designed with a low degree of coupling between the CPU and the GPU. The computationally intensive move evaluation of local search operators is entirely offloaded to the GPU. Meanwhile, the CPU handles only the updates based on the GPU's computation results. This design ensures that the TGA can be easily integrated into a wide range of local search-based algorithms and frameworks.

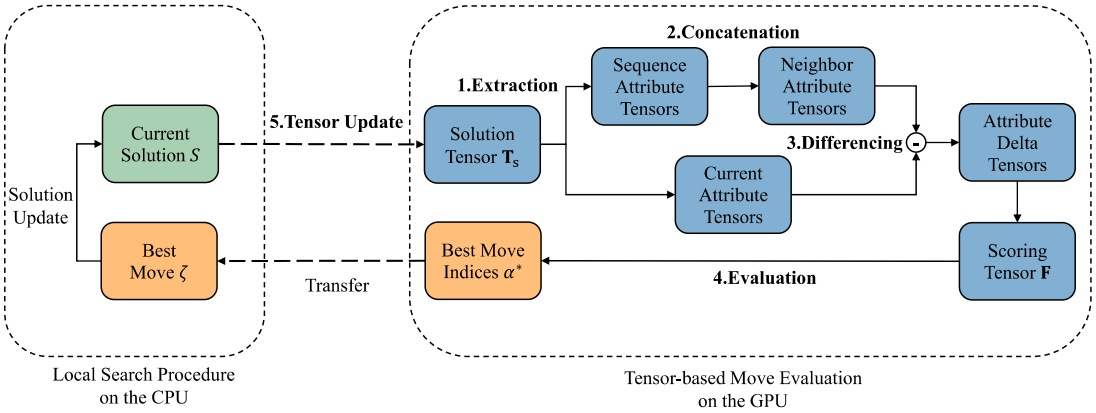


Figure 4.3 – Workflow of the TGA framework.

The workflow of the TGA framework is illustrated in Figure 4.3. For a given solution S

to be improved by local search on the CPU, a solution tensor $\mathbf{T}_s$ is first initialized on the GPU using the attribute matrices derived from S. Tensor-based move evaluation is then performed on the GPU using $\mathbf{T}_s$. To fully exploit GPU parallelism, the best-improvement strategy is employed to select the best move among all candidate moves.

The tensor-based implementation of each operator follows a generic five-step process, the first four steps perform tensor-based move evaluation, and the final step performs the necessary updates. Specifically, these steps are: (1) **Extraction**: Extract relevant *sequence attribute tensors* from the solution tensor $\mathbf{T}_s$ according to the specific operator. (2) **Concatenation**: Concatenate the extracted attribute tensors to construct the *neighbor attribute tensors* which represent all neighboring solutions within the operator's neighborhood. (3) **Differencing**: Compute the *attribute delta tensors* containing the differences between the current and modified routes. (4) **Evaluation**: Compute the *scoring tensor* $\mathbf{F}$ based on the evaluation function and identify the best-improvement value and the corresponding indices α^* in $\mathbf{F}$ for the best move ζ. (5) **Update**: Transfer the best move back to the CPU, update the solution S, and synchronize the updated solution tensor $\mathbf{T}_s$ along with other auxiliary tensors on the GPU for the next iteration.

4.4.2 Attribute-based solution tensor representation

As described in Section 4.3.2, we use *attribute matrices* to store attribute values, enabling constant-time move evaluation. Building on this, we aggregate these attribute matrices into a solution tensor $\mathbf{T}_s$ for GPU acceleration. Formally, the solution tensor $\mathbf{T}_s$ is a 4-dimensional tensor defined as

$$\mathbf{T}_s \in \mathbb{R}^{I \times J \times K \times K} \tag{4.5}$$

Here, I denotes the number of maintained attributes, J the maximum number of routes, and $K = \max_{1 \le i \le M}\{|R_i|\}$ the maximum route length, i.e., the number of nodes in the longest route. The entries of $\mathbf{T}_s$ are indexed as

$$(\mathbf{T}_s)_{ijkl}, \quad i \in \{1, \ldots, I\}, j \in \{1, \ldots, J\}, k, l \in \{1, \ldots, K\} \tag{4.6}$$

where $(\mathbf{T}_s)_{ijkl}$ represents the value of the i-th attribute for the subsequence starting at position k and ending at position l in the j-th route. Figure 4.4 illustrates the structure of the solution tensor representation.

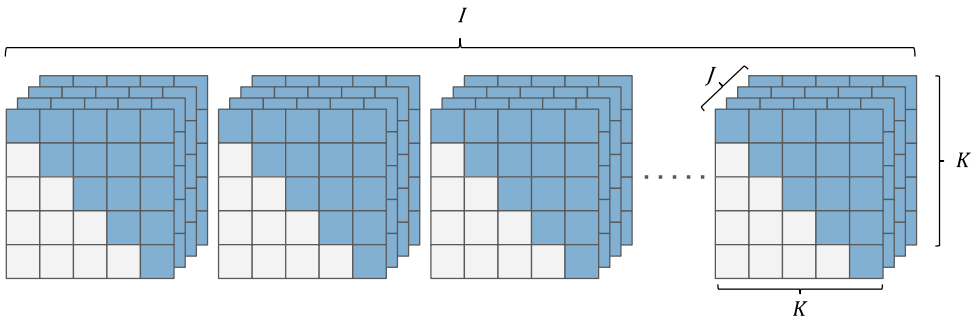


Figure 4.4 – Illustration of the solution tensor representation.

To handle considered attributes introduced in Section 4.3.3, we maintain the corresponding key attribute tensors: travel distance ($\mathbf{D}$), duration ($\mathbf{T}_D$), earliest and latest arrival times ($\mathbf{T}_E$, $\mathbf{T}_L$), time window violations ($\mathbf{T}_V$), incoming and outgoing loads ($\mathbf{L}_I$, $\mathbf{L}_O$), and maximum load ($\mathbf{L}_M$). Auxiliary tensors are also introduced to support efficient computation, including the first and last node indices of each sequence ($\mathbf{I}_S$, $\mathbf{I}_E$) and a boolean mask ($\mathbf{B}$) for valid entries. This attribute-based representation is flexible and can be extended for other VRP variants.

4.4.3 Tensor-based implementation of local search operators

Building on the solution tensor $\mathbf{T}_s$ defined in Equation (4.5), we develop the tensor-based implementation for four common local search operators in vehicle routing, *Relocate*, *Swap*, *2-opt**, and *2-opt*, as introduced in Section 4.3.1, taking advantage of GPU parallelism to speed up move evaluation. The following sections describe the five generic steps of the tensor-based implementation in detail.

Extraction

Based on the sequence concatenation-based move evaluation introduced in Section 4.3, *Extraction* step extracts specific parts of the solution tensor $\mathbf{T}_s$, referred to as *sequence attribute tensors*, using tensor operations such as indexing, slicing, and reshaping. These tensors contain all attribute values associated with the relevant route subsequences which are required to compute the attributes of the concatenated sequences in subsequent steps. This process leverages highly parallelizable tensor operations, such as slicing, indexing, and reshaping, making it well-suited for GPU acceleration. The *Extraction* step is the critical step that distinguishes the operators and is implemented separately for *intra-route*

and *inter-route* operators.

Route-based extraction for intra-route operators We propose a *route-based extraction* for the intra-route operators (*Intra-Relocate*, *Intra-Swap* and *2-opt*), which involve only a single route.

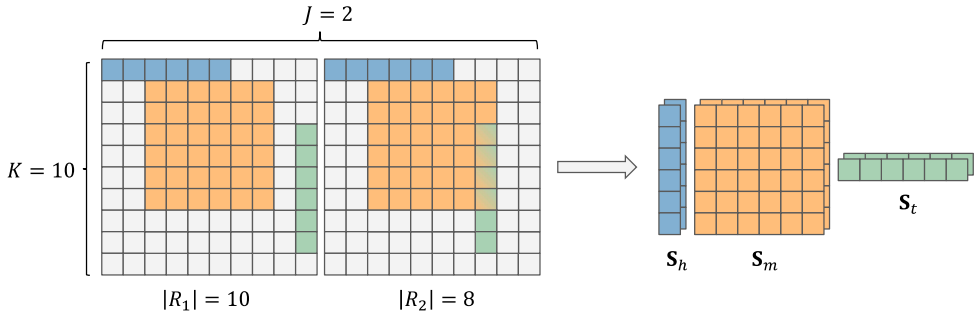


Figure 4.5 – Example of the extraction operation for *2-opt*. For clarity, we illustrate a single-attribute slice of the solution tensor and show a 3D tensor with dimensions $J \times K \times K = 2 \times 10 \times 10$, omitting the first dimension of $\mathbf{T}_s$.

As shown in Figure 4.1f, the *2-opt* operator divides the route at two cut points into three subsequences: the head sequence S_h, the middle sequence S_m, and the tail sequence S_t. The head sequence contains the nodes before the first cut, the middle sequence contains the nodes between the two cuts, and the tail sequence contains the nodes after the second cut. In the solution tensor $\mathbf{T}_s$, we extract the attribute values of all subsequences fully contained in each of these three parts into three *sequence attribute tensors* $\mathbf{S}_h$, $\mathbf{S}_m$, and $\mathbf{S}_t$. Formally, for each attribute $i \in \{1, \ldots, I\}$ and route $j \in \{1, \ldots, J\}$, we define:

$$\mathbf{S}_h = (\mathbf{T}_s)_{ij1l}, \qquad l \in \{1, \ldots, K-4\} \tag{4.7a}$$

$$\mathbf{S}_m = (\mathbf{T}_s)_{ijkl}, \qquad k \in \{2, \ldots, K-3\}, l \in \{3, \ldots, K-2\} \tag{4.7b}$$

$$\mathbf{S}_t = (\mathbf{T}_s)_{ijk|R_j|}, \qquad k \in \{4, \ldots, K-1\} \tag{4.7c}$$

As illustrated in Figure 4.5, for each attribute i and route R_j, the head sequence tensor $\mathbf{S}_h$ is always located in the first row of the attribute matrix, capturing the attribute values of relevant subsequences from the first node ($k = 1$) up to the first cut point. The middle sequence tensor $\mathbf{S}_m$ stores attributes between two cut points, forming a square submatrix. The tail sequence tensor $\mathbf{S}_t$ stores attributes from second cut point to the last node, corresponding to the R_j-th column of the attribute matrix. The similar principles of the extraction process are illustrated for the other operators in the subsequent Figures 4.6-

4.9. Since the *2-opt* operator is only applied to VRP variants with symmetric settings, reversing the middle sequence can be implemented by swapping the first node index tensor $\mathbf{I_s}$ and the last node index tensor $\mathbf{I_e}$ of the attribute tensor $\mathbf{S}_m$. The reshaping operation is consistently applied to extracted *sequence attribute tensors* across all operators, enabling the subsequent *Concatenation* step via tensor broadcasting.

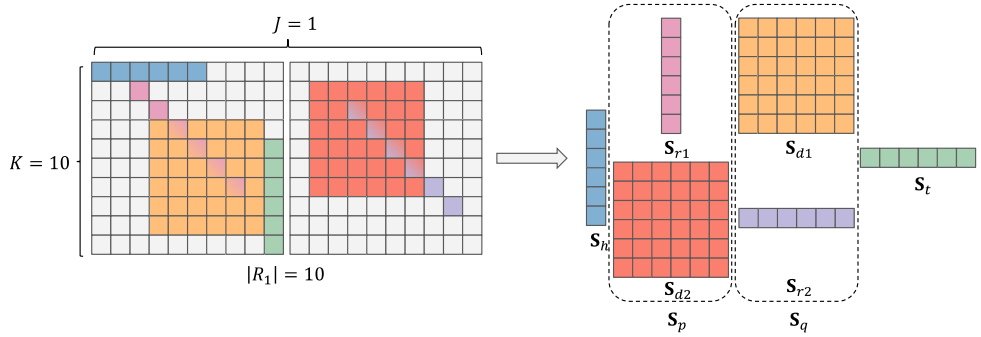


Figure 4.6 – Example of the extraction operation for *Intra-Relocate*. For clarity, we illustrate a single-attribute slice of the solution tensor and show a 3D tensor with dimensions $J \times K \times K = 1 \times 10 \times 10$, omitting the first dimension of $\mathbf{T}_s$ and setting $J = 1$.

As shown in Figure 4.1d, the *Intra-Relocate* operator divides a route into four subsequences: the head sequence S_h, the relocated sequence S_r of length N ($N > 0$), the destination sequence S_d preceding the insertion position, and the tail sequence S_t. The relative order of S_r and S_d depends on whether the relocated sequence is moved forward or backward along the route. To handle both cases efficiently on the GPU, we construct two combined sequence attribute tensors, $\mathbf{S}_p$ and $\mathbf{S}_q$. For each attribute $i \in \{1, \ldots, I\}$ and route $j \in \{1, \ldots, J\}$, the extraction proceeds as follows:

$$\mathbf{S}_h = (\mathbf{T}_s)_{ij1l}, \qquad l \in \{1, \ldots, K - N - 2\} \tag{4.8a}$$

$$\mathbf{S}_t = (\mathbf{T}_s)_{ijk|R_j|}, \qquad k \in \{N + 3, \ldots, K\} \tag{4.8b}$$

$$\mathbf{S}_p = \text{concat}(\mathbf{S}_{r1}, \mathbf{S}_{d2}), \qquad \mathbf{S}_q = \text{concat}(\mathbf{S}_{d1}, \mathbf{S}_{r2}) \tag{4.8c}$$

The auxiliary tensors $\mathbf{S}_{r1}$, $\mathbf{S}_{r2}$, $\mathbf{S}_{d1}$, and $\mathbf{S}_{d2}$ capture the two possible relative positions of the relocated and destination sequences along the route. Specifically, $\mathbf{S}_{r1}$ and $\mathbf{S}_{d1}$ correspond to the case where the relocated sequence S_r precedes the destination sequence S_d,

while $\mathbf{S}_{r2}$ and $\mathbf{S}_{d2}$ correspond to the opposite case. These tensors are defined as follows.

$$\mathbf{S}_{r1} = (\mathbf{T}_s)_{ijk(k+N-1)}, \qquad k \in \{2, \dots, K-N-1\} \tag{4.9a}$$
$$\mathbf{S}_{d2} = (\mathbf{T}_s)_{ijkl}, \qquad k, l \in \{2, \dots, K-N-1\} \tag{4.9b}$$
$$\mathbf{S}_{d1} = (\mathbf{T}_s)_{ijkl}, \qquad k, l \in \{N+2, \dots, K-1\} \tag{4.9c}$$
$$\mathbf{S}_{r2} = (\mathbf{T}_s)_{ijk(k+N-1)}, \qquad k \in \{3, \dots, K-N\} \tag{4.9d}$$

Similarly, as illustrated in Figure 4.1e, the *Intra-Swap* operator divides a single route into

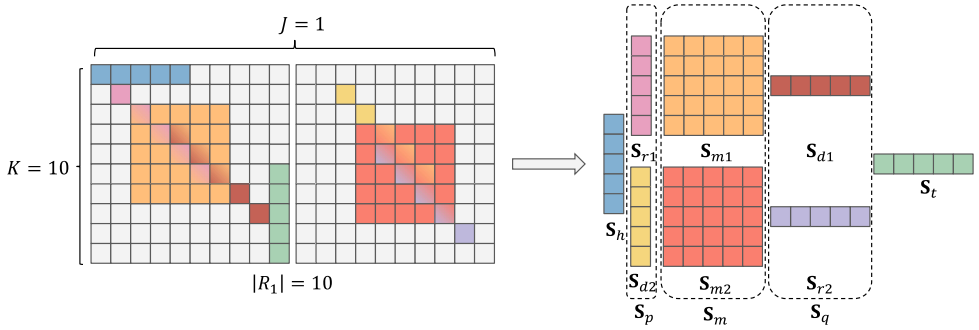


Figure 4.7 – Example of the extraction operation for *Intra-Swap* with $N_1 = 1, N_2 = 2$. For clarity, we illustrate a single-attribute slice of the solution tensor and show a 3D tensor with dimensions $J \times K \times K = 1 \times 10 \times 10$, omitting the first dimension of $\mathbf{T}_s$ and setting $J = 1$.

five subsequences: the head sequence S_h, the swapped sequence S_r of length N_1 ($N_1 > 0$), the middle sequence S_m, the swapped sequence S_d of length N_2 ($N_2 > 0$), and the tail sequence S_t. When the two swapped sequences have different lengths ($N_1 \neq N_2$), two cases must be considered, depending on whether S_r or S_d appears first. Thus, we also introduce auxiliary tensors to capture these cases. For each attribute $i \in \{1, \dots, I\}$ and route $j \in \{1, \dots, J\}$, the extraction process is defined as follows:

$$\mathbf{S}_h = (\mathbf{T}_s)_{ij1l}, \qquad l \in \{1, \dots, K-N_1-N_2-2\} \tag{4.10a}$$
$$\mathbf{S}_t = (\mathbf{T}_s)_{ijk|R_j|}, \qquad k \in \{N_1+N_2+3, \dots, K\}, \tag{4.10b}$$
$$\mathbf{S}_p = \text{concat}(\mathbf{S}_{r1}, \mathbf{S}_{d2}), \qquad \mathbf{S}_m = \text{concat}(\mathbf{S}_{m1}, \mathbf{S}_{m2}), \qquad \mathbf{S}_q = \text{concat}(\mathbf{S}_{d1}, \mathbf{S}_{r2}) \tag{4.10c}$$

The auxiliary tensors $\mathbf{S}_{r1}$, $\mathbf{S}_{r2}$, $\mathbf{S}_{m1}$, $\mathbf{S}_{m2}$, $\mathbf{S}_{d1}$, and $\mathbf{S}_{d2}$ are defined as follows. Here, $\mathbf{S}_{r1}$, $\mathbf{S}_{m1}$ and $\mathbf{S}_{d1}$ correspond to the case where the swapped sequence S_r precedes the swapped

sequence S_d, while $\mathbf{S}_{r2}$, $\mathbf{S}_{m2}$ and $\mathbf{S}_{d2}$ correspond to the opposite case.

$$\mathbf{S}_{r1} = (\mathbf{T}_s)_{ijk(k+N_1-1)}, \qquad k \in \{2, \ldots, K - N_1 - N_2 - 1\} \tag{4.11a}$$
$$\mathbf{S}_{d2} = (\mathbf{T}_s)_{ijk(k+N_2-1)}, \qquad k \in \{2, \ldots, K - N_1 - N_2 - 1\} \tag{4.11b}$$
$$\mathbf{S}_{m1} = (\mathbf{T}_s)_{ijkl}, \qquad k, l \in \{N_1 + 2, \ldots, K - N_2 - 1\} \tag{4.11c}$$
$$\mathbf{S}_{m2} = (\mathbf{T}_s)_{ijkl}, \qquad k, l \in \{N_2 + 2, \ldots, K - N_1 - 1\} \tag{4.11d}$$
$$\mathbf{S}_{d1} = (\mathbf{T}_s)_{ijk(k+N_2-1)}, \qquad k \in \{N_1 + 3, \ldots, K - N_2\} \tag{4.11e}$$
$$\mathbf{S}_{r2} = (\mathbf{T}_s)_{ijk(k+N_1-1)}, \qquad k \in \{N_2 + 3, \ldots, K - N_1\} \tag{4.11f}$$

Route-based extraction for inter-route operators Inter-route operators (*Inter-Relocate*, *Inter-Swap*, and *2-opt**) involve two distinct routes and share a common extraction scheme implemented via the *3-Seq*(N) operation, which splits the route into three subsequences: a head sequence S_h, a middle sequence S_m of length N to be relocated or swapped, and a tail sequence S_t. All three inter-route operators involve two *3-Seq* operations, denoted as *3-Seq*(N_1) and *3-Seq*(N_2). Specifically, for *Inter-Relocate*, $N_1 > 0$ and $N_2 = 0$; for *Inter-Swap*, $N_1 > 0$ and $N_2 > 0$; and for *2-opt**, $N_1 = N_2 = 0$.

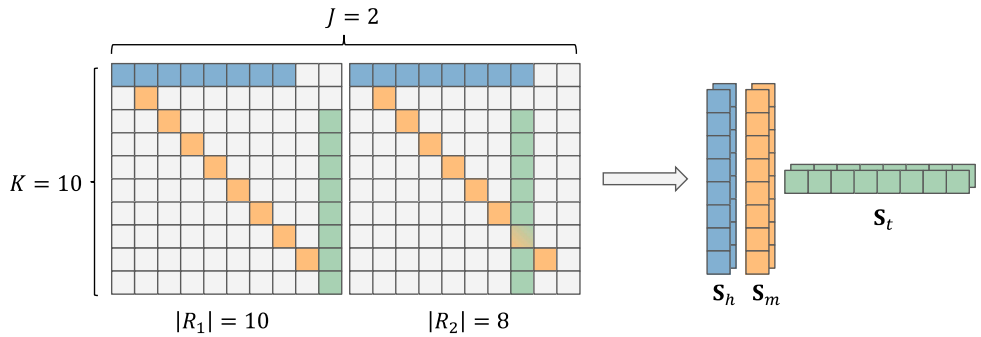


Figure 4.8 – Example of the route-based *3-Seq* operation with $N = 1$ for inter-route operators. For clarity, we illustrate a single-attribute slice of the solution tensor and show a 3D tensor with dimensions $J \times K \times K = 2 \times 10 \times 10$. Note that the length of the routes affects the tail sequence $\mathbf{S}_t$.

Building on the same principles used for intra-route operators, we extend the route-based extraction implementation to *3-Seq* operation. For each attribute $i \in \{1, \ldots, I\}$ and route $j \in \{1, \ldots, J\}$, the route-based *3-Seq(N)* operation is defined as follows:

$$\mathbf{S}_h = (\mathbf{T}_s)_{ij1l}, \qquad l \in \{1, \ldots, K - N - 1\}, N \geq 0 \tag{4.12a}$$
$$\mathbf{S}_m = (\mathbf{T}_s)_{ijk(k+N-1)}, \qquad k \in \{2, \ldots, K - N\}, \text{ for } N > 0; \qquad \emptyset \text{ for } N = 0. \tag{4.12b}$$
$$\mathbf{S}_t = (\mathbf{T}_s)_{ijk|R_j|}, \qquad k \in \{N + 2, \ldots, K\}, N \geq 0 \tag{4.12c}$$

Let L denote the feasible length of the *sequence attribute tensors* after route-based extraction, with $L = K - N - 2$ for the route-based *3-Seq* operation. Notably, this implementation generates high-dimensional intermediate and scoring tensors of size $O(J^2L^2)$ that contain many invalid elements, resulting in substantial memory overhead and increased computational cost in subsequent computations. A detailed complexity analysis is provided in Section 4.4.4.

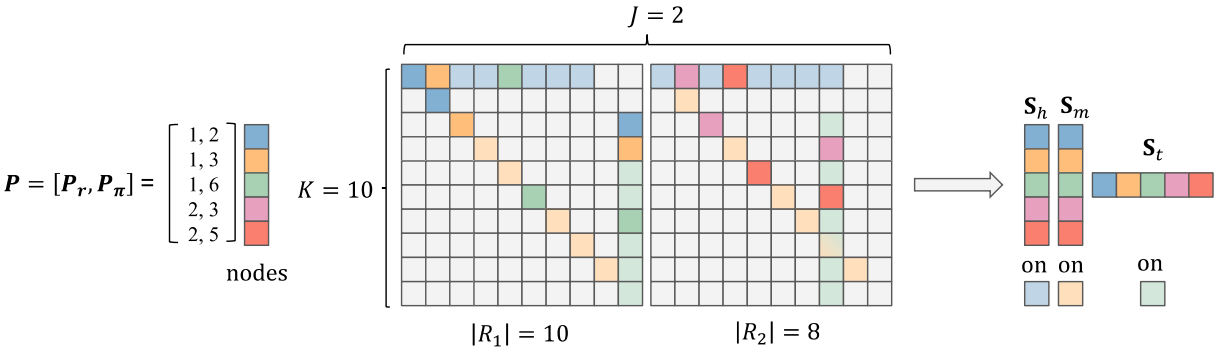


Figure 4.9 – Example of the node-based *3-Seq* operation. To illustrate the differences between node-based and route-based implementations, we reuse the same setting as in Figure 4.8. In this example, five nodes are distributed across two routes: the 2nd, 3rd, and 6th nodes on route R_1, and the 3rd and 5th nodes on route R_2. The figure shows how the head, middle, and tail sequences are extracted based on node positions, and highlights their relationship to those obtained through the route-based implementation.

Node-based extraction for inter-route operators To address the inefficiencies of route-based extraction for inter-route operators, we propose a node-based implementation of the *3-Seq* operation. This approach leverages explicit positional information to extract valid subsequences directly at the node level. Specifically, we construct *positional tensors* $\mathbf{P} = \{\mathbf{P}_r, \mathbf{P}_\pi\}$ from the solution tensor $\mathbf{T}_s$, which encode the position of each valid node, including depots and customers. The *route index tensor* $\mathbf{P}_r$ identifies the route to which each node belongs, while the *position index tensor* $\mathbf{P}_\pi$ specifies the node's position within its route. Both $\mathbf{P}_r$ and $\mathbf{P}_\pi$ are one-dimensional tensors of size Q, where $Q = M + N_C$ denotes the total number of valid nodes across all routes.

$$\mathbf{S}_h = (\mathbf{T}_s)_{i(p_r)k(p_\pi-1)} \tag{4.13a}$$

$$\mathbf{S}_m = (\mathbf{T}_s)_{i(p_r)(p_\pi)(p_\pi+N-1)} \tag{4.13b}$$

$$\mathbf{S}_t = (\mathbf{T}_s)_{i(p_r)(p_\pi+N)|R_{p_r}|} \tag{4.13c}$$

Using this positional information, we extract the *sequence attribute tensors* $\mathbf{S}_h$, $\mathbf{S}_m$, and $\mathbf{S}_t$ only for valid nodes, avoiding the invalid entries generated by the route-based extraction. Given positional indices $p_r \in \mathbf{P}_r$ and $p_\pi \in \mathbf{P}_\pi$, for each attribute $i \in \{1, \ldots, I\}$, the node-based extraction distinguishes two cases. If $N > 0$, the middle sequence exists (Equation (4.13)); if $N = 0$, the middle sequence is empty (Equation (4.14)).

$$\mathbf{S}_h = (\mathbf{T}_s)_{i(p_r)k(p_\pi)} \tag{4.14a}$$

$$\mathbf{S}_m = \emptyset \tag{4.14b}$$

$$\mathbf{S}_t = (\mathbf{T}_s)_{i(p_r)(p_\pi+1)|R_{p_r}|} \tag{4.14c}$$

Unlike the route-based extraction, which is based on the route structure and produces high-dimensional tensors with many invalid elements, the node-based extraction focuses on valid nodes using their positional information, resulting in greater computational efficiency. As a result, the intermediate and scoring tensors are reduced to size $O(Q^2)$, significantly lowering memory consumption and computational cost and enabling a more efficient GPU implementation.

Edge-based extraction for inter-route operators Although the node-based extraction significantly reduces memory usage and computational complexity compared to the route-based method, it still evaluates the entire neighborhood and therefore cannot directly leverage neighborhood reduction strategies [Vid+13] commonly used in CPU-based local search, which restrict neighborhood evaluation to promising moves based on specific criteria to improve search efficiency. To overcome this limitation, we propose an edge-based extraction as one possible implementation that shifts the focus from nodes to edges connecting pairs of nodes.

Building on the *positional tensors* $\mathbf{P} = \{\mathbf{P}_r, \mathbf{P}_\pi\}$ introduced in the node-based extraction, this edge-based method enables selective evaluation of promising moves while avoiding unnecessary computations. Specifically, we introduce a binary mask tensor $\mathbf{M} \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}$ to encode the outcome of neighborhood reduction. An entry $\mathbf{M}_{ij} = 1$ indicates that the edge (v_i, v_j) is valid under a given pruning criterion, and $\mathbf{M}_{ij} = 0$ otherwise. This mask restricts move evaluation to a reduced set of node pairs. Using the mask tensor $\mathbf{M}$ together with the positional tensor $\mathbf{P}$, we retain only valid node pairs $\{(v_i, v_j) \mid \mathbf{M}_{ij} = 1\}$ and construct the corresponding *edge index tensors* $\mathbf{E} = \{\mathbf{E}_i, \mathbf{E}_j\}$. For each valid edge (v_i, v_j), the tensors $\mathbf{E}_i$ and $\mathbf{E}_j$ store the positional information of the source and target

nodes, respectively, obtained by indexing $\mathbf{P}$ as

$$\mathbf{E}_i = \mathbf{P}_i = \{\mathbf{P}_{r_i}, \mathbf{P}_{\pi_i}\}, \quad \mathbf{E}_j = \mathbf{P}_j = \{\mathbf{P}_{r_j}, \mathbf{P}_{\pi_j}\}, \quad \forall (i,j) \text{ such that } \mathbf{M}_{ij} = 1. \tag{4.15}$$

Based on these *edge index tensors*, the edge-based implementation of *3-Seq* operation extracts the *sequence attribute tensors* $\mathbf{S}_h$, $\mathbf{S}_m$, and $\mathbf{S}_t$ following the same procedure as in the node-based extraction. Let E denote the number of selected edges (i.e., the number of nonzero entries in $\mathbf{M}$). The resulting intermediate and scoring tensors have size $O(E)$, yielding substantial reductions in memory consumption and computational cost compared to the node-based method.

In summary, we consider three extraction strategies for inter-route operators: route-based, node-based, and edge-based. As node-based extraction is more efficient than route-based (see the detailed analysis in Section 4.4.4), we define two standard TGA implementations: the node-implemented TGA (NTGA), which applies node-based extraction to inter-route operators with full neighborhood evaluation, and the edge-implemented TGA (ETGA), which uses edge-based extraction to enable neighborhood reduction. Both implementations use route-based extraction for intra-route operators.

Concatenation

After extracting the *sequence attribute tensors*, the extracted tensors are sliced along the first dimension to obtain the individual attribute tensors. The concatenation operation is then applied to these attribute tensors to construct the *neighbor attribute tensors*, which contain the attributes of the concatenated sequences and represent the neighboring solutions generated by the candidate moves.

$$\mathbf{D}(\mathbf{S}_1 \oplus \mathbf{S}_2) = \mathbf{D}(\mathbf{S}_1) + \mathbf{C}_{ij} + \mathbf{D}(\mathbf{S}_2) \tag{4.16a}$$

$$\mathbf{T}_D(\mathbf{S}_1 \oplus \mathbf{S}_2) = \mathbf{T}_D(\mathbf{S}_1) + \mathbf{T}_{ij} + \mathbf{\Delta}_w + \mathbf{T}_D(\mathbf{S}_2) \tag{4.16b}$$

$$i \in \mathbf{I}_e(\mathbf{S}_1), \quad j \in \mathbf{I}_s(\mathbf{S}_2) \tag{4.16c}$$

Let $\mathbf{S}_1$ and $\mathbf{S}_2$ denote the *sequence attribute tensors* of sequences σ_1 and σ_2, respectively. Most attributes of the concatenated sequence can be computed directly using the tensorized formulas in Section 4.3.3, which only involve basic arithmetic operations and are therefore well-suited for GPU parallelization. However, the computation of travel distance and duration requires accounting for the additional costs of connecting the two sequences.

Specifically, these costs can be derived from the travel distance tensor $\mathbf{C}$ (corresponding to the travel distance matrix $\mathcal{C}$), the travel time tensor $\mathbf{T}$ (corresponding to the travel time matrix $\mathcal{T}$), the last node index tensor of $\mathbf{S}_1$ ($\mathbf{I}_e(\mathbf{S}_1)$), and the first node index tensor of $\mathbf{S}_2$ ($\mathbf{I}_s(\mathbf{S}_2)$). The resulting concatenation formulas for travel distance $\mathbf{D}$ and duration $\mathbf{T}_D$ are given in Equations (4.16a) and (4.16b), respectively, which correspond to Equations (4.1b) and (4.4e) in Section 4.3.3.

Differencing

To evaluate the effect of a candidate move, we compute the changes in key attributes between the current and modified routes. The attributes of the current routes are stored in the solution tensor $\mathbf{T}_s$, referred to as the *current attribute tensors*, while the attributes of the modified routes are obtained through the *Concatenation* step, referred to as *neighbor attribute tensors*. The differences are then computed element-wise to produce the *attribute delta tensors*, which include the changes in travel distance ($\Delta\mathbf{D}$), time window violations ($\Delta\mathbf{T}_V$), and load violations ($\Delta\mathbf{L}_V$). These attribute delta tensors form the input for the *Evaluation* step, enabling the tensorized computation of the *scoring tensor* $\mathbf{F}$ for all candidate moves.

Evaluation

Building on the *attribute delta tensors* computed in the *Differencing* step, the *scoring tensor* $\mathbf{F}$ is calculated using the defined evaluation function $\mathcal{F}$ (Equation (4.17a)). Invalid entries are filtered via the boolean mask tensor $\mathbf{B}$ by assigning a large penalty (Equation (4.17b)).

$$\mathbf{F} = \mathcal{F}(\Delta\mathbf{D}, \Delta\mathbf{T}_V, \Delta\mathbf{L}_V) \tag{4.17a}$$

$$\mathbf{F} \leftarrow \mathbf{F} + (\neg\mathbf{B}) \cdot \infty \tag{4.17b}$$

$$F_{\min} = \mathbf{F}_{\boldsymbol{\alpha}^*} \quad \text{where} \quad \boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}} \mathbf{F}_{\boldsymbol{\alpha}} \tag{4.17c}$$

Then, an *argmin* operation is applied to the *scoring tensor* $\mathbf{F}$ to identify the best move with the minimal score value $F_{\min}$ and its corresponding indices $\boldsymbol{\alpha}^*$ (Equation (4.17c)). This operation is realized via a parallel *reduction* on the GPU. The indices $\boldsymbol{\alpha}^*$ encode the information necessary to identify the best move, including the involved routes and the starting positions of the sequences to be relocated or swapped. Finally, this move information is transmitted to the CPU and used to update the solution S.

Tensor update

Once the best move is performed on the CPU, the solution tensor representation $\mathbf{T}_s$ needs to be updated based on the resulting solution. Since the considered operators only impact one or two routes, only the affected parts of the tensor representation $\mathbf{T}_s$ require updating, avoiding the need to reconstruct the entire tensor from scratch. Specifically, the update process consists of two main steps: (1) Resize the solution tensor $\mathbf{T}_s$ if the maximum number of routes J or the maximum route length K change. (2) Update the attribute values of the affected routes on the tensor representation $\mathbf{T}_s$ on GPU. Additionally, the *positional tensors* $\mathbf{P} = \{\mathbf{P}_r, \mathbf{P}_\pi\}$ and the *edge index tensors* $\mathbf{E} = \{\mathbf{E}_i, \mathbf{E}_j\}$ also need to be updated accordingly if the node-based or edge-based extraction methods are used for inter-route operators.

4.4.4 Complexity analysis

We analyze the space and time complexity of the proposed tensor-based GPU acceleration (TGA) framework in terms of the number of tensor elements processed. We assume that up to p GPU threads can be scheduled in parallel. Let I be the number of maintained attributes, J the maximum number of routes, and K the maximum route length. Let L denote feasible length of the *sequence attribute tensors* in the route-based extraction. For inter-route operators, let $Q = M + N_C$ be the total number of valid nodes in the node-based extraction, and E the number of valid node pairs after neighborhood reduction in the edge-based extraction.

Tensor initialization and update The solution tensor $\mathbf{T}_s \in \mathbb{R}^{I \times J \times K \times K}$ requires $O(IJK^2)$ space and can be constructed on the GPU in $O(IJK^2/p)$ time. Since each accepted move modifies at most two routes, updating $\mathbf{T}_s$ costs $O(IK^2)$ space and $O(IK^2/p)$ time.

When node-based or edge-based extraction is enabled for inter-route operators, the *positional tensors* $\mathbf{P} = \{\mathbf{P}_r, \mathbf{P}_\pi\}$ and the *edge index tensors* $\mathbf{E} = \{\mathbf{E}_i, \mathbf{E}_j\}$ must be updated. Updating the *positional tensors* $\mathbf{P}$ has the complexity of $O(Q)$, whereas updating the *edge index tensors* $\mathbf{E}$ requires generating valid node pairs based on the *positional tensors* $\mathbf{P}$ and the mask tensor $\mathbf{M}$, and has a complexity of $O(Q^2)$, potentially becoming the bottleneck in the edge-based implementation.

Additionally, in practice, tensor initialization and updates involve data transfers be-

tween the CPU and GPU for building or updating the solution tensor $\mathbf{T}_s$. As a result, when tensor-based evaluation is relatively fast or the neighborhood size is small, data transfer overheads may dominate the total runtime despite the favorable complexity.

Tensor-based move evaluation Tensor-based move evaluation consists of four steps: *Extraction*, *Concatenation*, *Differencing*, and *Evaluation*. The *Extraction* step performs slicing, indexing, and reshaping operations on the solution tensor $\mathbf{T}_s$ to obtain the *sequence attribute tensors*. The *Concatenation* step constructs *neighbor attribute tensors* for each attribute independently, while the *Differencing* step computes *attribute delta tensors* for required attributes according to the evaluation function. Both of them involve element-wise arithmetic operations and broadcasting. These tensor operations involved in the first three steps can be highly parallelized on GPU. The Evaluation step computes the *scoring tensor* $\mathbf{F}$ from the *attribute delta tensors* and identifies the best move via a *reduction* operation. This step has a time complexity of $O(n/p+\log p)$, where n denotes the number of elements in $\mathbf{F}$.

Based on the above analysis, Table 4.2 presents the dominant overall complexity and summarizes the per-step space and time complexity for route-based intra-route operators and for three extraction strategies of inter-route operators: route-based, node-based, and edge-based. The table shows that the overall computational cost is largely determined by

Table 4.2 – Per-step space and time complexity of tensor-based move evaluation for intra-route operators (route-based), and inter-route operators (route-based, node-based, and edge-based).

Step	Intra-route (route-based)		Inter-route (route-based)		Inter-route (node-based)		Inter-route (edge-based)	
	Space	Time	Space	Time	Space	Time	Space	Time
Extraction	$O(IJL^2)$	$O\left(\frac{IJL^2}{p}\right)$	$O(IJL)$	$O\left(\frac{IJL}{p}\right)$	$O(IQ)$	$O\left(\frac{IQ}{p}\right)$	$O(IE)$	$O\left(\frac{IE}{p}\right)$
Concatenation	$O(IJL^2)$	$O\left(I\frac{JL^2}{p}\right)$	$O(IJ^2L^2)$	$O\left(I\frac{J^2L^2}{p}\right)$	$O(IQ^2)$	$O\left(I\frac{Q^2}{p}\right)$	$O(IE)$	$O\left(I\frac{E}{p}\right)$
Differencing	$O(JL^2)$	$O\left(\frac{JL^2}{p}\right)$	$O(J^2L^2)$	$O\left(\frac{J^2L^2}{p}\right)$	$O(Q^2)$	$O\left(\frac{Q^2}{p}\right)$	$O(E)$	$O\left(\frac{E}{p}\right)$
Evaluation	$O(JL^2)$	$O\left(\frac{JL^2}{p}+\log p\right)$	$O(J^2L^2)$	$O\left(\frac{J^2L^2}{p}+\log p\right)$	$O(Q^2)$	$O\left(\frac{Q^2}{p}+\log p\right)$	$O(E)$	$O\left(\frac{E}{p}+\log p\right)$
Overall	$O(IJL^2)$	$O\left(I\frac{JL^2}{p}+\log p\right)$	$O(IJ^2L^2)$	$O\left(I\frac{J^2L^2}{p}+\log p\right)$	$O(IQ^2)$	$O\left(I\frac{Q^2}{p}+\log p\right)$	$O(IE)$	$O\left(I\frac{E}{p}+\log p\right)$

the size of the *scoring tensor* $\mathbf{F}$. When $\mathbf{F}$ is large, the *Evaluation* step dominates the computational cost; otherwise, the *Extraction* and *Concatenation* steps, particularly the latter, become the main contributors. For inter-route operators, the sizes satisfy $J^2L^2 > Q^2 > E$, due to the presence of the large number of invalid entries in the route-based implementation and the neighborhood reduction in the edge-based implementation. Consequently, the route-based implementation exhibits the highest space and time complexity. The node-

based implementation significantly reduces complexity by restricting evaluation to valid nodes, while the edge-based implementation further improves computational efficiency by limiting evaluation to E valid node pairs, which is typically much smaller than Q^2.

However, the edge-based implementation introduces additional overhead associated with constructing and maintaining the *edge index tensors* $\mathbf{E}$. Moreover, when neighborhood reduction is less effective, this approach may compromise solution quality. These trade-offs are examined in detail in the next section.

4.5 Computational results

This section presents computational experiments to evaluate the performance of the proposed tensor-based GPU acceleration (TGA) framework, focusing on two implementations: the node-implemented TGA (NTGA) and the edge-implemented TGA (ETGA), applied to common local search operators in vehicle routing problems.

The experiments aim to assess the potential of the proposed TGA framework for solving vehicle routing problems and to establish a basis for future research in this direction. Accordingly, the following sections focus on designing fair and consistent experiments to assess the effectiveness of the TGA framework, particularly in terms of computational efficiency, compared to traditional CPU-based implementations.

4.5.1 Algorithms for experimentation

This section presents an overview of the algorithms used in the comparative study to evaluate the performance of the proposed tensor-based GPU acceleration (TGA) framework.

The TGA framework is highly extensible and can be integrated into various local search based algorithms. To demonstrate its effectiveness, we incorporate TGA into the MA-FIRD algorithm [LH25], which has shown its effectiveness in solving the VRPSPDTW. Since both the CVRP and the VRPTW can be considered special cases of the VRPSPDTW, this integration allows the algorithm to address these variants as well.

MA-FIRD of [LH25] is a memetic algorithm that combines a feasible and infeasible route descent search (FIRD) and an adaptive route-inheritance crossover (ARIX). The FIRD procedure is a local search procedure using the best-improvement strategy and the sequence concatenation-based move evaluation method. The ARIX crossover generates

offspring solutions by inheriting routes from multiple parent solutions and incorporates a learning mechanism to adaptively adjust the number of parents and the heuristics used to complete the partial offspring under construction.

To evaluate the performance of TGA, we modified the initial MA-FIRD algorithm of [LH25] and incorporated TGA. Specifically, we first adapted the original MA-FIRD algorithm of [LH25] as the baseline, denoted MA-N. Its TGA-based counterpart, referred to as MA-NTGA, integrates the proposed node-implemented TGA (NTGA) into MA-N. Both MA-N and MA-NTGA perform full neighborhood evaluation without applying neighborhood reduction of Section 4.4.3. To further assess the impact of neighborhood reduction in conjunction with TGA, we adopted two additional algorithm variants (MA-E and MA-ETGA) using the neighborhood reduction mechanism proposed in [Vid+13], with the granularity threshold θ controlling the number of nearest neighbors per node. Specifically, MA-E extends the CPU-based MA-N by incorporating neighborhood reduction into its local search operators, while MA-ETGA is its TGA-based counterpart, which integrates the edge-implemented TGA (ETGA) to support the same neighborhood reduction mechanism.

The general MA framework of four variants is illustrated in Algorithm 4. For a given problem instance $\mathcal{I}$, the algorithm begins by generating an initial population $\mathcal{P}$ consisting of μ candidate solutions (line 4), with the best solution S^* being recorded (line 5). The algorithm then iteratively updates the population by creating new candidate solutions (lines 6-18). In each generation, the adaptive route-inheritance crossover ARIX generates a new solution S (line 7), which is then improved using feasible and infeasible search based on the given local search operators in $\mathcal{O}$ (line 8). Note that the route descent strategy is omitted in this setting to simplify the search process and ensure compatibility with problem variants that do not consider the number of vehicles as an objective. The improved solution S is used to update the population according to the population management strategy (line 9). If an improved best solution is found, S^* is updated, and the stagnation counter is reset to zero (lines 11-13). Otherwise, the stagnation generation counter φ_{st} increments (lines 14-15). The algorithm terminates and returns the best solution S^* (line 19) when the given termination condition is reached (line 6).

Within the MA framework, the feasible and infeasible search constitutes the core local search component responsible for search intensification and distinguishes the four algorithmic variants considered in our comparative study. Algorithm 5 presents the general procedure of the feasible and infeasible search for the TGA-based algorithms (MA-NTGA

Algorithm 4 General MA framework

Input: Instance $\mathcal{I}$, population size μ, set of operators $\mathcal{O} = \{O_1, O_2, \ldots, O_n\}$.
Output: The best solution S^*.
$\varphi \leftarrow 0, \varphi_{st} \leftarrow 0$ /* Current generation and stagnation generation counter */
$\mathcal{P} \leftarrow Initialization(\mathcal{I}, \mu)$
$S^* \leftarrow BestSolution(\mathcal{P})$ /* Record current best solution */
while The given termination condition is not met **do**
  $S \leftarrow ARIX(\mathcal{P})$
  $S \leftarrow FeasibleInfeasibleSearch(S, \mathcal{O})$
  $\mathcal{P} \leftarrow UpdatePopulation(S, \mathcal{P})$
  $S_{best} \leftarrow BestSolution(\mathcal{P})$ /* The best solution in this generation */
  if $f(S_{best}) < f(S^*)$ **then**
    $S^* \leftarrow S_{best}$ /* Update the best solution */
    $\varphi_{st} \leftarrow 0$
  else
    $\varphi_{st} \leftarrow \varphi_{st} + 1$ /* Stagnation counter is incremented */
  end if
  $\varphi \leftarrow \varphi + 1$
end while
return S^*

Algorithm 5 Feasible and infeasible search

Input: Input solution S, set of operators $\mathcal{O} = \{O_1, O_2, \ldots, O_n\}$.
Output: Improved solution S.
Initialize tensor $\mathbf{T}_s$ on GPU based on the solution S.
while The given termination criterion is not met **do**
  Select an operator $O_i \in \mathcal{O}$.
  $\zeta \leftarrow Evaluate(\mathbf{T}_s, S, O_i)$ /*Evaluate the neighborhood of the operator O_i and obtain the best move ζ*/.
  if The move ζ is accepted based on the defined evaluation **then**
    Update current solution S based on the move ζ.
    Update tensor $\mathbf{T}_s$ based on current solution S.
  end if
end while
return S

and MA-ETGA), in which the proposed tensor-based GPU acceleration is employed for move evaluation (line 6). In contrast, their CPU-based counterparts (MA-N and MA-E) exclude lines 3 and 9 of Algorithm 5 and rely on traditional CPU implementations of the local search operators for move evaluation (line 6).

Table 4.3 – Characteristics of the four algorithm variants.

Algorithm	Move Evaluation	Neighborhood Reduction
MA-N	CPU-based	No
MA-E	CPU-based	Yes
MA-NTGA	TGA-based	No
MA-ETGA	TGA-based	Yes

As summarized in Table 4.3, these four algorithms (MA-N, MA-E, MA-NTGA, and MA-ETGA) share the same memetic algorithm framework, best-improvement local search strategy, and set of operators: *Relocate*, *Swap*, *2-opt** for the VRPTW and VRPSPDTW, plus *2-opt* for the CVRP. The only differences among them lie solely in their move evaluation implementations and the use of neighborhood reduction.

4.5.2 Benchmark instances

We assess the proposed TGA on three VRP variants: the capacitated VRP (CVRP), VRP with time windows (VRPTW), and the VRP with simultaneous pickup and delivery and time windows (VRPSPDTW), using widely recognized benchmark instances from the literature. Specifically, experiments were conducted on a representative subset of 210 benchmark instances across three VRP variants: 100 *X* benchmark instances [Uch+17] for CVRP, with 100 to 1000 customers (see Section 1.1.2); 20 *JD* benchmark instances [LTY21] for VRPSPDTW, based on real-world JD Logistics data with 200, 400, 600, 800, and 1000 customers; and 90 selected *GH* instances for VRPTW, covering a wide range of sizes and characteristics. As presented in Section 1.1.2, the *GH* benchmark [GH99] originally comprises 300 instances with 200 to 1000 customers, divided into three types: *C* (clustered), *R* (random), and *RC* (mixed), each including instances with both narrow and wide time windows and small and large vehicle capacities.

4.5.3 Experimental protocol

All four algorithm variants, MA-N, MA-E, MA-NTGA, and MA-ETGA, were implemented in C++, with the tensor-based GPU acceleration leveraging the PyTorch C++ API. Experiments were performed on an Intel Xeon Gold 6248 2.50 GHz processor and an NVIDIA V100 GPU with 32 GB memory. As discussed in Section 4.2.1, the V100 is no longer state-of-the-art but remains a mature and widely used GPU platform for algorithm evaluation. Importantly, since TGA is deterministic and hardware-independent, the GPU

choice affects only runtime performance, and newer GPUs are expected to further improve efficiency without changing solution quality.

Our experimental design aims to provide a fair and consistent evaluation of TGA, with a particular focus on computational gains over traditional CPU-based implementations. To ensure controlled comparisons, the algorithmic parameters were adopted from MA-FIRD [LH25] and kept consistent across all variants, avoiding biases due to differing configurations. Key parameters, including the population size μ and the granularity threshold θ (introduced in MA-E and MA-ETGA), were tuned using the automatic parameter configuration tool Irace [Lóp+16] on a selected subset of instances from each benchmark. Table 4.4 reports the candidate ranges and selected values for each benchmark set.

Table 4.4 – Parameter tuning results.

Parameter	Description	Candidate values	*X*	*GH*	*JD*
μ	Population size	{10, 20, 30, 40, 50}	40	10	10
θ	Granularity threshold	{20, 50, 80, 100, 120}	20	100	80

To ensure fair comparisons with state-of-the-art methods, we adopt the same termination criteria as in the corresponding studies for each benchmark set. For the *X* (CVRP) instances, the time limit is proportional to the number of customers, set to $N_C \times 240/100$ seconds [Vid+12; Vid22]. For the *GH* (VRPTW) instances, the limits are 1800, 3600, and 7200 seconds for instances with ≤ 200, ≤ 800, and > 800 customers, respectively [Koo+22]. For the *JD* (VRPSPDTW) instances, following [LH25], termination occurs when any of the following criteria is met: a maximum of 5000 generations, 500 consecutive generations without improvement, or a time limit of 7200 seconds. Each instance was solved 10 times by each algorithm variant.

4.5.4 Computational results and comparisons

Figure 4.10 compares the runtime and average gap of MA-E, MA-NTGA, and MA-ETGA relative to the baseline MA-N across 210 instances of three benchmark sets: *X* (CVRP), *GH* (VRPTW), and *JD* (VRPSPDTW). The average gap is defined as the relative difference in objective value with respect to MA-N, and the runtime corresponds to the time limit for *X* and *GH*, and the average computation time for *JD*. Detailed numerical results reported in Tables B.4-B.8 in the appendix. Overall, all three variants

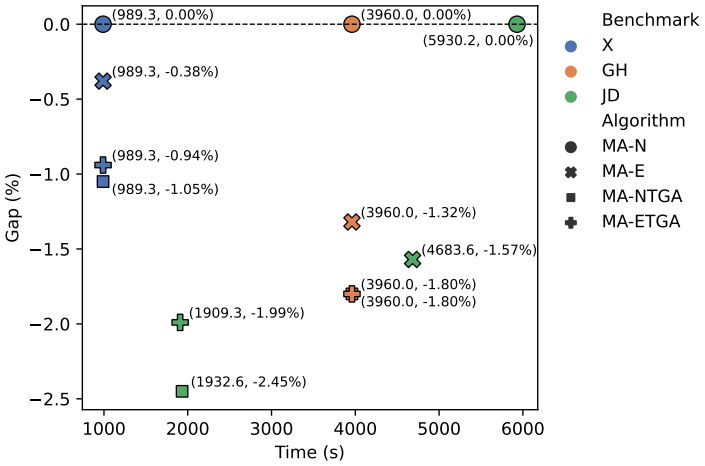


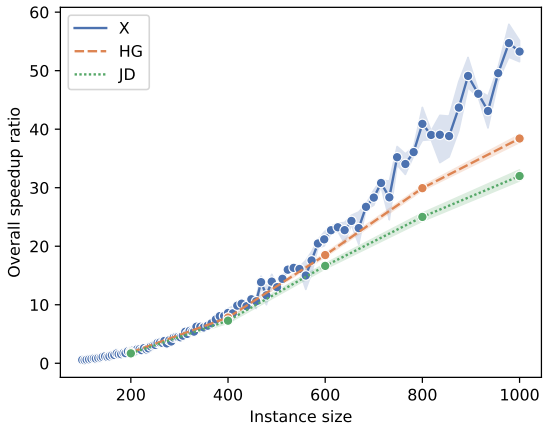
Figure 4.10 – Comparative performance of MA-N, MA-E, MA-NTGA, and MA-ETGA on three benchmark sets. The x-axis shows runtime (s), and the y-axis shows average gap (%) relative to the baseline MA-N.

outperform MA-N, while the TGA-based algorithms further improve upon both MA-N and MA-E across all benchmarks. For the *X* and *GH* instances, where termination is solely time-based, performance differences are clearly reflected in the gap values. On *X*, MA-NTGA and MA-ETGA achieve average gaps of -1.05% and -0.94%, respectively, compared to -0.38% for MA-E. On *GH*, both MA-NTGA and MA-ETGA attain average gaps of -1.80%, outperforming MA-E's -1.32%. For the *JD* instances, which additionally use generation-based and stagnation-based stopping criteria, MA-NTGA and MA-ETGA achieve larger improvements (-2.45% and -1.99%) than MA-E (-1.57%), while requiring shorter average computation times. These results demonstrate both the computational efficiency and enhanced search capability of the TGA-based implementations.

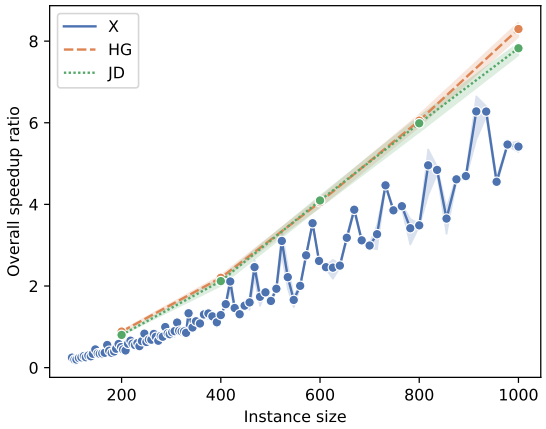
MA-NTGA and MA-ETGA exhibit comparable performance, with MA-NTGA showing a slight advantage. This suggests that although the edge-implemented TGA (ETGA) effectively accelerates move evaluation by reducing the neighborhood size, it does not automatically translate into superior solution quality. The effectiveness of neighborhood reduction strongly depends on its design, inappropriate pruning may eliminate promising high-quality moves. Moreover, the edge-based implementation introduces extra overhead, affecting overall efficiency, which is analyzed in detail in the following section.

4.5.5 Analysis of overall speedup ratio

To provide an intuitive assessment of the performance gains achieved by TGA-based algorithms over their traditional CPU-based counterparts, we conducted additional experiments to measure speedup ratios for move evaluation across benchmark instances. Let t_{CPU} denote the total runtime spent on move evaluations of the CPU-based algorithms (MA-N and MA-E) after 100 iterations, and t_{TGA} the corresponding runtime for the TGA implementations (MA-NTGA and MA-ETGA) under the same iteration budget. The speedup ratio is defined as $\gamma_s = \frac{t_{\text{CPU}}}{t_{\text{TGA}}}$. Each instance was executed 10 times, with paired algorithm variants (MA-N vs. MA-NTGA and MA-E vs. MA-ETGA) following identical search trajectories in each run to ensure fair and consistent comparisons. Figure 4.11 reports the resulting speedup ratios across all tested instances, highlighting the relationship between instance size and speedup ratio for different benchmark instances.



(a) Overall speedup of MA-NTGA relative to MA-N.



(b) Overall speedup of MA-ETGA relative to MA-E.

Figure 4.11 – Relationship between instance size and speedup ratio across benchmark instances. The x-axis denotes the instance size, while the y-axis reports the speedup of TGA-based algorithms over their CPU-based counterparts.

The experimental results reveal a clear positive correlation between instance size and speedup ratio across all benchmarks. For the *GH* (VRPTW) and *JD* (VRPSPDTW) instances, both TGA variants exhibit monotonic speedup growth as instance size increases. In particular, MA-NTGA achieves substantial acceleration, with speedup ratios rising from approximately 2 to over 30 when scaling from 200 to 1000 nodes, consistent with theoretical expectations and demonstrating the effectiveness of TGA on large-scale instances. MA-ETGA follows the same trend but with smaller speedup ratios (from about

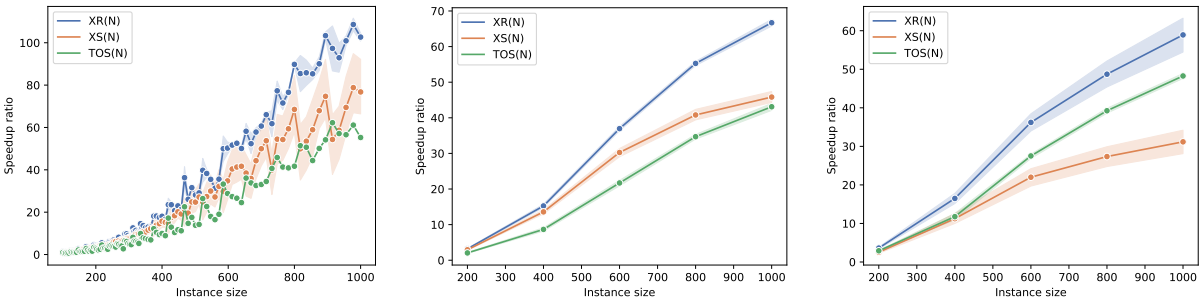
1 to over 8), as MA-E already reduces move evaluation via neighborhood reduction, leaving less exploitable parallelism for TGA.

A similar trend is observed for the X (CVRP) benchmark, where MA-NTGA achieves even higher acceleration on large instances due to the less constrained nature of CVRP, with speedups exceeding 50× on approximately 1000-node instances. In contrast, no noticeable acceleration is observed for small (around 100-node) instances, where GPU overhead outweighs the performance gains, underscoring that TGA is most beneficial for large and computationally intensive problems. For MA-ETGA on X instances, the speedup ratios are substantially lower, as MA-E already eliminates a large portion of move evaluations under the granularity threshold $\theta = 20$, leaving limited scope for further acceleration by TGA. Additionally, some variability in speedup is observed among X instances, which can be attributed to instance-specific characteristics, particularly the number of vehicles. As discussed in Section 4.4.4, instances requiring more vehicles generate larger scoring tensors, which increases the computational cost to indentify the best move.

4.5.6 Acceleration of move evaluation across operators

We further analyze TGA performance at the operator level by examining speedup gains across benchmark instances of varying sizes. Figure 4.12 shows the speedup ratios for different local search operators, which are calculated as the average runtime of move evaluation for the CPU-based implementation divided by that of the corresponding TGA-based implementation.

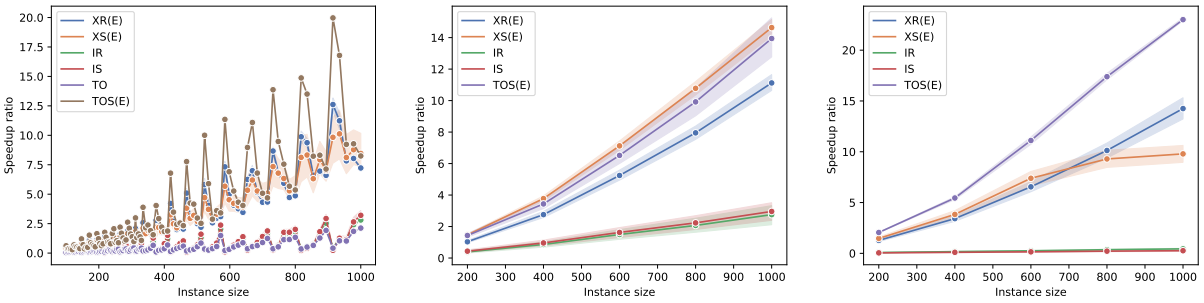
Overall, the speedup ratio increases with instance size. TGA achieves particularly significant acceleration for inter-route operators (*Inter-Relocate*, *Inter-Swap*, *2-opt**), which involve complex cross-route interactions and benefit strongly from GPU parallelization. Node-based implementations reach speedups from 2 up to over 60 for the *GH* and *JD* instances, and exceed 100 for around 1000-node X instances. Edge-based implementations also achieve substantial speedups, though slightly lower, as neighborhood reduction strategy already reduces many move evaluations, leaving less room for additional acceleration. In contrast, intra-route operators (*Intra-Relocate*, *Intra-Swap*, *2-opt*) show more modest gains, with speedups up to around 2 for large 1000-node instances. This contrast highlights the particular advantage of the TGA framework for inter-route operators, which require more complex cross-route computations.



(a) Node-based inter-route operators on the *X* instances.

(b) Node-based inter-route operators on the *GH* instances.

(c) Node-based inter-route operators on the *JD* instances.



(d) intra-route operators and edge-based inter-route operators on the*X* instances.

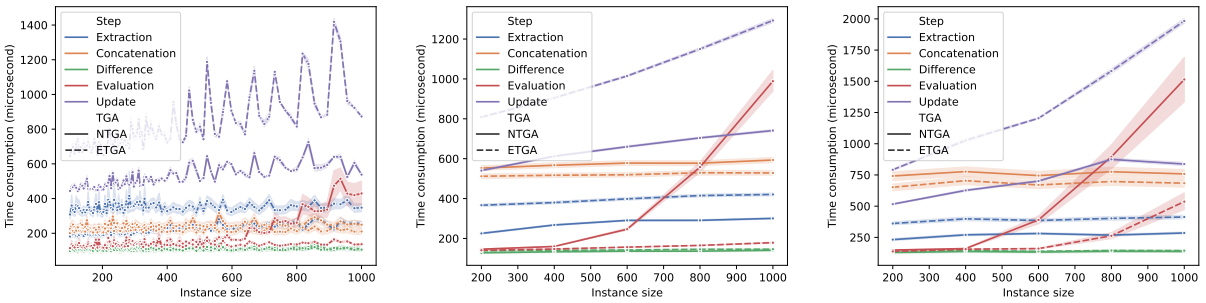
(e) intra-route operators and edge-based inter-route operators on the *GH* instances.

(f) intra-route operators and edge-based inter-route operators on the *JD* instances.

Figure 4.12 – Speedup ratios of local search operators across benchmark instances, including inter-route operators (*Inter-Relocate*, *Inter-Swap*, *2-opt**) with node-based (XR(N), XS(N), TOS(N)) and edge-based implementations (XR(E), XS(E), TOS(E)), as well as route-based intra-route operators (*Intra-Relocate* (IR), *Intra-Swap* (IS), *2-opt* (TO)). The x-axis denotes instance size and the y-axis the corresponding speedup ratio.

4.5.7 Time analysis of key steps in TGA

To complement the theoretical complexity analysis in Section 4.4.4, we examine the practical runtime of TGA's key steps (*Extraction*, *Concatenation*, *Differencing*, *Evaluation*, and *Update*) defined in Section 4.4.3. Figure 4.13 reports the time consumption of each step across varying instance sizes for both NTGA and ETGA.



(a) Time consumption of TGA's steps for different instance sizes on *X* instances.

(b) Time consumption of TGA's steps for different instance sizes on *GH* instances.

(c) Time consumption of TGA's steps for different instance sizes on *JD* instances.

Figure 4.13 – Time analysis of TGA's key steps for varying instance sizes. The x-axis indicates instance size, and the y-axis represents computation time for each step.

Across all three benchmarks, tensor-based move evaluation steps *Extraction*, *Concatenation*, and *Differencing* show relatively stable runtimes, reflecting near-constant complexity. In contrast, *Evaluation* shows the largest variability, as its cost scales with the size of the *scoring tensor*, which depends on instance characteristics such as the number of customers and vehicles. Therefore, for inter-route operators with large neighborhoods and complex cross-route interactions, *Evaluation* dominates the overall runtime. For intra-route operators, the smaller neighborhoods substantially reduce the *Evaluation* cost, making *Extraction* and *Concatenation* more prominent. Additionally, a clear distinction is observed between NTGA and ETGA in terms of *Evaluation* step. By restricting the effective neighborhood through edge-based implementation, ETGA significantly reduces the *Evaluation* time, particularly on large instances. Finally, the *Update* step emerges as a non-negligible and increasing cost, especially for large instances and in the ETGA. *Update* step involves GPU-CPU synchronization, data transfer, and the update of the solution tensor. In ETGA, this cost is further amplified by the maintenance of *edge index tensors*. Moreover, as ETGA substantially shortens the *Evaluation* step, these fixed overheads account for a larger fraction of the total runtime, making *Update* both absolutely and relatively more expensive.

Overall, this analysis identifies the primary performance bottlenecks of TGA. In NTGA, *Evaluation* dominates due to full neighborhood exploration, whereas in ETGA, the substantial reduction in *Evaluation* time makes *Update* the dominant cost for large-scale instances. These observations suggest potential avenues for performance improvement in future work.

4.5.8 Comparative analysis with algorithms in the literature

Although this study does not aim to achieve new best-known solutions (BKS), we provide a comparative analysis of the proposed TGA-based algorithms (MA-NTGA and MA-ETGA) against leading methods from the literature. For the CVRP and VRPTW, BKS values are taken from the CVRPLIB repository (`http://vrp.galgos.inf.puc-rio.br/`) as of December 25, 2025. CVRP comparisons include HGS-2012 [Vid+12] and HGS-CVRP [Vid22]. For the VRPTW, with the objective of minimizing the total travel distance, comparisons include the DIMACS competition reference results and the champion algorithm HGS-DIMACS [Koo+22]. For VRPSPDTW, we report results from the state-of-the-art MA-FIRD method [LH25]. MA-NTGA and MA-ETGA results correspond to the best solutions in Section 4.5.4, with all algorithms evaluated under the same termination criteria as reported in their respective references. Although our TGA-based algorithms leverage GPU acceleration while reference algorithms are CPU-based, Table 4.5 reports PassMark CPU benchmark scores (`https://www.passmark.com/`) for both our experimental CPU and those used in the reference studies. The reported differences show that our CPU performance is within ±10% of the references, indicating broadly comparable CPU performance.

Table 4.5 – Processor specifications and corresponding CPU benchmark scores.

Algorithm	Processor	Base frequency	CPU mark	Difference (%)
HGS-2012	Intel Xeon Gold 6148	2.40 GHz	2072	-8.2%
HGS-CVRP	Intel Xeon Gold 6148	2.40 GHz	2072	-8.2%
HGS-DIMACS	Intel Core i7-6850K	3.60 GHz	2388	+5.8%
MA-FIRD	Intel Xeon E5-2690 v4	2.60 GHz	2069	-8.3%
Ours	Intel Xeon Gold 6248	2.50 GHz	2257	0%

Figure 4.14 summarizes the results, with detailed numerical results reported in Tables B.9-B.13 in the appendix. On the *JD* instances, MA-NTGA and MA-ETGA achieve competitive performance, with mean gaps of -0.66% and -0.27% relative to BKS in shorter

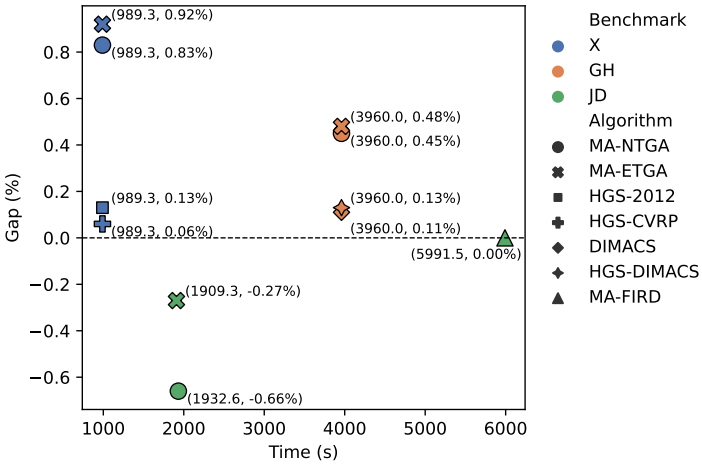


Figure 4.14 – Comparative performance of TGA-based algorithms and state-of-the-art methods on benchmark instances. The x-axis shows runtime (s), and the y-axis shows gap (%) relative to the BKS.

computation times. For the *X* (CVRP) and *GH* (VRPTW) instances, the TGA-based algorithms exhibit larger gaps compared to state-of-the-art methods. Specifically, MA-NTGA attains mean gaps of 0.83% on *X* and 0.45% on *GH*, while MA-ETGA achieves 0.92% and 0.48% on the same benchmarks, respectively. These results indicate that TGA-based algorithms do not achieve top-tier performance on these two benchmarks, which is expected and can be reasonably explained by several factors.

First, the TGA-based algorithms are built on MA-FIRD, originally designed for the more complex VRPSPDTW rather than for the CVRP or VRPTW. This is particularly impactful for the relatively simpler CVRP, where such sophisticated algorithmic components are not strictly necessary, thereby limiting the competitiveness of TGA-based algorithms on these instances. Second, our algorithms employ a limited set of basic local search operators within the TGA framework, whereas state-of-the-art algorithms typically integrate richer and advanced operators, such as SWAP* [Vid22] and RELOCATE* [Koo+22], which significantly enhance solution quality and search capability. Third, MA-NTGA relies on a best-improvement strategy without neighborhood reduction. Although MA-ETGA introduces neighborhood reduction through an edge-based implementation, it remains more constrained than the sophisticated pruning techniques adopted by state-of-the-art algorithms, which are critical to accelerate the search process and improve search efficiency. Taken together, these factors explain the performance gap observed on the *X* and *GH* instances.

In summary, while TGA-based algorithms do not yet match leading CPU-based meth-

ods, they demonstrate the potential of the TGA framework and GPU-based approaches for vehicle routing problems, providing a promising foundation for future improvements.

4.6 Discussion

The TGA framework demonstrates strong extensibility and adaptability. Its attribute-based solution tensor representation allows it to handle a wide range of VRP variants by maintaining problem-specific attribute matrices. Even for more complex VRP variants involving additional decision-making beyond route optimization, such as the multi-depot VRP, the periodic VRP, and the location routing problem, TGA can efficiently optimize the routing component. Its low-coupling architecture fully offloads intensive computations to the GPU, allowing seamless integration into existing local search-based algorithms.

Building on the tensorized implementation of basic local search operators, TGA also supports more advanced operators. For example, *Relocate* and *Swap* can incorporate reversed subsequences by exchanging the first and last node index tensors, as in *2-opt*. Multi-route sequence exchanges can be accommodated by adding tensor dimensions, and filtering strategies, such as tabu-like mechanisms, can be implemented via mask tensors to exclude irrelevant moves.

TGA also exhibits several limitations. Speedups for intra-route operators are less pronounced than for inter-route operators, indicating room for improvement. In node-implemented TGA, the efficiency of the *Evaluation* step is sensitive to the size of the *scoring tensor*. Edge-implemented TGA mitigates this by restricting the neighborhood but introduces additional overhead in the *Update* step due to maintaining *edge index tensors*. These trade-offs suggest that the current solution tensor design does not yet fully support advanced neighborhood reduction techniques commonly used in CPU-based local search. Exploring alternative representations that better accommodate such mechanisms is a promising direction for future work.

Overall, TGA provides a promising direction for leveraging GPU parallelism for solving challenging routing problems. However, comparisons with state-of-the-art algorithms show that it remains early-stage relative to mature CPU-based implementations, with substantial room for improvement in both solution quality and search efficiency. Rather than directly replicating CPU-oriented search strategies which are inherently designed and optimized for CPU architectures, future research should develop search strategies tailored to TGA's parallel execution model. For example, moves are currently evaluated

in parallel on the GPU, but acceptance remains sequential, with only one move applied per iteration. In contrast, methods like *Dynasearch* [CPV02] use dynamic programming to identify optimal combinations of multiple independent moves within the neighborhood. Adapting similar ideas could better exploit TGA's massive parallelism to apply multiple moves simultaneously, enhancing both efficiency and solution quality.

Furthermore, the principles of TGA extend beyond local search and can benefit a wide range of optimization methods. Population-based metaheuristics can leverage parallel evaluation and manipulation of multiple solutions on GPUs, while exact methods can exploit tensor representations of partial solutions or subproblems to evaluate feasibility, bounds, and branching decisions in parallel. This approach accelerates traditionally sequential processes such as branch-and-bound or dynamic programming, offering high computational efficiency while maintaining optimality. More broadly, TGA's philosophy can be applied to other combinatorial optimization problems, improving both algorithmic efficiency and performance.

4.7 Chapter conclusion

This chapter proposes a Tensor-based GPU Acceleration framework, along with two implementations, node-implemented TGA and edge-implemented TGA, for accelerating common local search operators in vehicle routing problems. By leveraging fine-grained GPU parallelism, the proposed framework significantly accelerates neighborhood exploration, which is one of the main computational bottlenecks in local search-based algorithms.

Extensive experiments on three routing problems, CVRP, VRPTW, and VRPSPDTW, confirm the effectiveness and efficiency of our proposed framework, achieving remarkable performance improvements and computational efficiency gains on benchmark instances compared to traditional CPU-based implementations. In-depth analyses of operator-level acceleration and the key steps within the framework provide valuable insights and reveal critical bottlenecks for future improvement.

Due to its low-coupling architecture and strong extensibility, the TGA framework can be seamlessly integrated into various local search-based routing algorithms, offering a powerful and efficient tool for solving a wide range of vehicle routing problem variants. Beyond vehicle routing, the generality of its attribute-based representation and tensorized evaluation pipeline suggests strong potential for extending the framework to other com-

binatorial optimization problems and methods. Future work will focus on refining the framework, enhancing GPU utilization, and exploring broader applications within and beyond routing problems.

Chapter 5

A GPU-accelerated hybrid method for a class of multi-depot vehicle routing problems

Multi-depot vehicle routing problems (MDVRPs) are prevalent in a variety of practical applications. However, they are computationally challenging to solve due to their inherent complexity. This chapter proposes an effective hybrid algorithm for a class of MDVRPs. The algorithm integrates a learning-driven, diversity-controlled route-exchange crossover and a multi-depot-supported feasible-and-infeasible search framework guided by a multi-penalty evaluation function. Two dedicated depot-related local search operators are incorporated to further strengthen the search capability in multi-depot settings. To improve computational efficiency and scalability, an enhanced version of the algorithm is developed that uses a tensor-based GPU acceleration combined with a novel multi-move update strategy. Extensive computational experiments on benchmark instances of three MDVRP variants show that the proposed algorithms are highly competitive with state-of-the-art methods, especially for large-scale instances. The content of this chapter is based on an article submitted to *IEEE Transactions on Evolutionary Computation*.

5.1 Introduction

Since the introduction of the MDVRP [Til69], numerous algorithms have been proposed to address the problem and its variants. Section 1.3.2 presented a comprehensive review of research on MDVRPs. Despite these advancements, no single method dominates across all benchmarks. In terms of algorithmic design, most approaches follow the "cluster-first, route-second" strategy or treat depot assignment and routing separately, rather than integrating them into a unified framework. Several local search based methods [Vid+13; Esc+14; SÇA21] incorporate penalty terms in the objective function to guide the search process. However, these typically neglect explicit modeling of depot-related constraints. In addition, while some efforts have aimed to improve computational efficiency, such as through parallel computing [CM12] or neighborhood reduction techniques [Esc+14; Tu+14], these approaches are generally implemented on traditional CPU architectures. Moreover, most crossover strategies in the MDVRP literature are based on the giant-tour representation combined with the *Split* algorithm [Vid+12; Vid+13], which places strong dependence on an efficient split procedure for offspring generation. Furthermore, only a few studies have evaluated their methods on large-scale problem instances.

In light of these considerations and to address the underlying challenges, this work introduces MDFIHA, a novel hybrid algorithm that combines the exploration capabilities of population-based search with the exploitation strengths of local search. Its key components include: (i) a multi-depot feasible-and-infeasible search guided by a multi-penalty evaluation function and enriched with two depot-specific operators; (ii) a diversity-controlled route-exchange crossover that incorporates a learning mechanism to dynamically adjust the diversity of offspring solutions; and (iii) a fitness-and-distance-based population management strategy to maintain population health. To further address the computational challenges posed by large-scale instances, we propose an enhanced variant, MDFIHA-ETGA, which features two main improvements. First, this algorithm incorporates an efficient edge-implemented tensor-based GPU acceleration (ETGA) [LHW26], which accelerates neighborhood evaluations during local search by exploiting GPU parallelism. Second, the algorithm uses a leader-follower multi-move update strategy that applies multiple moves within a single local search iteration. This strategy leverages the parallel evaluation capabilities of GPUs more fully.

We evaluate both the CPU-based MDFIHA and the GPU-enhanced MDFIHA-ETGA algorithms using widely tested benchmark instances of three well-known multi-depot rout-

ing problems: the MDVRP, the MDVRPTW, and the MDOVRP. The results demonstrate the proposed algorithms' effectiveness and robustness across various problem variants, achieving numerous improved upper bounds on the benchmark instances: 7 new bounds for the 33 MDVRP instances, 4 and 19 new bounds for the 20 classical and 28 large-scale MDVRPTW instances, respectively, and 6 new bounds for the 24 MDOVRP instances. The GPU-enhanced MDFIHA-ETGA algorithm further improves the performance of the CPU-based MDFIHA algorithm on the 28 large-scale MDVRPTW instances, achieving improved upper bounds for 26 instances and elevating 14 of the 19 new bounds from MDFIHA.

The remainder of this chapter is organized as follows. Section 5.2 describes the proposed algorithm in detail. Section 5.3 reports computational results on benchmark instances to demonstrate the effectiveness of the method. Section 5.4 provides an in-depth analysis of the key algorithmic components. Finally, Section 5.5 offers concluding remarks and outlines potential directions for future research.

5.2 Multi-depot feasible-and-infeasible hybrid algorithm

The proposed MDFIHA algorithm for solving MDVRPs relies on the memetic search framework [MC10], which combines genetic search and local optimization. Specifically, MDFIHA incorporates a powerful multi-depot-supported feasible-and-infeasible search (MDFIS) along with a learning-driven diversity-controlled route-exchange crossover (DCREX) to balance intensification and diversification. To effectively handle the complex constraints and multi-depot characteristics of these problems, we design a multi-penalty evaluation function that guides the search process through feasible and infeasible regions and introduce two dedicated depot-related local search operators to enhance the search capability in multi-depot settings. To further improve computational efficiency for large-scale instances, we propose an enhanced version of the algorithm, referred to as MDFIHA-ETGA, which incorporates the edge-implemented tensor-based GPU acceleration [LHW26] and introduces a leader-follower multi-move update strategy to fully exploit GPU parallelism.

Algorithm 6 presents the main framework of the proposed algorithm. Given an instance $\mathcal{I}$, the algorithm first initializes a population of μ solutions and records the best solution S^* (lines 4-5). Then in each generation, an offspring solution S' is generated via DCREX

Algorithm 6 Main framework of MDFIHA

1: **Input**: Instance $\mathcal{I}$, Population size μ, Maximum number of generations φ_{max}, Stagnation patience threshold ρ, Maximum running time τ.
2: **Output**: The best solution S^*.
3: $\varphi \leftarrow 0, \varphi_{st} \leftarrow 0$ /* Current generation and stagnation counter */
4: $\mathcal{P} \leftarrow Initialization(\mathcal{I}, \mu)$ /*Initialization in Section 5.2.1 */
5: $S^* \leftarrow BestSolution(\mathcal{P})$ /* Record current best solution */
6: **while** $\varphi \leq \varphi_{max}$ **and** $\varphi_{st} \leq \rho$ **and** $time() \leq \tau$ **do**
7: $S' \leftarrow DCREX(\mathcal{P})$ /* diversity-controlled route-exchange crossover in Section 5.2.3 */
8: $S' \leftarrow MDFIS(S')$ /* Multi-depot-supported feasible-and-infeasible search in Section 5.2.2 */
9: $\mathcal{P} \leftarrow UpdatePopulation(S', \mathcal{P})$ /*Population management in Section 5.2.4 */
10: **if** $f(S') < f(S^*)$ **then**
11: $S^* \leftarrow S'$ /* Update the best solution */
12: $\varphi_{st} \leftarrow 0$
13: **else**
14: $\varphi_{st} \leftarrow \varphi_{st} + 1$ /* Stagnation counter is incremented */
15: **end if**
16: $\varphi \leftarrow \varphi + 1$
17: **end while**
18: **return** S^*

(line 7), and refined using the MDFIS procedure (line 8). The improved offspring is then added to the population using a dedicated population update strategy (line 9). According to whether S' improves on S^*, the best solution and stagnation counter are updated (lines 10-15). This process repeats until one of the stopping conditions is met (line 6): the maximum number of generations $\varphi_{\max}$, the maximum patience threshold ρ for stagnation generations φ_{st}, or the maximum runtime τ. The best solution S^* is then returned (line 18).

5.2.1 Initialization

The initialization phase generates the starting population for the algorithm. Given the complexity of MDVRPs, we adopt a “cluster-first, route-second" strategy and introduce a three-step initialization process: (i) depot assignment, (ii) route construction, and (iii) route pooling. In the first step, customers are assigned to their nearest depot based on travel distance. Next, in the route construction step, routes are built for each depot using a greedy insertion heuristic that considers travel distance, duration, and vehicle load,

while promoting diversity. Only feasible insertions are permitted; if none are possible, a new route is created. This continues until all customers are routed or the vehicle limit N_V is reached. In the final route pooling step, the constructed routes are merged into a complete solution. If any customers remain unrouted, a relaxed insertion heuristic allowing infeasible insertions is applied to ensure all customers are routed. If an initial solution is infeasible, it is still accepted and handled by subsequent procedures.

5.2.2 Multi-depot-supported feasible-and-infeasible search

The Multi-Depot-supported Feasible-and-Infeasible Search (MDFIS) is a core local search component of our proposed algorithm for search intensification (see Algorithm 7). MDFIS iteratively improves the input solution using a set of common VRP local search operators and two dedicated depot operators (line 5, Section 5.2.2), stopping when no improvement is found or a search depth threshold δ is reached (line 4). As discussed in [GH11], controlled exploration of infeasible regions can help escape local optima in complex constrained search spaces. To support this, we employ a multi-penalty evaluation function that guides the search across the neighborhoods defined by different move operators (lines 6 and 7, Section 5.2.2).

Algorithm 7 Multi-depot-supported feasible-and-infeasible search

1: **Input**: Input solution S.
2: **Output**: Improved solution S.
3: **if** ETGA is enabled, **then** initialize tensor $\mathbf{T}_s$ on GPU based on the solution S.
4: **while** improvement is found **and** search depth threshold δ is not reached **do**
5: **for** each operator O_i in a random permutation of the operator set $\mathcal{O} = \{O_1, O_2, \ldots, O_n\}$ **do**
6: **if** ETGA is enabled, **then** $\zeta \leftarrow Evaluate(\mathbf{T}_s, S, O_i)$ /* ETGA-based neighborhood evaluation */
7: **else** $\zeta \leftarrow Evaluate(S, O_i)$ /* Traditional CPU-based neighborhood evaluation */
8: **if** The move ζ is accepted **then**
9: Update current solution S based on the move ζ.
10: **if** ETGA is enabled, **then** update tensor $\mathbf{T}_s$ based on current solution S.
11: **end if**
12: **end for**
13: **end while**
14: **return** S

Finally, to address the computational challenges of large-scale instances, we adopt

the edge-implemented tensor-based GPU acceleration (ETGA) framework [LHW26] to enhance MDFIS (line 6, Section 5.2.2). ETGA leverages GPU parallelism to efficiently evaluate all neighboring solutions within the reduced neighborhood simultaneously and reduce computational time significantly on large instances. To further improve neighborhood exploration under ETGA, we additionally introduce a leader-follower multi-move update strategy (Section 5.2.2) that applies multiple independent moves within a single local search iteration. When ETGA is enabled, the move ζ (lines 6 and 8) represents a set of independent moves selected by this strategy, and additional processing steps are required (lines 3 and 10).

Multi-penalty evaluation function

$$\begin{aligned}
\text{Minimize} \quad & \mathcal{F}(S) = D(S) + \lambda_1^k \cdot V_1(S) + \lambda_2^k \cdot V_2(S) + \lambda_3^k \cdot V_3(S) + \lambda_4^k \cdot V_4(S) \qquad (5.1)\\
\text{where} \quad & D(S) = \sum_{i=1}^{M} \sum_{j=0}^{L_i} c_{n_{i,j} n_{i,j+1}} \\
& V_1(S) = \Gamma \cdot \sum_{i=1}^{M} \sum_{j=1}^{L_i} \max\{a_{n_{i,j}} - l_{n_{i,j}}, 0\} \\
& V_2(S) = \Theta \cdot \frac{\sum_{i=1}^{M} \max\{\sum_{j=1}^{L_i} q_{n_{i,j}} - \mathcal{Q}, 0\}}{\mathcal{Q}} \\
& V_3(S) = \Theta \cdot \frac{\sum_{i=1}^{M} \max\{\sum_{j=0}^{L_i} (w_{n_{i,j}} + s_{n_{i,j}} + t_{n_{i,j} n_{i,j+1}}) - \mathcal{D}, 0\}}{\mathcal{D}} \\
& V_4(S) = \Theta \cdot (\sum_{i=1}^{M} 1_{\{n_{i,0} \neq n_{i,L_{i+1}}\}} + \sum_{n_d \in \mathcal{V}_D} \max\{\sum_{i=1}^{M} 1_{\{n_{i,0} = n_d\}} - N_V, 0\}) \\
& S = \{R_1, \ldots, R_M\} \\
& R_i = \{n_{i,0}, n_{i,1}, \ldots, n_{i,L_i}, n_{i,L_i+1}\}, \quad i = 1, \ldots, M \\
& \Gamma = \frac{\sum_{e_{uv} \in \mathcal{E}} c_{uv}}{\sum_{e_{uv} \in \mathcal{E}} t_{uv}} \\
& \Theta = 2 \cdot \max c_{uv} - \min c_{uv}, \qquad e_{uv} \in \mathcal{E}
\end{aligned}$$

The feasible and infeasible search dynamically explores both feasible and infeasible regions using an evaluation function with penalty terms. While prior works (e.g., [Vid+13; Esc+14; SÇA21]) applied similar strategies for MDVRPs, they typically address only vehicle capacity, route duration, and time window violations. To better handle the diverse

constraints of MDVRPs, including multi-depot setting, we propose a unified multi-penalty evaluation function defined by Equation (5.1) that measures the quality of a given solution S during the search process. This function aims to minimize the total travel distance while integrating four normalized penalty terms corresponding to violations of time windows, vehicle capacity, maximum route duration, and depot-related constraints.

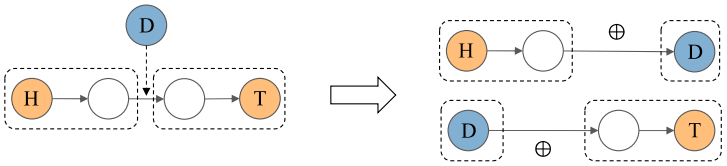
In this formulation, $D(S)$ denotes the total distance traveled by all vehicles, which is the primary objective of the MDVRPs. The terms $V_1(S)$, $V_2(S)$, $V_3(S)$, and $V_4(S)$ represent the normalized violation values for the time windows, vehicle capacity, maximum tour duration, and depot related constraints, respectively. It is flexible and can be adapted to different MDVRP variants by enabling or disabling specific terms (e.g., omitting the time window term for the basic MDVRP). The factor Γ represents the average distance-time ratio over all edges and is used to scale the time-window violation term into distance units. A scaling factor Θ is introduced to normalize the penalty terms, calculated based on the maximum and minimum edge distances. To handle violations of time window constraints, we adopt the time-warp technique [Vid+13], which allows for a smooth penalty application when time constraints are violated. Notably, the depot-related penalty $V_4(S)$ accounts for both route closure violations, penalizing the routes that do not start and end at the same depot, and depot capacity violations for exceeding available vehicles. The penalty coefficients $\lambda_1^k - \lambda_4^k$ are updated dynamically after each accepted move based on constraint feasibility. Here, k denotes the iteration count during the search. Each penalty coefficient is adjusted independently. If the associated constraint is violated, the coefficient is divided by an adjustment factor $\kappa \in (0, 1)$ (i.e., $\lambda_i^k \leftarrow \frac{\lambda_i^k}{\kappa}$) to prioritize satisfying that constraint; otherwise, it is multiplied by κ (i.e., $\lambda_i^k \leftarrow \lambda_i^k \cdot \kappa$) to encourage exploration in infeasible regions.

Move operators and evaluation

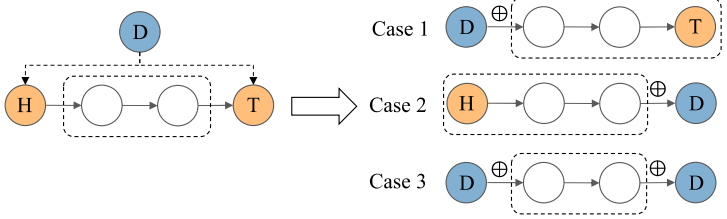
MDFIS performs route optimization with four local search operators commonly used in vehicle routing (*Relocate*, *Swap*, *2-opt**, and *2-opt*) and two proposed dedicated operators for the multi-depot setting (*Depot-Insert* and *Depot-Replace*). To efficiently evaluate move operators, we employ a sequence concatenation-based move evaluation [LHW26], enabling constant-time computation of complex constraint violations. This method treats moves as rearrangements of route subsequences and merges them using a concatenation operator $\oplus$.

Among the four common routing operators, the *Relocate* operator moves a node se-

quence within the same route or to another route. The *Swap* operator exchanges two node sequences between or within routes. In our implementation, the length of each node sequence for both *Relocate* and *Swap* is limited to 1 or 2. For symmetric problems like MDVRP, 2-node sequences can also be reversed to enhance search capacity. The *2-opt** operator works on two routes by removing one edge from each route and exchanging the resulting subsequences to create new routes. The *2-opt* operator modifies a single route by removing two non-adjacent edges and reconnecting the subsequences in reverse order. Note that *2-opt* is not applied in MDVRPTW instances, as the presence of time window constraints introduces asymmetry.



(a) *Depot-Insert*



(b) *Depot-Replace*

Figure 5.1 – Illustration of the two dedicated depot operators *Depot-Insert* and *Depot-Replace*, along with their concatenation operations. Yellow nodes represent the original depot nodes, while "H" and "T" denote the head and tail of the route, respectively. The blue "D" nodes represent the new depot being inserted or used for replacement.

To enhance performance in the multi-depot setting, we design two dedicated depot operators: *Depot-Insert* and *Depot-Replace* (see Figure 5.1). They build on the depot-related penalty term $V_4(S)$ from Equation (5.1), which allows for routes to have different departure and arrival depots. The *Depot-Insert* operator introduces a depot node into a route, splitting the original route into two separate routes. The *Depot-Replace* operator replaces the depot of a route with a new route by evaluating three cases: replacing the departure depot, replacing the arrival depot, or replacing both. Among them, the best resulting configuration is applied. The time complexity is $O(N_D \cdot (M + N_C))$ for *Depot-*

Insert and $O(N_D \cdot M)$ for *Depot-Replace*, where N_D is the number of depots, M is the number of routes, and N_C is the number of customers.

Neighborhood reduction

We adopt the best-improvement strategy for local search, which requires exploring the entire neighborhood of each operator to identify the best neighboring solution. However, exhaustively searching the neighborhood is computationally intensive, and becomes a bottleneck for large instances. To address this issue, we apply a neighborhood reduction technique to eliminate unpromising neighboring solutions.

We use two pruning strategies to reduce the neighborhood. First, based on the time window constraints, any edge $e_{ij} \in \mathcal{E}$ in the complete graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is removed if visiting v_j after v_i violates the latest allowed time, i.e., $e_i + s_i + t_{ij} > l_j$. Second, the neighborhood exploration is restricted to the θ nearest neighbors of each node. The parameter θ controls the neighborhood granularity, retaining only the θ most correlated neighbors. The neighbors are selected according to the correlation metric η_{ij} in Equation (5.2) inspired by [Vid+13; LTY21], which accounts for both spatial proximity and time-window compatibility.

$$\eta_{ij} = c_{ij} + \Gamma\Big(\alpha \cdot \max\{e_j - l_i - s_i - t_{ij}, 0\} + \beta \cdot \max\{l_i + s_i + t_{ij} - l_j, 0\}\Big) \quad (5.2)$$

$$\text{where} \quad \Gamma = \frac{\sum_{e_{uv} \in \mathcal{E}} c_{uv}}{\sum_{e_{uv} \in \mathcal{E}} t_{uv}}, \qquad v_i, v_j \in \mathcal{V}.$$

The factor Γ represents the average distance-time ratio over all edges and is used to scale time-related penalties into distance units. The constants α and β control the relative importance of early-arrival and late-arrival incompatibility, respectively. In this work, we adopt $\alpha = 1$ and $\beta = 10$ following [LTY21] to preserve the original scale of early-arrival penalties while emphasizing the greater severity of late arrivals.

Edge-implemented TGA

For large-scale instances, we adopt the edge-implemented tensor-based GPU acceleration [LHW26] to accelerate move evaluation within the reduced neighborhood by leveraging the parallel computing capabilities of GPUs.

The tensor-based GPU acceleration (TGA) framework proposed in [LHW26] speeds

up local search for vehicle routing problems, by accelerating the move evaluation for commonly used local search operators, *Relocate*, *Swap*, *2-opt**, and *2-opt*. This is achieved by tensorizing the move evaluation based on an attribute-based solution representation and exploiting massive GPU parallelism, leading to substantial computational gains over traditional CPU-based implementations of local search operators.

Specifically, the attribute-based solution representation is based on a four-dimensional tensor $\mathbf{T}_s \in \mathbb{R}^{I \times J \times K \times K}$ stored on the GPU, where I denotes the number of maintained attributes, J the maximum number of routes, and K the maximum route length. By extracting and concatenating relevant attribute tensors from the solution tensor $\mathbf{T}_s$, move evaluation can be performed through a sequence of highly parallel tensor operations on the GPU.

Building upon this framework, Lei et al. [LHW26] introduced the edge-implemented TGA to support neighborhood reduction and further decrease the computational cost of move evaluation. ETGA employs a binary mask tensor $\mathbf{M} \in \{0,1\}^{|\mathcal{V}| \times |\mathcal{V}|}$ to encode the result of a predefined neighborhood reduction strategy. Each element $\mathbf{M}_{ij}$ indicates whether a valid edge exists between nodes v_i and v_j. By using this mask, ETGA restricts tensor extraction and move evaluation to valid node pairs only, thereby avoiding unnecessary computations over the full neighborhood and significantly improving efficiency.

In this work, we adopt ETGA to implement the neighborhood reduction strategy described in Section 5.2.2 to further improve the computational efficiency of the local search, particularly for large-scale instances. Owing to the flexibility and extensibility of its attribute-based solution tensor representation, ETGA can be readily adapted to support the proposed multi-penalty evaluation function for MDVRPs by incorporating the required attributes. Note that ETGA is applied only to the four local operators (*Relocate*, *Swap*, *2-opt**, and *2-opt*); the two proposed depot-specific operators (*Depot-Insert* and *Depot-Replace*) remain CPU-based due to their low computational complexity.

Leader-follower multi-move update strategy

As discussed in Section 5.2.2, ETGA exploits GPU parallelism to evaluate all candidate moves within a reduced neighborhood simultaneously. However, the original implementation of ETGA [LHW26] updates the current solution by applying only the single best move in each local search iteration, similar to the conventional CPU-based local search. Consequently, a significant portion of the evaluated moves go unused, and the available parallelism remains underutilized.

To better leverage GPU parallelism, several studies on GPU-based local search for the TSP [QC18; QC20; Rob+18] have applied multiple moves per iteration. Motivated by these studies, we introduce a multi-move update strategy within the ETGA framework to further enhance the efficiency of the proposed local search component MDFIS. However, unlike the TSP considered in [QC18; QC20; Rob+18], MDVRPs involve more complex constraints, while MDFIS explores both feasible and infeasible regions guided by a multi-penalty evaluation function, making it difficult to directly adopt existing multi-move strategies.

Therefore, we propose a leader-follower multi-move update strategy specifically tailored to the proposed MDFIS under the ETGA framework. The *leader* move is defined as the single best move identified according to the evaluation function, which is consistent with the traditional single-move local search. The *follower* moves are selected from the remaining candidates that satisfy two conditions: (i) a strict decrease in travel distance (i.e., $\Delta D < 0$), and (ii) no variation in any constraint-related penalty term (i.e., $\Delta V_i = 0$, $i = 1, 2, 3, 4$). Identifying the leader and follower moves can be efficiently performed on the GPU during move evaluation. Then, all candidate moves are transferred to the CPU for selection.

The move selection process proceeds as follows. The leader move is always applied to maintain consistency with single-move local search behavior. The follower moves are sorted in ascending order of their evaluation scores and selected greedily, provided that they do not conflict with any previously selected moves. Two moves are considered conflicting if they involve the same route. Through this process, a set of independent moves is identified and applied simultaneously to update the current solution in each local search iteration.

By evaluating and applying multiple independent moves within a single iteration, the proposed multi-move update strategy more effectively exploits GPU parallelism under the ETGA framework and enhances overall search efficiency, particularly for large-scale instances. Moreover, retaining the best move as the leader while restricting follower moves to those that improve travel distance without increasing constraint violations ensures that the search process remains well guided by the multi-penalty evaluation function, even when multiple moves are applied concurrently.

5.2.3 Diversity-controlled route-exchange crossover

Unlike giant-tour-based crossover approaches [Pri04; Vid+12; Vid+13], which require a carefully designed and efficient split procedure, or edge-assembly-crossover-based meth-

ods [Nag97; HH23a; HH23b], which face challenges in handling complex multi-depot problems, such as MDVRPTW, where time windows introduce asymmetry, this work adopts a route-based crossover paradigm that provides more flexibility for multi-depot settings and complex constraints.

The proposed Diversity-Controlled Route-Exchange Crossover (DCREX) generates promising offspring solutions from multiple parent solutions through route exchanges guided by a learning-driven diversity control mechanism. DCREX is inspired by the Adaptive Route Inheritance Crossover (ARIX) introduced in [LH25], which was originally developed for the vehicle routing problem with simultaneous pickup and delivery and time windows (VRPSPDTW). Although ARIX incorporates strategies tailored to the characteristics of VRPSPDTW, its underlying principles, including route-based multi-parent crossover and learning-based decision mechanisms, are broadly applicable across different VRP variants.

Building on these general ideas, we design DCREX as a generalized route exchange crossover with adaptive diversity control, which also supports multi-depot settings in MDVRPs. The objective is to achieve a better balance between solution quality and population diversity during the evolutionary search process. Specifically, DCREX generates an offspring solution by selecting one parent as the main parent and selectively replacing a subset of its routes with routes inherited from other parent solutions, with the route exchange process guided by a learning-based diversity control mechanism.

General crossover procedure

As shown in Algorithm 8, the DCREX crossover uses a multi-parent design to increase solution diversity, while the number of parents is determined adaptively. A random parent is first selected as the main parent P_m (line 3), and the offspring S' is initialized as a copy of P_m (line 4). A target diversity degree σ_i is then determined via a learning mechanism (line 5, see next section) from 20 discrete levels. The corresponding diversity bounds $\sigma_{\min}$ and $\sigma_{\max}$ are computed (lines 6-7), and the current diversity σ_{cur} is set to zero (line 8). DCREX then iterates over other parents to exchange routes (lines 9-17). For each candidate parent P, DCREX scores route pairs $\{R_i, R_j\}$, where R_i is from P_m and R_j from P (line 10). The score accounts for four factors: (i) newly introduced edges E_{ij}^I, (ii) missing nodes N_{ij}^R, (iii) redundant nodes N_{ij}^M appearing more than once, and (iv) conflicting nodes N_{ij}^C that already exist in the introduced routes. Each factor is weighted to reflect its relative importance, with conflicting nodes heavily penalized to avoid duplicates. Lower scores

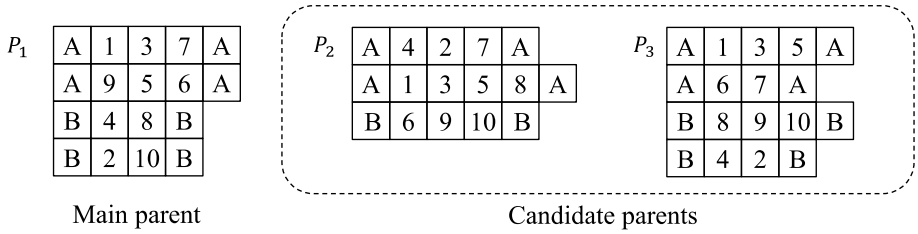
Algorithm 8 Diversity-controlled route-exchange crossover

1: **Input**: Population $\mathcal{P}$.
2: **Output**: Offspring solution S'.
3: Let P_m be the main parent randomly selected from $\mathcal{P}$ /* Main parent */
4: $S' \leftarrow P_m$ /* Initialize offspring using main parent */
5: Determine diversity degree $\sigma_i = \frac{i}{20}, i \in \{0, 1 \cdots, 19\}$ based on the learning mechanism /* Section 5.2.3 */
6: $\sigma_{min} = \sigma_i \cdot (N_C + M_{P_m})$ /* Calculate the diversity bounds σ_{min} and σ_{max} based on number of
7: $\sigma_{max} = \sigma_{i+1} \cdot (N_C + M_{P_m})$ customer nodes N_C and number of routes M_{P_m} in main parent P_m*/
8: $\sigma_{cur} = 0$ /* Initialize current diversity value */
9: **for** $P \in \mathcal{P} \setminus \{P_m\}$ **do**
10: $Score_{ij} = -(E^I_{ij} - N^R_{ij} - N^M_{ij} - 10 \cdot N^C_{ij})$ /* Calculate the score of each route pair $\{R_i, R_j\}, R_i \in P_m, R_j \in P$ */
11: Randomly select $\{R^*_i, R^*_j\}$ from the 5 lowest-scored route pairs
12: $S' \leftarrow (S' \setminus R^*_i) \cup R^*_j$ /* Remove the route R^*_i and add the route R^*_j */
13: $\Delta\sigma = E^I_{ij} + N^R_{ij} + N^M_{ij}$ /* Calculate the variation of diversity */
14: $\sigma_{cur} = \sigma_{cur} + \Delta\sigma$ /* Update the current diversity value */
15: **if** $\sigma_{cur} \geq \sigma_{max}$ **then** Continue to next parent
16: **else if** $\sigma_{cur} \geq \sigma_{min}$ **then** Break the loop
17: **end for**
18: Remove redundant nodes in S'
19: $U \leftarrow \{n \in \mathcal{V} | n \notin S'\}$ /* Unrouted nodes */
20: Insert the unrouted nodes in U into S' using the insertion operator $\boxdot$
21: **return** S'

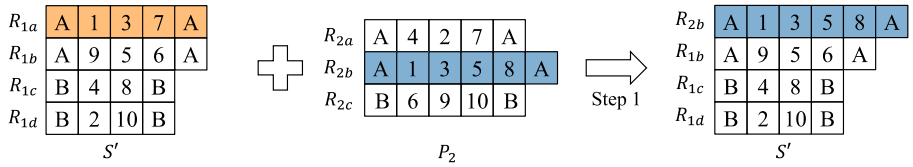
indicate better route pairs. One of the top five ranked pairs $\{R^*_i, R^*_j\}$ is selected randomly (line 11), and R^*_i in S' is replaced by R^*_j (line 12). The resulting diversity variation $\Delta\sigma$ is computed (line 13), and σ_{cur} is updated (line 14). If σ_{cur} exceeds the upper bound σ_{max}, indicating the current parent cannot provide suitable routes, the algorithm continues with the next parent (line 15). If σ_{cur} exceeds the lower bound σ_{min}, meaning the desired diversity degree has been reached, the loop breaks (line 16).

After the route exchange, the repair procedure [LH25] is applied to ensure all customers are properly routed (lines 18-20). Specifically, redundant nodes are removed from the offspring S' (line 18), and unrouted nodes are inserted using an insertion operator $\boxdot$ (lines 19-20), selected via a learning mechanism (see next section) among five operators. Feasible best insertion (FBI) and infeasible best insertion (IBI) both prioritize minimal cost increase; FBI creates a new route if no feasible insertion is possible, while IBI allows

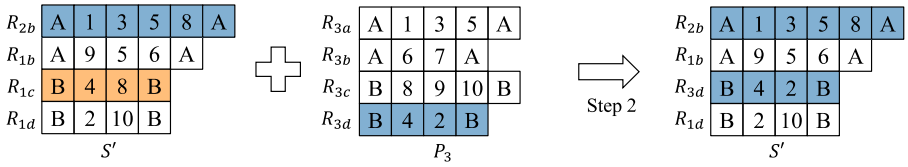
infeasible insertions without adding new routes. Feasible regret insertion (FRI) and infeasible regret insertion (IRI) evaluate each insertion based on the regret value, defined as the difference in travel cost between the best and second-best insertions. Random insertion (RI) inserts an unrouted customer randomly into a randomly selected route. If the resulting offspring is still infeasible, it is then addressed and refined by the MDFIS procedure (Section 5.2.2). In the worst case, the time complexity of DCREX is $\mathcal{O}(|\mathcal{P}| \cdot M^2)$, where $|\mathcal{P}|$ denotes the number of solutions in the population and M represents the average number of routes per solution.



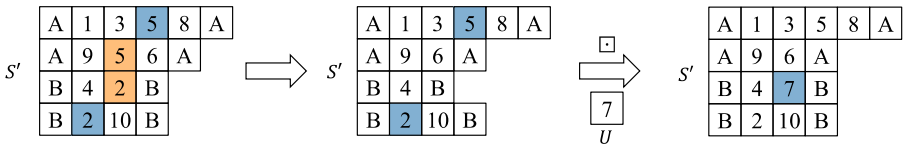
(a) Main parent P_1 and candidate parents P_2 and P_3.



(b) Selected route pair $\{R_{1a}, R_{2b}\}$ with score computed as $-(E^I - N^R - N^M - 10 \cdot N^C) = -(3-2-1-0) = 0$, and diversity increase $\Delta\sigma = E^I + N^R + N^M = 6$.



(c) Selected route pair $\{R_{1c}, R_{3d}\}$ with score computed as $-(E^I - N^R - N^M - 10 \cdot N^C) = -(1-0-0-0) = -1$, and diversity increase $\Delta\sigma = E^I + N^R + N^M = 1$.



(d) Redundant nodes removal and unrouted nodes insertion.

Figure 5.2 – Illustration of the diversity-controlled route-exchange crossover. “A" and “B" are depot nodes. Blue indicates the routes added or nodes retained, while yellow represents the routes or nodes that are removed.

Figure 5.2 shows a step-by-step example of the diversity-controlled route-exchange crossover (DCREX) proposed in this work. This illustration uses three parent solutions,

where P_1 serves as the main parent P_m, while P_2 and P_3 serve as candidate parents (Figure 5.2a). The diversity degree is set to $\sigma_{10} = 0.5$, with corresponding diversity value bounds $\sigma_{\min} = 7.0$ and $\sigma_{\max} = 7.7$. The offspring S' first exchanges a route with P_2 (Figure 5.2b), yielding a diversity increase of $\Delta\sigma = 6$. Since further exchanges with P_2 would exceed $\sigma_{\max}$, P_3 is then considered (Figure 5.2c), resulting in a second exchange with $\Delta\sigma = 1$, reaching the diversity target $\sigma_{\min} = 7.0$. Finally, redundant nodes 2 and 5 are removed, and the unrouted node 7 is inserted using the selected insertion operator $\boxdot$ (Figure 5.2d), producing the final offspring S'.

Learning-based decision mechanism

During the crossover process, DCREX needs to determine the diversity degree and the insertion operator $\boxdot$ for unrouted nodes. For this purpose, it uses a learning mechanism based on the discounted UCB1 algorithm from the multi-armed bandit framework to inform these decisions. Unlike the standard UCB1 used in [LH25], the discounted variant uses a discount factor ($\gamma_d = 0.99$) to prioritize recent rewards and limit the impact of outdated data.

$$\begin{aligned} \mathrm{UCB1}_i(t) &= \frac{\mathcal{R}_i^d(t)}{N_i(t)} + \sqrt{\frac{2\ln\left(\sum_{j=1}^{|\mathcal{A}|} N_j(t)\right)}{N_i(t)}} \\ \text{where} \quad \mathcal{R}_i^d(t) &= \sum_{\tau=1}^{t} \gamma_d^{t-\tau} \mathcal{R}_i(\tau) \cdot 1_{\{a_\tau = i\}} \\ N_i(t) &= \sum_{\tau=1}^{t} \gamma_d^{t-\tau} \cdot 1_{\{a_\tau = i\}} \end{aligned} \tag{5.3}$$

For each action $a_i \in \mathcal{A}$, the discounted UCB1 algorithm computes the upper confidence bound $\mathrm{UCB1}_i(t)$ at step t (Equation (5.3)) using its discounted reward $\mathcal{R}_i^d$ and selection count N_i. The action with the highest value is selected at each step. In our case, the actions $\mathcal{A}$ includes 20 diversity degrees $\{0, 1, \ldots, 19\}$ and the five insertion operators presented in Section 5.2.3. The reward $\mathcal{R}$ for each action is the improvement ratio of the offspring S' over the main parent P_m, defined as $\mathcal{R} = 100 \times \frac{f(P_m) - f(S')}{f(P_m)}$.

5.2.4 Population management

During the execution of the algorithm, the population $\mathcal{P}$ is expanded with new offspring solutions to the size of 1.5μ for $\frac{\mu}{2}$ generations [Vid+12]. $\mathcal{P}$ is then reduced back to

its original size μ by removing the worst solutions according to the fitness-and-distance-based scoring function χ defined by Equation (5.4).

$$\chi(S) = \frac{|\mathcal{P}| - \text{Rank}_f(S)}{|\mathcal{P}|} + \xi \cdot \frac{|\mathcal{P}| - \text{Rank}_{D_\mathcal{P}}(S)}{|\mathcal{P}|} \tag{5.4}$$

$$\text{where} \quad D_\mathcal{P}(S) = \frac{1}{5} \sum_{S_i \in \mathcal{N}_5(S)} D_S(S, S_i), \quad S_i \neq S \tag{5.5}$$

$$D_S(S_1, S_2) = 100 \cdot \left(1 - \frac{|\mathcal{E}_{S_1} \cap \mathcal{E}_{S_2}|}{\max\{|\mathcal{E}_{S_1}|, |\mathcal{E}_{S_2}|\}}\right), \quad \mathcal{E}_{S_1}, \mathcal{E}_{S_2} \in \mathcal{E} \tag{5.6}$$

The scoring function χ assigns a fitness score to each solution S in the population $\mathcal{P}$ by considering two factors: the rank of S with respect to the objective value $\text{Rank}_f(S)$ and its rank based on the distance within the population $\text{Rank}_{D_\mathcal{P}}(S)$. The parameter ξ balances the trade-off between solution quality and diversity. A low value of $\chi(S)$ indicates a solution of low fitness.

To quantify diversity, we measure the distance between solutions using Equation (5.6), where $\mathcal{E}_{S_1}$ and $\mathcal{E}_{S_2}$ represent the sets of edges in solutions S_1 and S_2, respectively. The distance of a solution S from the population $\mathcal{P}$ is defined as the average distance to its 5 nearest neighbors in $\mathcal{P} \setminus \{S\}$, denoted by $\mathcal{N}_5(S)$, as shown in Equation (5.5).

5.2.5 Discussion

The proposed MDFIHA algorithm introduces several novel components to address the specific challenges of multi-depot vehicle routing problems.

First, unlike the “cluster-first, route-second" strategy commonly adopted in existing MDVRP studies [GTV02; TV06; Tor+22], which treats depot assignment and routing as two separate phases, MDFIHA is built upon the principle of jointly optimizing both aspects within a unified framework. This is achieved through its local search component MDFIS (Section 5.2.2), based on a multi-penalty evaluation function and dedicated depot-related local search operators. Although penalty-based local search has been widely used in the literature [Vid+13; Esc+14; SÇA21], existing approaches typically do not explicitly model depot-related violations. In MDFIS, the proposed multi-penalty evaluation function includes a dedicated term for depot-related violations, thereby explicitly accounting for depot assignment and route closure constraints. Furthermore, two specialized depot operators, namely *Depot-Insert* and *Depot-Replace*, are introduced to directly modify depot assignments during the search process. By allowing temporary route-closure inconsistencies

and enabling depot-changing moves, MDFIS explores neighborhoods that simultaneously modify both routing and depot structures within a single local optimization procedure.

Second, to address the computational challenges of neighborhood exploration, particularly for large-scale instances, MDFIHA incorporates an edge-implemented tensor-based GPU acceleration technique [LHW26] to accelerate move evaluation. Unlike CPU-based parallel methods [CM12], ETGA offloads the most computationally intensive components of neighborhood evaluation to the GPU and exploits its massive parallelism to achieve substantial efficiency gains. In this work, ETGA is extended to support multi-depot settings and is integrated into the core local search operators of MDFIHA, marking the first ETGA application to these MDVRPs. This integration significantly improves the computational efficiency and enhances the scalability of the proposed algorithm when solving large-scale MDVRP instances. Furthermore, MDFIHA-ETGA introduces a novel leader-follower multi-move update strategy within the ETGA framework. This strategy maximizes GPU utilization and enhances the search capability by applying multiple independent moves simultaneously in each local search iteration.

Third, MDFIHA's crossover component DCREX (Section 5.2.3) incorporates learning-based diversity control at the route-exchange level to help improve solution quality and maintain population diversity during the search process. The underlying route-based crossover paradigm offers greater flexibility in handling multi-depot settings and complex constraints than giant-tour-based methods [Pri04; Vid+12; Vid+13], which require an efficient split algorithm to decode the resulting giant-tour into a route solution after applying a simple crossover, such as ordered crossover, on a giant-tour representation. Our DCREX crossover also differs from the popular edge-assembly-crossover approaches, such as those described in [Nag97; HH23a; HH23b], which have proven effective for the TSP and certain symmetric VRP variants. However, these approaches face difficulties when applied to complex MDVRPs, especially MDVRPTW, where time windows introduce asymmetry. Our route-based DCREX naturally accommodates depot assignments and problem-specific constraints by directly exchanging entire routes between parent solutions, making it particularly well suited to multi-depot scenarios.

Finally, DCREX is inspired by the general ideas of ARIX [LH25], including route-based multi-parent crossover and learning-based decision mechanisms. However, DCREX is designed as a more general and flexible crossover approach. Unlike ARIX, which relies on mechanisms specifically designed for the hierarchical objective structure of the VRPSPDTW, including adaptive parent selection and route inheritance control, DCREX

employs a learning-based strategy to dynamically determine the target diversity level of offspring solutions. To support this process, DCREX introduces a dedicated diversity scoring mechanism and employs an adaptive multi-parent design driven by diversity bounds. As a result, DCREX enables diversity control while remaining applicable to a broader class of routing problems.

5.3 Computational results

This section presents computational results for MDFIHA and MDFIHA-ETGA on standard benchmarks, which are introduced in Section 1.3.3 and compares them with state-of-the-art methods.

5.3.1 Experimental protocol and parameter tuning

Both MDFIHA and its GPU-enhanced version MDFIHA-ETGA are implemented in C++. MDFIHA-ETGA leverages edge-implemented tensor-based GPU acceleration via the PyTorch C++ API. Experiments were conducted on a computer with an AMD EPYC 7282 2.8 GHz CPU. For MDFIHA-ETGA, an NVIDIA V100 GPU with 32 GB memory was used.

Table 5.1 – Parameter tuning results.

Parameter	Description	Candidate values	Final value
μ	population size	{10, 20, 30, 40, 50}	20
θ	granularity threshold	{25, 50, 75, 100}	50
δ	local search depth	{100, 200, 300, 400, 500}	500
κ	coefficient adjustment factor	(0, 1)	0.5
ξ	fitness function coefficient	(0, 1)	0.7

To balance solution quality and computational efficiency, several parameters were tuned empirically. The maximum number of generations φ_{max} was set to 5000, and the stagnation patience ρ was set to 500. Five critical parameters, including population size μ, granularity threshold θ, local search depth δ, coefficient adjustment factor κ, and fitness function coefficient ξ, were tuned using the automatic parameter configuration tool Irace [Lóp+16], with 10 randomly selected instances and a tuning budget of 1000. Table 5.1 summarizes the candidate ranges and best values. These settings were used consistently in all experiments. In the experiments, each instance was solved 10 times independently,

except for the MDVRPTW *V13* instances, which were solved 5 times in accordance with [Vid+13].

5.3.2 Reference algorithms

For each benchmark set, we compare our algorithm's results with the best-known solutions (BKS) and the state-of-the-art algorithms that have contributed to achieving these BKS (see Table 5.2). Note that our MDFIHA algorithm is tested on all sets, while the GPU-enhanced MDFIHA-ETGA algorithm is only applied to the large-scale *V13* instances.

Table 5.2 – Reference algorithms for benchmark sets.

Reference algorithm	Approach	Problems	Benchmarks
CGL [CGL97]	Tabu Search	MDVRP	*C97*
PR [PR07]	Adaptive Large Neighborhood Search	MDVRP	*C97*
VCGLR [Vid+12]	Hybrid Genetic Search	MDVRP	*C97*
ELTG [Esc+14]	Hybrid Granular Tabu Search	MDVRP	*C97*
VCGP [Vid+13]	Hybrid Genetic Search	MDVRPTW	*C01*, *C01-R*, *V13*
JB [Bra20]	Iterated Local Search	MDOVRP	*L14*
RM [LM21]	Exact method	MDOVRP	*L14*
SCA [SÇA21]	Variable Tabu Neighborhood Search	MDVRP, MDVRPTW, MDOVRP	*C97*, *C97-T*, *C01*, *L14*
MDFIHA (Ours)	Memetic Algorithm	MDVRP, MDVRPTW, MDOVRP	*C97*, *C97-T*, *C01*, *C01-R*, *V13*, *L14*
MDFIHA-ETGA (Ours)	Memetic Algorithm with GPU acceleration	MDVRPTW	*V13*

To account for hardware differences, Table 5.3 reports CPU benchmark scores PassMark (`https://www.passmark.com/`) and conversion ratios (γ) to normalize reported computation times of reference algorithms with our AMD EPYC 7282 as the baseline. The CPU information for the Sun Sparcstation 10 used to run CGL [CGL97] is unavailable (indicated by -).

Table 5.3 – Processor information and conversion ratio.

Reference algorithm	Approach	Processor	Base frequency	CPU mark	γ
CGL [CGL97]	Tabu Search	Sun Sparcstation 10	-	-	-
PR [PR07]	Adaptive Large Neighborhood Search	Pentium 4	3.0 GHz	528	0.27
VCGLR [Vid+12]	Hybrid Genetic Search	Pentium 4	3.0 GHz	528	0.27
ELTG [Esc+14]	Hybrid Granular Tabu Search	Intel Core Duo	2.0 GHz	608	0.31
VCGP [Vid+13]	Hybrid Genetic Search	Intel Xeon	2.93 GHz	1404	0.72
JB [Bra20]	Iterated Local Search	Intel Core i7-3820	3.60 GHz	1744	0.90
RM [LM21]	Exact method	Intel Core i7	3.70 GHz	1960	1.01
SCA [SÇA21]	Variable Tabu Neighborhood Search	Intel Core i7-8700	3.20 GHz	2631	1.35
MDFIHA (Ours)	Memetic Algorithm	AMD EPYC 7282	2.80 GHz	1945	1.00
MDFIHA-ETGA (Ours)	Memetic Algorithm with GPU acceleration	AMD EPYC 7282, NVIDIA V100			1.00

5.3.3 Computational results and comparisons

Table 5.4 summarizes the comparative results of the proposed algorithm against various reference algorithms across different benchmark instance sets. It specifies the problem type, the benchmark set, and the number of instances (*#Instances*). For each algorithm, the table reports the average best objective function value ($\bar{f}_{Best}$), the average computation time ($\bar{t}$), the normalized average computation time ($\gamma \cdot \bar{t}$) and the average gap relative to BKS ($\overline{\text{Gap}}$). The gap is calculated as $\text{Gap}(\%) = \frac{f-\text{BKS}}{\text{BKS}} \times 100$, where a negative value indicates that the algorithm has found a new improved upper bound. The number of wins (*#Wins*), ties (*#Ties*), and losses (*#Losses*) indicates cases where the MDFIHA performs better, equally, or worse than the reference algorithm, respectively. To assess statistical significance, the Wilcoxon signed-rank test [Wil92] was applied at a 0.05 confidence level, with the resulting p-values also included in the table. The symbol '-' denotes unavailable results. Detailed results for all benchmark sets are provided in Tables C.14-C.19 in the appendix.

Table 5.4 – Summary of comparative results on benchmarks between the proposed algorithms and the reference algorithms.

Problem	Benchmark	#Instances	Algorithm	$\bar{f}_{Best}$	$\bar{t}$ (s)	$\gamma \cdot \bar{t}$ (s)	$\overline{\text{Gap}}$ (%)	#Wins	#Ties	#Losses	p-value
MDVRP	*C97*	33	BKS	2423.02	-	-	0.00	7	23	3	0.047
			CGL	2455.56	1112.27	-	1.16	25	8	0	1.23E-05
			PR	2428.30	236.61	63.88	0.21	15	18	0	9.82E-04
			VCGLR	2425.22	254.30	68.66	0.10	8	25	0	0.012
			ELTG	2430.12	138.82	43.03	0.29	11	19	3	3.51E-03
			SCA	2430.86	217.37	293.45	0.20	15	17	1	6.43E-04
			MDFIHA	2421.33	229.61	229.61	-0.09	-	-	-	-
	C97-T	10	Optima*	1879.31	-	-	0.00	0	10	0	-
			SCA	1881.16	380.64	513.86	0.08	3	7	0	0.109
			MDFIHA	1879.31	382.19	382.19	0.00	-	-	-	-
MDVRPTW	*C01*	20	BKS	2224.42	-	-	0.00	4	15	1	0.138
			VCGP	2224.43	389.58	280.50	0.00	5	14	1	0.075
			SCA	2225.91	1022.86	1380.86	0.05	9	11	0	0.012
			MDFIHA	2223.61	732.71	732.71	-0.03	-	-	-	-
	C01-R	10	VCGP	2021.60	451.98	325.43	0.00	9	1	0	0.008
			MDFIHA	1988.86	554.29	554.29	-2.14	-	-	-	-
	V13	28	VCGP	8779.76	5338.86	3843.98	0.00	19	0	9	0.048
			MDFIHA	8765.20	6352.19	6352.19	-0.22	-	-	-	-
			MDFIHA-ETGA †	8722.38	6490.46	6490.46	-0.60	26	0	2	6.56E-07
MDOVRP	*L14*	24	BKS	1398.69	-	-	0.00	6	17	1	0.028
			JB	1402.74	64.57	58.11	0.22	15	9	0	6.55E-04
			RM	1411.14	2551.24	2576.75	0.51	7	17	0	0.018
			SCA	1399.38	141.44	190.94	0.03	6	17	1	0.028
			MDFIHA	1397.39	78.99	78.99	-0.06	-	-	-	-

*Optimal solutions from [SUP21] are used as references for the *C97-T* instances.
†The comparisons and p-values for MDFIHA-ETGA refer to its performance relative to VCGP.

On the MDVRP *C97* instances, MDFIHA achieves an average gap of -0.09%, establishes 7 new best upper bounds, and matches the BKS for 23 of the remaining 26 instances. While some older algorithms show shorter running times, especially after time conversion,

MDFIHA consistently delivers better solution quality across most instances, and it outperforms the recent algorithm SCA [SÇA21] in both quality and computation time. On the *C97-T* instances with tighter fleet-size limits, MDFIHA reaches all 10 known optimal solutions with shorter computation times than SCA.

On the well-solved MDVRPTW *C01* instances, MDFIHA achieves an average gap of -0.03%, establishes 4 improved upper bounds, and again outperforms the recent SCA. Against the high-performing VCGP algorithm [Vid+13], MDFIHA achieves better solution quality on 5 instances and it also finds 9 better solutions out of 10 *C01-R* instances with relaxed fleet-size limits.

On the 28 large-scale and challenging MDVRPTW *V13* instances, despite using a shorter time limit (7200 seconds) compared to VCGP's 5-hour limit (18000 seconds, converted to 12960 seconds), MDFIHA achieves an average gap of -0.22% and improves 19 best-known solutions. The GPU-enhanced MDFIHA-ETGA delivers even stronger results, with an average gap of -0.60% and 26 improved upper bounds, highlighting its scalability and efficiency when addressing large-scale instances.

Finally, MDFIHA outperforms all three reference algorithms on the MDOVRP *L14* instances, achieving an average gap of -0.05%, reporting 6 improved upper bounds within short computation times. These results further demonstrate the effectiveness of the proposed algorithm across different problem variants.

Additionally, most associated p-values are below 0.05, indicating that MDFIHA statistically outperforms both the BKS and the reference algorithms. Exceptions on the *C97-T* and *C01* sets are due to small benchmark sizes and the presence of known optimal solutions or well-solved instances, for which no further improvement is possible.

5.4 Performance analysis

This section analyzes the impact of DCREX crossover and MDFIS procedure on the performance of the proposed algorithm on the 73 benchmark instances of *C97*, *C97-T*, *C01*, and *C01-R*. To evaluate the impact of GPU acceleration within the ETGA framework, we use the 28 large-scale *V13* instances.

5.4.1 Analysis on diversity-controlled route-exchange crossover

To investigate the impact of the DCREX crossover, we construct several variants based on the original MDFIHA algorithm as the baseline:

- MDFIHA-RDP: Remove DCREX and use a single-individual population. Each generation applies a random destroy-and-repair operator followed by local optimization with MDFIS.
- MDFIHA-CDP: The same as MDFIHA-RDP, but we use a correlated destroy-and-repair operator based on the correlation metric from Section 5.2.2.
- MDFIHA-SREX: Replace DCREX with the selective route exchange crossover (SREX) [NK10], which swaps correlated route clusters between parents.
- MDFIHA-RARI: Replace DCREX with the route assembly regret insertion crossover (RARI) [LTY21], which iteratively inherits routes from two parents and then repairs the offspring.

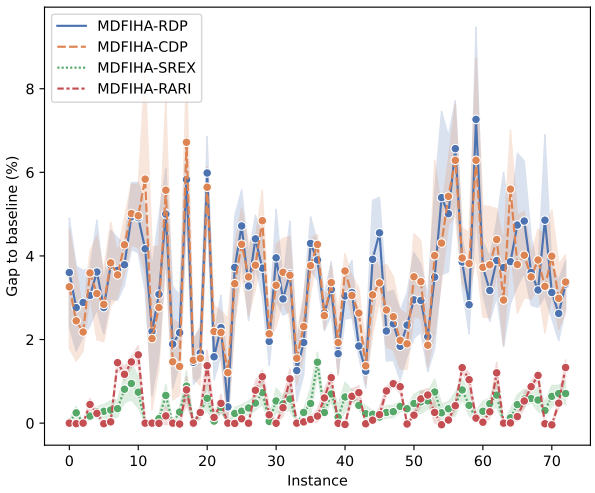
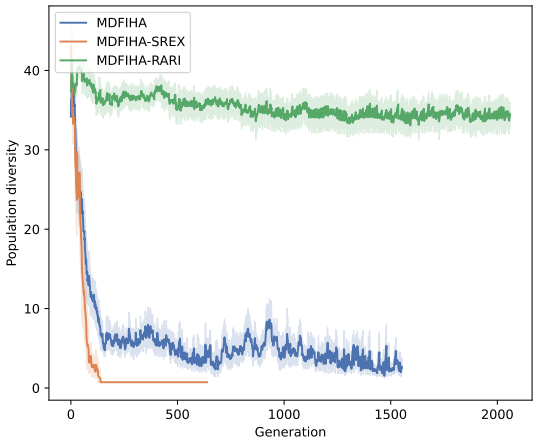


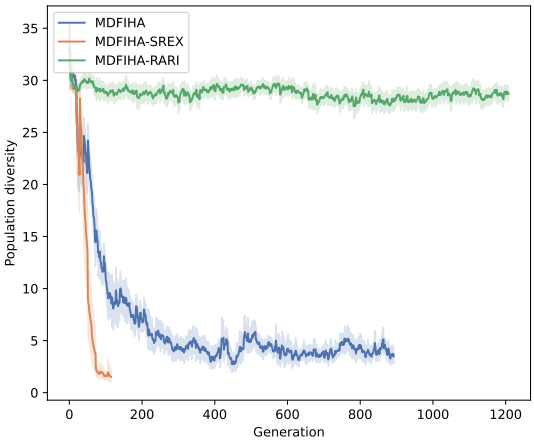
Figure 5.3 – Performance comparison of MDFIHA and its variants. The x-axis indicates instances, and the y-axis shows the relative gap to the baseline MDFIHA. Colored lines denote mean gaps with 0.95 confidence intervals.

Figure 5.3 compares MDFIHA with its variants using different destroy-and-repair and crossover strategies. MDFIHA-RDP and MDFIHA-CDP, which rely on single-solution destroy-and-repair methods, perform significantly worse, highlighting the advantage of population-based search. While MDFIHA-SREX and MDFIHA-RARI outperform the destroy-and-repair-based variants, they still lag behind the original MDFIHA, confirming the effectiveness of the DCREX crossover.

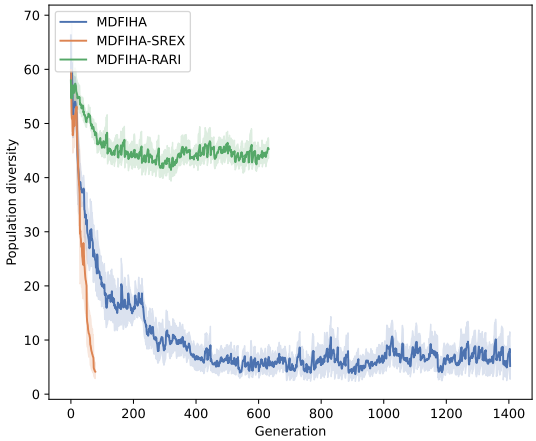
To better understand the role of DCREX, we examine its impact on population diversity throughout the search process. Figure 5.4 illustrates the evolution of population diversity for four representative instances. In all cases, MDFIHA-SREX exhibits a rapid decrease in diversity, resulting in premature convergence. Although MDFIHA-RARI maintains a high level of diversity all the time, it fails to converge to high-quality solutions. In contrast, the proposed DCREX crossover enables faster convergence in the early stages while preserving a moderate level of diversity later on. This helps the algorithm to balance exploration and exploitation more effectively, leading to better overall results.



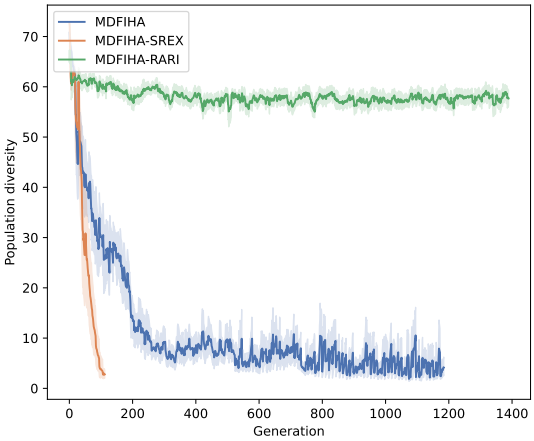
(a) Instance *p11* from *C97*.



(b) Instance *p21* from *C97*.



(c) Instance *pr05* from *C01*.



(d) Instance *pr15* from *C01*.

Figure 5.4 – Evolution of population diversity across generations for MDFIHA and its variants MDFIHA-SREX and MDFIHA-RARI. The diversity is measured by the solution distances within population.

5.4.2 Analysis on multi-depot-supported feasible-and-infeasible search

To study the impact of the multi-depot-supported feasible-and-infeasible search, we construct the following MDFIHA variants.

- MDFIHA-F: Accept only feasible solutions during the search.
- MDFIHA-PC10: Retain feasible and infeasible search with a fixed penalty coefficient of 10.
- MDFIHA-wo-DODT: Remove *Depot-Insert* and *Depot-Replace* operators and excludes the depot-related penalty from the evaluation function (Equation (5.1)).

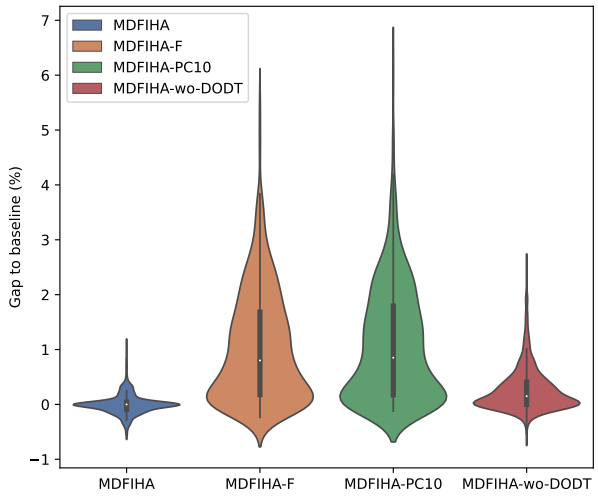
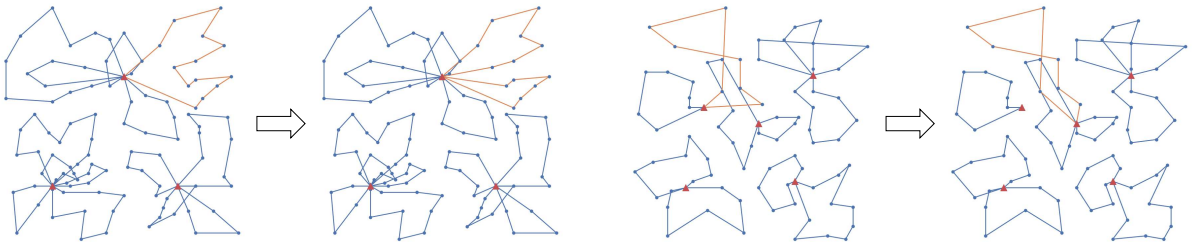


Figure 5.5 – Performance comparison of MDFIHA and its variants. The x-axis indicates algorithm variants, and the y-axis shows the relative gap to the baseline MDFIHA.

According to the violin plots in Figure 5.5, MDFIHA-F and MDFIHA-PC10 perform significantly worse than the baseline MDFIHA, with notably higher median gaps, greater variance and more extreme outliers. This suggests that either excluding infeasible solutions or using fixed penalty coefficients reduces the effectiveness of the search process. Similarly, MDFIHA-wo-DODT also shows degraded performance, which confirms the importance of depot-related components in improving the quality of solutions in multi-depot problems.

Additionally, to clarify the roles of the depot operators, Figure 5.6 illustrates the actions of *Depot-Insert* and *Depot-Replace* during the search. *Depot-Insert* is observed to effectively split a route into two, enabling rapid transitions in the search space, while *Depot-Replace* operator moves an entire route to a different depot, which is essentially a depot reallocation. These two operators jointly enhance the algorithm's ability to explore



(a) A *Depot-Insert* move on instance *p06* from *C97*.

(b) A *Depot-Replace* move on instance *p03* from *C97*.

Figure 5.6 – Illustration of depot-related operators. Red triangles denote depots, blue points represent customers. Blue edges indicate unchanged routes, yellow edges highlight modified routes.

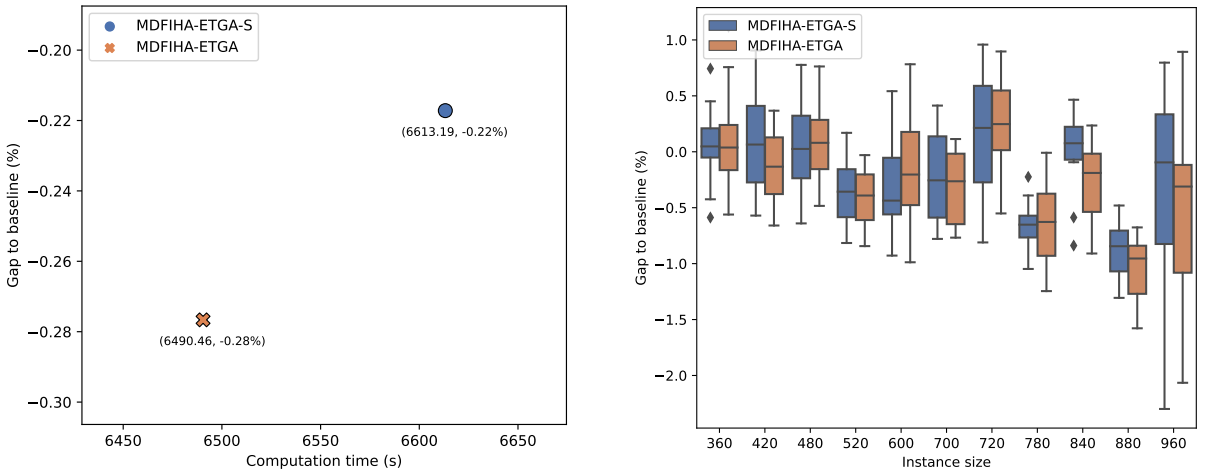
the multi-depot solution space more thoroughly by simultaneously considering both depot assignments and vehicle routings.

5.4.3 Analysis of GPU parallelism benefits

To evaluate the benefits of GPU parallelism, including the computational efficiency and speedup achieved by the edge-implemented tensor-based GPU acceleration, as well as the performance gains from the proposed leader-follower multi-move update strategy, we conduct additional experiments on the 28 large-scale MDVRPTW *V13* instances.

First, we compare MDFIHA-ETGA and MDFIHA-ETGA-S under the same experimental settings as set in Section 5.3.1, where MDFIHA-ETGA-S is a variant that adopts a conventional single-move mechanism without the multi-move strategy of Section 5.2.2. This comparison aims to isolate and assess the contribution of the proposed multi-move update strategy. The corresponding results are reported in Figure 5.7.

Figure 5.7a shows that MDFIHA-ETGA achieves better overall performance than MDFIHA-ETGA-S, with a lower average gap and shorter average computation time. This observation highlights the effectiveness of the proposed leader-follower multi-move update strategy in enhancing solution quality while preserving computational efficiency. Figure 5.7b further presents the performance comparison across different instance sizes. While some variability can be observed, MDFIHA-ETGA demonstrates more stable and robust behavior and attains notably better average solution quality, especially for large-scale instances with more than 800 nodes, compared to MDFIHA-ETGA-S. These results suggest that the leader-follower multi-move strategy is particularly well suited for larger and more complex instances, as it enables more effective exploration of the search space



(a) Overall performance comparison. The x-axis reports the average computation time in seconds, and the y-axis shows the relative gap to the baseline.

(b) Performance comparison across instance sizes. The x-axis denotes instance size, and the y-axis shows the relative gap to the baseline.

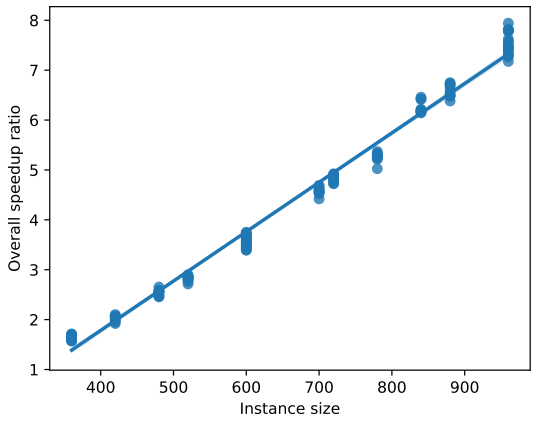
Figure 5.7 – Performance comparison between MDFIHA-ETGA and MDFIHA-ETGA-S, with VCGP [Vid+13] serving as the baseline.

and a greater ability to escape local optima than conventional single-move updates.

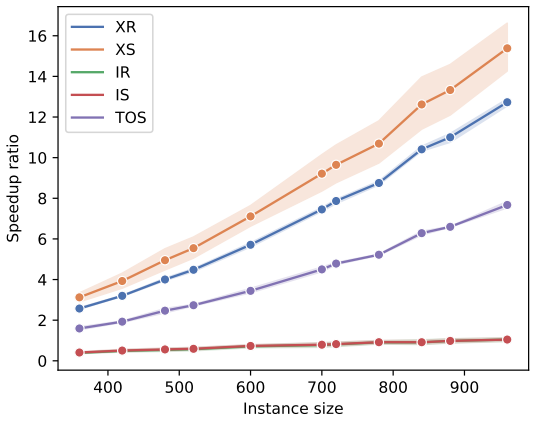
Next, to independently evaluate the computational efficiency in move evaluation of ETGA, we compare MDFIHA-ETGA-S with the CPU-based MDFIHA, both employing the same single best move strategy. Each algorithm is run five times per instance, with each run limited to 100 generations. The overall speedup ratio is defined as $\gamma_s = \frac{t_{\text{MDFIHA}}}{t_{\text{MDFIHA-ETGA-S}}}$, where t_{MDFIHA} and $t_{\text{MDFIHA-ETGA-S}}$ denote the average computation times for move evaluation in MDFIHA and MDFIHA-ETGA-S, respectively. Speedups for individual operators are computed similarly, based on their average computation times. The experimental results are shown in Figure 5.8.

Figure 5.8a shows a strong linear relationship (coefficient of determination $R^2 = 0.99$) between instance size and speedup, with slope ≈ 0.0099. This indicates excellent scalability of the ETGA, which means that larger instances benefit from greater acceleration. Specifically, the speedup grows from about 1.6 for 360-node instances to nearly 7.5 for 960-node instances, demonstrating significant improvements in computational efficiency, even under neighborhood reduction and pruning settings.

Figure 5.8b illustrates the speedup ratios of different local search operators across various instance sizes in the *V13* benchmark. Recall that the operators in ETGA are categorized as intra-operators (*Intra-Relocate* (IR), *Intra-Swap* (IS), and *2-opt* (TO)) and



(a) Overall speedup across different instance sizes, with a regression slope ≈ 0.0099 and $R^2 = 0.99$.



(b) Speedup ratios for operators across different instance size.

Figure 5.8 – ETGA speedup analysis on *V13* instances. The x-axis indicates the instance sizes, y-axis represents the speedup ratio.

inter-operators (*Inter-Relocate* (XR), *Inter-Swap* (XS), and *2-opt** (TOS)) depending on whether they act on one or two routes. It can be observed that inter-operators (XR, XS, TOS) achieve substantial speedups because the ETGA framework efficiently parallelizes their computationally intensive evaluations on the GPU. In contrast, intra-route operators (IR, IS) on a single route show more modest speedups. This is because these operators have lower time complexity and benefit less from GPU parallelism; the overhead of the GPU operations becomes more significant relative to the total computation.

5.5 Chapter conclusion

We propose a novel hybrid algorithm MDFIHA for solving a class of multi-depot vehicle routing problems. MDFIHA incorporates several key innovations: a multi-depot-supported feasible-and-infeasible search guided by a multi-penalty evaluation function and enhanced with two depot-related operators to effectively address the complex multi-depot constraints, and a learning-driven diversity-controlled route-exchange crossover to generate promising offspring solutions. To further improve computational efficiency on large-scale instances, we develop the enhanced MDFIHA-ETGA algorithm that integrates the edge-implemented tensor-based GPU acceleration to significantly accelerate neighborhood evaluations, and incorporates a leader-follower multi-move update strategy to fully exploit GPU parallelism and enhance solution quality.

Extensive experiments on four benchmark sets, covering three different MDVRP problems and totaling 125 instances, demonstrate the effectiveness of the proposed algorithm. MDFIHA achieves competitive or superior performance compared to state-of-the-art algorithms, establishing 45 record-breaking upper bounds. Additionally, the tensor-based GPU acceleration framework with multi-move update strategy in MDFIHA-ETGA significantly enhances the search process, achieving substantial speedups while further improving solution quality on large-scale instances.

The proposed algorithm has strong potential for extension to other vehicle routing problems. Future work will investigate these extensions and further leverage the tensor-based GPU acceleration framework and GPU parallelism to enhance scalability and computational efficiency.

CONCLUSION

Conclusion

This thesis has presented a comprehensive study on memetic algorithms and GPU acceleration for multi-attribute vehicle routing problems. We focused on several complex VRP variants, including the Vehicle Routing Problem with Simultaneous Pickup and Delivery and Time Windows (VRPSPDTW), a particularly challenging single-depot VRP variant featuring hierarchical objectives and multiple complex constraints. The VRPSPDTW can also be viewed as a generalization of the classic CVRP and VRPTW. Additionally, we studied a class of Multi-Depot VRPs (MDVRPs), including the MDVRP, MDVRPTW, and MDOVRP, which involve various complex constraints and whose multi-depot setting aligns with practical real-world applications. As discussed in Chapter 1, numerous methods have been proposed to solve these problems. However, significant challenges remain. To address these, we developed advanced memetic algorithms specifically tailored to the characteristics of these complex VRPs. To further tackle the challenge of computational efficiency on large-scale instances, we introduced tensor-based GPU acceleration (TGA) framework, along with two implementations, node-implemented TGA (NTGA) and edge-implemented TGA (ETGA), which enable highly efficient local search and neighborhood evaluation by exploiting fine-grained GPU parallelism. Extensive experimental results have demonstrated that the proposed algorithms achieve state-of-the-art solution quality, effectively balance exploration and exploitation, and significantly improve computational efficiency, confirming the practical value of our approaches for both single-depot and multi-depot MAVRPs with complex constraints. The detailed conclusions for each chapter are summarized below.

In Chapter 2, we presented a comprehensive review of GPU-based metaheuristics for routing problems published since 2010. This chapter first revisited the foundational concepts of metaheuristics and GPU parallel computing, and then examined the opportunities for GPU-based metaheuristics. It surveyed recent advances in GPU-accelerated methods, highlighting key strategies for exploiting parallelism as well as common GPU programming practices. Furthermore, it discussed the main challenges and promising research

directions in leveraging GPU parallel computing to enhance the efficiency and scalability of routing problem solution approaches. Overall, this review establishes a solid foundation for the future development of GPU-accelerated algorithms and provides valuable insights for researchers in this field.

In Chapter 3, we focused on the VRPSPDTW, a generalization of classic VRP variants such as the CVRP and VRPTW, and proposed an effective memetic algorithm to tackle this challenging problem. The algorithmic framework integrates several innovations to balance exploration and exploitation, including (1) a lightweight feasible-infeasible search with route descent, penalty management, and constant-time move evaluation, (2) a learning-based adaptive route-inheritance crossover to improve generalization and robustness, and (3) a fitness-distance population management scheme using max-min normalization to preserve diversity. Extensive experiments show that our algorithm consistently outperforms reference methods with high computational efficiency, attaining the best solutions for all 65 popular benchmark instances (57 best-known solutions and 8 new upper bounds) and setting new best upper bounds for all 20 large real-world instances. These results highlight both its effectiveness and practical applicability, providing a solid foundation for subsequent research.

In Chapter 4, we reformulated the constant-time move evaluation from Chapter 3 into a sequence-concatenation move evaluation using attribute representation, which also forms the foundation of the solution tensor representation suited for GPU parallelism. Building on this, we proposed the Tensor-based GPU Acceleration (TGA) framework to accelerate local search in vehicle routing. By exploiting fine-grained GPU parallelism, TGA greatly speeds up neighborhood evaluation in local search. Extensive experiments on the CVRP, VRPTW, and VRPSPDTW demonstrate its effectiveness, delivering substantial performance gains over traditional CPU-based implementations. Furthermore, detailed analyses of operator acceleration and key algorithmic steps provide valuable insights and highlight critical bottlenecks for future improvement.

In Chapter 5, we addressed Multi-Depot Vehicle Routing Problems (MDVRPs) and proposed a novel hybrid algorithm MDFIHA based on memetic algorithm framework. The algorithm integrates several key components: (1) a multi-depot-supported feasible-infeasible search guided by a multi-penalty evaluation function and enhanced by two depot-related operators tailored to complex multi-depot constraints, (2) a diversity-controlled route-exchange crossover for generating promising offspring, and (3) a fitness-distance-based population management strategy. To enhance computational efficiency on large

instances, we further developed an enhanced MDFIHA-ETGA with edge-implemented TGA and leader-follower multi-move update strategy, fully exploiting GPU parallelism. Extensive experiments on 125 instances from four benchmark sets, covering three MD-VRP variants, confirm the effectiveness of MDFIHA. The algorithm delivers competitive or superior performance to state-of-the-art methods, establishing 45 new best upper bounds. Moreover, ETGA significantly accelerates the search process, yielding substantial speedups and further improving solution quality on large-scale instances.

Perspectives

In this thesis, we reviewed state-of-the-art methodologies for several complex multi-attribute vehicle routing problems, as well as GPU-based metaheuristics for routing problems. Building upon this foundation, we developed advanced memetic algorithms and tensor-based GPU acceleration framework to effectively address complex multi-attribute VRPs. The proposed methods demonstrated strong solution quality and computational efficiency across a broad range of benchmark instances. These results open several promising directions for future research.

The proposed memetic algorithms have demonstrated considerable flexibility and robustness on problems such as the Vehicle Routing Problem with Simultaneous Pickup and Delivery and Time Windows, the Multi-Depot Vehicle Routing Problem, and related variants. Their core components, including the feasible and infeasible search, the multi-penalty evaluation function, route-based crossover operators, and learning-enhanced strategies, form a modular and extensible framework. Future work may extend these strategies to richer VRP variants by incorporating additional operational constraints and more complex multi-attribute characteristics. Further improvements could also be achieved by integrating more advanced learning-based strategies to guide the search process within complex feasible and infeasible regions. In particular, adaptive operator selection mechanisms and more intelligent search control strategies could enhance the balance between diversification and intensification, leading to improved robustness and solution quality across diverse instance types.

The proposed Tensor-based GPU Acceleration framework, together with its two implementations, node-implemented TGA and edge-implemented TGA, also offers substantial opportunities for further development. TGA can be integrated into a wide range of local search-based routing algorithms, providing an efficient and extensible tool for solving

diverse VRP variants. Its tensorized design makes it adaptable to other combinatorial optimization problems and large-scale real-world applications. However, several challenges remain, including computational bottlenecks, storage of invalid or redundant information, limited pruning strategies, CPU-GPU communication overhead, and performance gaps relative to highly optimized CPU-based methods. Addressing these issues could lead to a more mature and unified GPU acceleration framework capable of efficiently solving large-scale VRPs and related combinatorial optimization problems.

Although TGA was primarily designed to accelerate local search, its underlying philosophy of structured tensorization to exploit GPU parallelism can be extended beyond this context. Population-based metaheuristics, such as genetic algorithms, can greatly benefit from the efficient parallel evaluation of large populations and computationally intensive operators. Therefore, a fully GPU-based memetic algorithm could be developed by integrating TGA into a tensorized population-based metaheuristic framework, enabling parallel execution of both local search procedures and population-level operators. Exploring these extensions could substantially broaden the applicability and impact of GPU-accelerated optimization methods.

Finally, due to its tensor-based representation, TGA can be naturally combined with deep learning, reinforcement learning, and large language model-based techniques to further enhance search intelligence. For example, learning models could be trained to predict promising regions of the search space, estimate move quality, or guide adaptive operator selection using the structured information encoded in solution tensors. Such integration could lead to more intelligent, adaptive, and data-driven optimization frameworks.

Overall, the methodologies developed in this thesis provide both a practical toolkit for solving complex vehicle routing problems and a conceptual foundation for future innovations at the intersection of metaheuristics, artificial intelligence, and high-performance parallel computing. Continued research along these directions may further advance the state-of-the-art in scalable combinatorial optimization.

Reference

[Abb+20] Mahdi Abbasi et al., « An efficient parallel genetic algorithm solution for vehicle routing problem in cloud implementation of the intelligent transportation systems », *in*: *Journal of Cloud Computing* 9 (2020), pp. 1–14.

[AM02] Enrico Angelelli and Renata Mansini, « The vehicle routing problem with time windows and simultaneous pick-up and delivery », *in*: *Quantitative Approaches to Distribution Logistics and Supply Chain Management*, Springer, 2002, pp. 249–267.

[ARR07] Renata M Aiex, Mauricio GC Resende, and Celso C Ribeiro, « TTT plots: a perl program to create time-to-target plots », *in*: *Optimization Letters* 1 (2007), pp. 355–366.

[AS20] Marwan F Abdelatti and Manbir S Sodhi, « An improved gpu-accelerated heuristic technique applied to the capacitated vehicle routing problem », *in*: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 2020, pp. 663–671.

[ASS22] Marwan Abdelatti, Manbir Sodhi, and Resit Sendag, « A multi-gpu parallel genetic algorithm for large-scale vehicle routing problems », *in*: *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, IEEE, 2022, pp. 1–8.

[Bal+15] Olfa Bali et al., « GPU particle swarm optimization applied to travelling salesman problem », *in*: *2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip*, IEEE, 2015, pp. 112–119.

[Bal+17] Olfa Bali et al., « ACO-PSO optimization for solving TSP problem with GPU acceleration », *in*: *Intelligent Systems Design and Applications: 16th International Conference on Intelligent Systems Design and Applications (ISDA 2016) held in Porto, Portugal, December 16-18, 2016*, Springer, 2017, pp. 559–569.

[BB18] Abdelhamid Benaini and Achraf Berrajaa, « Genetic algorithm for large dynamic vehicle routing problem on GPU », *in*: *2018 4th International Conference on Logistics Operations Management (GOL)*, IEEE, 2018, pp. 1–9.

[BHS19] Lukas Bach, Geir Hasle, and Christian Schulz, « Adaptive large neighborhood search on the graphics processing unit », *in*: *European Journal of Operational Research* 275.*1* (2019), pp. 53–66.

[BM09] Roberto Baldacci and Aristide Mingozzi, « A unified exact method for solving different classes of vehicle routing problems », *in*: *Mathematical Programming* 120 (2009), pp. 347–380.

[BMS17] Marco Baioletti, Alfredo Milani, and Valentino Santucci, « Algebraic particle swarm optimization for the permutations search space », *in*: *2017 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2017, pp. 1587–1594.

[Bra20] José Brandão, « A memory-based iterated local search algorithm for the multi-depot open vehicle routing problem », *in*: *European Journal of Operational Research* 284.*2* (2020), pp. 559–571.

[BRV16] Kris Braekers, Katrien Ramaekers, and Inneke Van Nieuwenhuyse, « The vehicle routing problem: State of the art classification and review », *in*: *Computers & Industrial Engineering* 99 (2016), pp. 300–313.

[CAO20] Seval Capraz, Halil Azyikmis, and Adnan Ozsoy, « An optimized GPU-accelerated route planning of multi-UAV systems using simulated annealing », *in*: *International Journal of Machine Learning and Computing* 10.*3* (2020).

[Cat+14] Diego Cattaruzza et al., « A memetic algorithm for the multi trip vehicle routing problem », *in*: *European Journal of Operational Research* 236.*3* (2014), pp. 833–848.

[Cec+13] José M Cecilia et al., « Enhancing data parallelism for ant colony optimization on GPUs », *in*: *Journal of Parallel and Distributed Computing* 73.*1* (2013), pp. 42–51.

[Cec+18] José M Cecilia et al., « High-throughput ant colony optimization on graphics processing units », *in*: *Journal of Parallel and Distributed Computing* 113 (2018), pp. 261–274.

[CG19] John Runwei Cheng and Mitsuo Gen, « Accelerating genetic algorithms with GPU computing: A selective overview », *in*: *Computers & Industrial Engineering* 128 (2019), pp. 514–525.

[CGL97] Jean-François Cordeau, Michel Gendreau, and Gilbert Laporte, « A tabu search heuristic for periodic and multi-depot vehicle routing problems », *in*: *Networks: An International Journal* 30.*2* (1997), pp. 105–119.

[CL03] Jean-François Cordeau and Gilbert Laporte, « A tabu search heuristic for the static multi-vehicle dial-a-ride problem », *in*: *Transportation Research Part B: Methodological* 37.*6* (2003), pp. 579–594.

[CLM01] Jean-François Cordeau, Gilbert Laporte, and Anne Mercier, « A unified tabu search heuristic for vehicle routing problems with time windows », *in*: *Journal of the Operational Research Society* 52.*8* (2001), pp. 928–936.

[CLM04] Jean-François Cordeau, Gilbert Laporte, and Anne Mercier, « Improved tabu search algorithm for the handling of route duration constraints in vehicle routing problems with time windows », *in*: *Journal of the Operational Research Society* 55.*5* (2004), pp. 542–546.

[CM12] Jean-François Cordeau and Mirko Maischberger, « A parallel iterated tabu search heuristic for vehicle routing problems », *in*: *Computers & Operations Research* 39.*9* (2012), pp. 2033–2050.

[CM14] Claudio Contardo and Rafael Martinelli, « A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints », *in*: *Discrete Optimization* 12 (2014), pp. 129–146.

[Coe+16] IM Coelho et al., « An integrated CPU–GPU heuristic inspired on variable neighbourhood search for the single vehicle routing problem with deliveries and selective pickups », *in*: *International Journal of Production Research* 54.*4* (2016), pp. 945–962.

[CPV02] Richard K Congram, Chris N Potts, and Steef L van de Velde, « An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem », *in*: *INFORMS Journal on Computing* 14.*1* (2002), pp. 52–67.

[CW64] Geoff Clarke and John W Wright, « Scheduling of vehicles from a central depot to a number of delivery points », *in*: *Operations Research* 12.*4* (1964), pp. 568–581.

[DDK12] Audrey Delévacq, Pierre Delisle, and Michaël Krajecki, « Parallel GPU implementation of iterated local search for the travelling salesman problem », *in*: *International Conference on Learning and Intelligent Optimization*, Springer, 2012, pp. 372–377.

[Del+13] Audrey Delévacq et al., « Parallel ant colony optimization on graphics processing units », *in*: *Journal of Parallel and Distributed Computing* 73.*1* (2013), pp. 52–61.

[Det01] Jan Dethloff, « Vehicle routing and reverse logistics: The vehicle routing problem with simultaneous delivery and pick-up: Fahrzeugeinsatzplanung und Redistribution: Tourenplanung mit simultaner Auslieferung und Rückholung », *in*: *OR-Spektrum* 23 (2001), pp. 79–96.

[DFJ54] George Dantzig, Ray Fulkerson, and Selmer Johnson, « Solution of a large-scale traveling-salesman problem », *in*: *Journal of the Operations Research Society of America* 2.*4* (1954), pp. 393–410.

[DR59] George B Dantzig and John H Ramser, « The truck dispatching problem », *in*: *Management Science* 6.*1* (1959), pp. 80–91.

[DS13] Laurence Dawson and Iain Stewart, « Improving ant colony optimization performance on the GPU using CUDA », *in*: *2013 IEEE Congress on Evolutionary Computation*, IEEE, 2013, pp. 1901–1908.

[Esc+14] John Willmer Escobar et al., « A hybrid granular tabu search algorithm for the multi-depot vehicle routing problem », *in*: *Journal of Heuristics* 20 (2014), pp. 483–509.

[Ess+19] Mokhtar Essaid et al., « GPU parallelization strategies for metaheuristics: a survey », *in*: *International Journal of Parallel, Emergent and Distributed Systems* 34.*5* (2019), pp. 497–522.

[Fer+13] Ana M Ferreiro et al., « An efficient implementation of parallel simulated annealing algorithm in GPUs », *in*: *Journal of Global Optimization* 57 (2013), pp. 863–890.

[FGI05] Birger Funke, Tore Grünert, and Stefan Irnich, « Local search for vehicle routing and scheduling problems: Review and conceptual integration », *in*: *Journal of Heuristics* 11 (2005), pp. 267–306.

[FLZ10] Jie Fu, Lin Lei, and Guohua Zhou, « A parallel ant colony optimization algorithm with GPU-acceleration based on all-in-roulette selection », *in*: *Third International Workshop on Advanced Computational Intelligence*, IEEE, 2010, pp. 260–264.

[FT11] Noriyuki Fujimoto and Shigeyoshi Tsutsui, « A highly-parallel TSP solver for a GPU computing platform », *in*: *Numerical Methods and Applications: 7th International Conference, NMA 2010, Borovets, Bulgaria, August 20-24, 2010. Revised Papers 7*, Springer, 2011, pp. 264–271.

[GH11] Fred Glover and Jin-Kao Hao, « The case for strategic oscillation », *in*: *Annals of Operations Research* 183(1) (2011), pp. 163–173.

[GH99] Hermann Gehring and Jörg Homberger, « A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows », *in*: *Proceedings of EUROGEN99*, vol. 2, Citeseer, 1999, pp. 57–64.

[GTV02] ID Giosa, IL Tansini, and IO Viera, « New assignment algorithms for the multi-depot vehicle routing problem », *in*: *Journal of the Operational Research Society* 53.*9* (2002), pp. 977–984.

[GW93] V Scott Gordon and Darrell Whitley, « Serial and parallel genetic algorithms as function optimizers », *in*: *ICGA*, 1993, pp. 177–183.

[Har+07] Mark Harris et al., « Optimizing parallel reduction in CUDA », *in*: *Nvidia Developer Technology* 2.*4* (2007), p. 70.

[Hel00] Keld Helsgaun, « An effective implementation of the Lin–Kernighan traveling salesman heuristic », *in*: *European Journal of Operational Research* 126.*1* (2000), pp. 106–130.

[HH23a] Pengfei He and Jin-Kao Hao, « General edge assembly crossover-driven memetic search for split delivery vehicle routing », *in*: *Transportation Science* 57.*2* (2023), pp. 482–511.

[HH23b] Pengfei He and Jin-Kao Hao, « Memetic search for the minmax multiple traveling salesman problem with single and multiple depots », *in*: *European Journal of Operational Research* 307.*3* (2023), pp. 1055–1070.

[HS19] Julian Hof and Michael Schneider, « An adaptive large neighborhood search with path relinking for a class of vehicle-routing problems with simultaneous pickup and delivery », *in*: *Networks* 74.*3* (2019), pp. 207–250.

[JJL08] Adam Janiak, Wladyslaw A Janiak, and Maciej Lichtenstein, « Tabu Search on GPU. », *in*: *J. Univers. Comput. Sci.* 14.*14* (2008), pp. 2416–2426.

[Kan+16] Semin Kang et al., « GPU-based parallel genetic approach to large-scale travelling salesman problem », *in*: *The Journal of Supercomputing* 72 (2016), pp. 4399–4414.

[KC13] Sally Kassem and Mingyuan Chen, « Solving reverse logistics vehicle routing problems with time windows », *in*: *The International Journal of Advanced Manufacturing Technology* 68 (2013), pp. 57–68.

[KE95] James Kennedy and Russell Eberhart, « Particle swarm optimization », *in*: *Proceedings of ICNN'95-International Conference on Neural Networks*, vol. 4, IEEE, 1995, pp. 1942–1948.

[KLT20] Çağrı Koç, Gilbert Laporte, and İlknur Tükenmez, « A review of vehicle routing with simultaneous pickup and delivery », *in*: *Computers & Operations Research* 122 (2020), p. 104987.

[Koo+22] Wouter Kool et al., « Hybrid genetic search for the vehicle routing problem with time windows: A high-performance implementation », *in*: *12th DIMACS Implementation Challenge Workshop*, DIMACS, Rutgers University Piscataway, NJ, 2022.

[KP14] Volodymyr Kuleshov and Doina Precup, « Algorithms for multi-armed bandit problems », *in*: *arXiv preprint arXiv:1402.6028* (2014).

[KP15] Sašo Karakatič and Vili Podgorelec, « A survey of genetic algorithms for solving multi depot vehicle routing problem », *in*: *Applied Soft Computing* 27 (2015), pp. 519–532.

[Lab+16] Nacima Labadie et al., *Metaheuristics for vehicle routing problems*, Wiley Online Library, 2016.

[Lap84] Gilbert Laporte, « Optimal solutions to capacitated multidepot vehicle routing problems », *in*: *Congressus Nemerantium* 4 (1984), pp. 283–292.

[Lar+99] Pedro Larranaga et al., « Genetic algorithms for the travelling salesman problem: A review of representations and operators », *in*: *Artificial Intelligence Review* 13 (1999), pp. 129–170.

[LC10] Mingyong Lai and Erbao Cao, « An improved differential evolution algorithm for vehicle routing problem with simultaneous pickups and deliveries and time windows », *in*: *Engineering Applications of Artificial Intelligence* 23.*2* (2010), pp. 188–195.

[LH24] Zhenyu Lei and Jin-Kao Hao, « An effective memetic algorithm for the close-enough traveling salesman problem », *in*: *Applied Soft Computing* 153 (2024), p. 111266.

[LH25] Zhenyu Lei and Jin-Kao Hao, « A Memetic Algorithm for Vehicle Routing with Simultaneous Pickup and Delivery and Time Windows », *in*: *IEEE Transactions on Evolutionary Computation* 29.*5* (2025), pp. 1924–1936.

[LHW22] Mingjie Li, Jin-Kao Hao, and Qinghua Wu, « Learning-driven feasible and infeasible tabu search for airport gate assignment », *in*: *European Journal of Operational Research* 302.*1* (2022), pp. 172–186.

[LHW26] Zhenyu Lei, Jin-Kao Hao, and Qinghua Wu, *Speeding up Local Optimization in Vehicle Routing with Tensor-based GPU Acceleration*, 2026, arXiv: `2506.17357 [cs.DC]`, URL: `https://arxiv.org/abs/2506.17357v2`.

[LJG14] Ran Liu, Zhibin Jiang, and Na Geng, « A hybrid genetic algorithm for the multi-depot open vehicle routing problem », *in*: *OR Spectrum* 36 (2014), pp. 401–421.

[LK73] Shen Lin and Brian W Kernighan, « An effective heuristic algorithm for the traveling-salesman problem », *in*: *Operations Research* 21.*2* (1973), pp. 498–516.

[LL96] Soo-Young Lee and Kyung Geun Lee, « Synchronous and asynchronous parallel simulated annealing with multiple Markov chains », *in*: *IEEE Transactions on Parallel and Distributed Systems* 7.*10* (1996), pp. 993–1008.

[LM21] Eduardo Lalla-Ruiz and Martijn Mes, « Mathematical formulations and improvements for the multi-depot open vehicle routing problem », *in*: *Optimization Letters* 15.*1* (2021), pp. 271–286.

[LNT88] Gilbert Laporte, Yves Nobert, and Serge Taillefer, « Solving a family of multi-depot vehicle routing and location-routing problems », *in*: *Transportation Science* 22.*3* (1988), pp. 161–172.

[Lóp+16] Manuel López-Ibáñez et al., « The irace package: Iterated racing for automatic algorithm configuration », *in*: *Operations Research Perspectives* 3 (2016), pp. 43–58.

[LR78] Jan Karel Lenstra and AHG Rinnooy Kan, « Complexity of scheduling under precedence constraints », *in*: *Operations Research* 26.*1* (1978), pp. 22–35.

[LTY21] Shengcai Liu, Ke Tang, and Xin Yao, « Memetic search for vehicle routing with simultaneous pickup-delivery and time windows », *in*: *Swarm and Evolutionary Computation* 66 (2021), p. 100927.

[LW05] Andrew Lim and Fan Wang, « Multi-depot vehicle routing problem: A one-stage approach », *in*: *IEEE Transactions on Automation Science and Engineering* 2.*4* (2005), pp. 397–402.

[LZZ09] Chun-Hua Liang, Hong Zhou, and Jian Zhao, « Vehicle routing problem with time windows and simultaneous pickups and deliveries », *in*: *2009 16th International Conference on Industrial Engineering and Engineering Management*, IEEE, 2009, pp. 685–689.

[Mal18] Vinaya Malleypally, « Parallelizing Tabu Search Based Optimization Algorithm on GPUs », MA thesis, University of South Florida, 2018.

[MC10] Pablo Moscato and Carlos Cotta, « A modern introduction to memetic algorithms », *in*: *Handbook of Metaheuristics* (2010), pp. 141–183.

[Men+19] Breno AM Menezes et al., « Parallelization strategies for GPU-based ant colony optimization solving the traveling salesman problem », *in*: *2019 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2019, pp. 3094–3101.

[Min89] Hokey Min, « The multiple vehicle routing problem with simultaneous delivery and pick-up points », *in*: *Transportation Research Part A: General* 23.*5* (1989), pp. 377–386.

[MLG22] Sandeep U Mane, Pooja S Lokare, and Harsha R Gaikwad, « Overview and applications of GPGPU based parallel ant colony optimization », *in*: *arXiv preprint arXiv:2203.11487* (2022).

[Mon+15] Jairo R Montoya-Torres et al., « A literature review on the vehicle routing problem with multiple depots », *in*: *Computers & Industrial Engineering* 79 (2015), pp. 115–129.

[Nag97] Yuichi Nagata, « Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem », *in*: *Proceedings of the 7th Internatinal Conferencenon Genetic Algorithms* (1997), pp. 450–457.

[NBD10] Yuichi Nagata, Olli Bräysy, and Wout Dullaert, « A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows », *in*: *Computers & Operations Research* 37.*4* (2010), pp. 724–737.

[NC12] Ferrante Neri and Carlos Cotta, « Memetic algorithms and memetic computing optimization: A literature review », *in*: *Swarm and Evolutionary Computation* 2 (2012), pp. 1–14.

[NK10] Yuichi Nagata and Shigenobu Kobayashi, « A memetic algorithm for the pickup and delivery problem with time windows using selective route exchange crossover », *in*: *International Conference on Parallel Problem Solving from Nature*, Springer, 2010, pp. 536–545.

[OB15] Molly A O'Neil and Martin Burtscher, « Rethinking the parallelization of random-restart hill climbing: a case study in optimizing a 2-opt TSP solver for GPU execution », *in*: *Proceedings of the 8th Workshop on General Purpose Processing using GPUs*, 2015, pp. 99–108.

[OÖ01] Esin Onbaşoğlu and Linet Özdamar, « Parallel simulated annealing algorithms in global optimization », *in*: *Journal of Gglobal Optimization* 19.*1* (2001), pp. 27–50.

[OTB11] Molly A O'Neil, Dan Tamir, and Martin Burtscher, « A parallel gpu version of the traveling salesman problem », *in*: *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2011, p. 1.

[Pan+18] Ramesh Ramasamy Pandi et al., « GPU-accelerated tabu search algorithm for dial-a-ride problem », *in*: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2018, pp. 2519–2524.

[Pan+20] Ramesh Ramasamy Pandi et al., « A generic GPU-accelerated framework for the dial-a-ride problem », *in*: *IEEE Transactions on Intelligent Transportation Systems* 22.*10* (2020), pp. 6473–6488.

[PHK10] Daniel Cosmin Porumbel, Jin-Kao Hao, and Pascale Kuntz, « An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring », *in*: *Computers & Operations Research* 37.*10* (2010), pp. 1822–1832.

[PNC11] Martín Pedemonte, Sergio Nesmachnow, and Héctor Cancela, « A survey on parallel ant colony optimization », *in*: *Applied Soft Computing* 11.*8* (2011), pp. 5181–5197.

[Pol+04] Michael Polacek et al., « A variable neighborhood search for the multi depot vehicle routing problem with time windows », *in*: *Journal of Heuristics* 10 (2004), pp. 613–627.

[Pol+08] Michael Polacek et al., « A cooperative and adaptive variable neighborhood search for the multi depot vehicle routing problem with time windows », *in*: *Business Research* 1 (2008), pp. 207–218.

[PR07] David Pisinger and Stefan Ropke, « A general heuristic for vehicle routing problems », *in*: *Computers & Operations Research* 34.*8* (2007), pp. 2403–2435.

[PR18] David Pisinger and Stefan Ropke, « Large neighborhood search », *in*: *Handbook of Metaheuristics*, Springer, 2018, pp. 99–127.

[Pra+24] Rafael Praxedes et al., « A unified exact approach for a broad class of vehicle routing problems with simultaneous pickup and delivery », *in*: *Computers & Operations Research* 162 (2024), p. 106467.

[Pri04] Christian Prins, « A simple and effective evolutionary algorithm for the vehicle routing problem », *in*: *Computers & Operations Research* 31.*12* (2004), pp. 1985–2002.

[QC17] Wen-Bao Qiao and Jean-Charles Créput, « Parallel 2-opt local search on GPU », *in*: *Int J Electric Comput Energetic Electron Commun Eng* 11.*3* (2017), pp. 291–295.

[QC18] Wen-Bao Qiao and Jean-Charles Créput, « Massive 2-opt and 3-opt moves with high performance gpu local search to large-scale traveling salesman problem », *in*: *International Conference on Learning and Intelligent Optimization*, Springer, 2018, pp. 82–97.

[QC20] Wen-Bao Qiao and Jean-Charles Créput, « Multiple k- opt evaluation multiple k- opt moves with GPU high performance local search to large-scale traveling salesman problems », *in*: *Annals of Mathematics and Artificial Intelligence* 88.*4* (2020), pp. 347–365.

[RBL96] Jacques Renaud, Fayez F Boctor, and Gilbert Laporte, « An improved petal heuristic for the vehicle routeing problem », *in*: *Journal of the Operational Research Society* 47.*2* (1996), pp. 329–336.

[Rei91] Gerhard Reinelt, « TSPLIB—A traveling salesman problem library », *in*: *ORSA Journal on Computing* 3.*4* (1991), pp. 376–384.

[Rey+18] Antón Rey et al., « A cpu-gpu parallel ant colony optimization solver for the vehicle routing problem », *in*: *Applications of Evolutionary Computation: 21st International Conference, EvoApplications 2018, Parma, Italy, April 4-6, 2018, Proceedings 21*, Springer, 2018, pp. 653–667.

[RLB96] Jacques Renaud, Gilbert Laporte, and Fayez F Boctor, « A tabu search heuristic for the multi-depot vehicle routing problem », *in*: *Computers & Operations Research* 23.*3* (1996), pp. 229–235.

[Rob+18] Jeffrey A Robinson et al., « Analysis of a high-performance TSP solver on the GPU », *in*: *Journal of Experimental Algorithmics (JEA)* 23 (2018), pp. 1–22.

[RRV12] Celso C Ribeiro, Isabel Rosseti, and Reinaldo Vallejos, « Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms », *in*: *Journal of Global Optimization* 54 (2012), pp. 405–429.

[RS12a] Kamil Rocki and Reiji Suda, « Accelerating 2-opt and 3-opt local search using GPU in the travelling salesman problem », *in*: *2012 International Conference on High Performance Computing & Simulation (HPCS)*, IEEE, 2012, pp. 489–495.

[RS12b] Kamil Rocki and Reiji Suda, « An efficient GPU implementation of a multi-start TSP solver for large problem instances », *in*: *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, 2012, pp. 1441–1442.

[RS13] Kamil Rocki and Reiji Suda, « High performance GPU accelerated local optimization in TSP », *in*: *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, IEEE, 2013, pp. 1788–1796.

[SÇA21] Mir Ehsan Hesam Sadati, Bülent Çatay, and Deniz Aksen, « An efficient variable neighborhood search with tabu shaking for a class of multi-depot vehicle routing problems », *in*: *Computers & Operations Research* 133 (2021), p. 105269.

[Sch13] Christian Schulz, « Efficient local search on the GPU—investigations on the vehicle routing problem », *in*: *Journal of Parallel and Distributed Computing* 73.*1* (2013), pp. 14–31.

[Sch20] Guido Schryen, « Parallel computational optimization in operations research: A new integrative framework, literature review and research directions », *in*: *European Journal of Operational Research* 287.*1* (2020), pp. 1–18.

[SD13] Jagiełło Szymon and Żelazny Dominik, « Solving multi-criteria vehicle routing problem by parallel tabu search on GPU », *in*: *Procedia Computer Science* 18 (2013), pp. 2529–2532.

[SD99] Thomas Stützle and Marco Dorigo, « ACO algorithms for the traveling salesman problem », *in*: *Evolutionary Algorithms in Engineering and Computer Science* 4 (1999), pp. 163–183.

[Shi+20] Yong Shi et al., « A lexicographic-based two-stage algorithm for vehicle routing problem with simultaneous pickup–delivery and time window », *in*: *Engineering Applications of Artificial Intelligence* 95 (2020), p. 103901.

[Ski16] Rafał Skinderowicz, « The GPU-based parallel ant colony system », *in*: *Journal of Parallel and Distributed Computing* 98 (2016), pp. 48–60.

[Ski20] Rafał Skinderowicz, « Implementing a GPU-based parallel MAX–MIN Ant System », *in*: *Future Generation Computer Systems* 106 (2020), pp. 277–295.

[Sol87] Marius M Solomon, « Algorithms for the vehicle routing and scheduling problems with time window constraints », *in*: *Operations Research* 35.*2* (1987), pp. 254–265.

[SS06] Kenneth Sörensen and Marc Sevaux, « MA|PM: memetic algorithms with population management », *in*: *Computers & Operations Research* 33.*5* (2006), pp. 1214–1225.

[SS20] Rohit Sharma and Sanjay Saini, « Heuristics and meta-heuristics based multiple depot vehicle routing problem: A review », *in*: *2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)*, IEEE, 2020, pp. 683–689.

[Sun+18] Wen Sun et al., « Adaptive feasible and infeasible tabu search for weighted vertex coloring », *in*: *Information Sciences* 466 (2018), pp. 203–219.

[SUP21] Ruslan Sadykov, Eduardo Uchoa, and Artur Pessoa, « A bucket graph–based labeling algorithm with application to vehicle routing », *in*: *Transportation Science* 55.*1* (2021), pp. 4–28.

[Tal09] El-Ghazali Talbi, *Metaheuristics: from design to implementation*, John Wiley & Sons, 2009.

[Tan+21] Ke Tang et al., « Few-shots parallel algorithm portfolio construction via co-evolution », *in*: *IEEE Transactions on Evolutionary Computation* 25.*3* (2021), pp. 595–607.

[TD15] Ying Tan and Ke Ding, « A survey on GPU-based implementation of swarm intelligence algorithms », *in*: *IEEE Transactions on Cybernetics* 46.*9* (2015), pp. 2028–2041.

[Til69] Frank A Tillman, « The multiple terminal delivery problem with probabilistic demands », *in*: *Transportation Science* 3.*3* (1969), pp. 192–204.

[Tor+22] Isis Torres-Pérez et al., « New heuristics for assigning in the multi-depot vehicle routing problem », *in*: *IFAC-PapersOnLine* 55.*10* (2022), pp. 2228–2233.

[Tu+14] Wei Tu et al., « A bi-level Voronoi diagram-based metaheuristic for a large-scale multi-depot vehicle routing problem », *in*: *Transportation Research Part E: Logistics and Transportation Review* 61 (2014), pp. 84–97.

[TV06] Libertad Tansini and Omar Viera, « New measures of proximity for the assignment algorithms in the MDVRPTW », *in*: *Journal of the Operational Research Society* 57.*3* (2006), pp. 241–249.

[TYS16] Tolgahan Turker, Guray Yilmaz, and Ozgur Koray Sahingoz, « GPU-accelerated flight route planning for multi-UAV systems using simulated annealing », *in*: *International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, Springer, 2016, pp. 279–288.

[Uch+17] Eduardo Uchoa et al., « New benchmark instances for the capacitated vehicle routing problem », *in*: *European Journal of Operational Research* 257.*3* (2017), pp. 845–858.

[UIN14] Akihiro Uchida, Yasuaki Ito, and Koji Nakano, « Accelerating ant colony optimisation for the travelling salesman problem on the GPU », *in*: *International Journal of Parallel, Emergent and Distributed Systems* 29.*4* (2014), pp. 401–420.

[Vid+12] Thibaut Vidal et al., « A hybrid genetic algorithm for multidepot and periodic vehicle routing problems », *in*: *Operations Research* 60.*3* (2012), pp. 611–624.

[Vid+13] Thibaut Vidal et al., « A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows », *in*: *Computers & Operations Research* 40.*1* (2013), pp. 475–489.

[Vid+14] Thibaut Vidal et al., « A unified solution framework for multi-attribute vehicle routing problems », *in*: *European Journal of Operational Research* 234.*3* (2014), pp. 658–673.

[Vid22] Thibaut Vidal, « Hybrid genetic search for the CVRP: Open-source implementation and SWAP* Neighborhood », *in*: *Computers & Operations Research* 140 (2022), p. 105643.

[VM05] Joannes Vermorel and Mehryar Mohri, « Multi-armed bandit algorithms and empirical evaluation », *in*: *European Conference on Machine Learning*, Springer, 2005, pp. 437–448.

[Wan+13] Chao Wang et al., « Simulated annealing for a vehicle routing problem with simultaneous pickup-delivery and time windows », *in*: *Advances in Production Management Systems. Sustainable Production and Service Supply Chains: IFIP WG 5.7 International Conference, APMS 2013, State College, PA, USA, September 9-12, 2013, Proceedings, Part II*, Springer, 2013, pp. 170–177.

[Wan+15] Chao Wang et al., « A parallel simulated annealing method for the vehicle routing problem with simultaneous pickup–delivery and time windows », *in*: *Computers & Industrial Engineering* 83 (2015), pp. 111–122.

[Wan+20] Boqun Wang et al., « Multipopulation genetic algorithm based on GPU for solving TSP problem », *in*: *Mathematical Problems in Engineering* 2020.*1* (2020), p. 1398595.

[WC12] Hsiao-Fan Wang and Ying-Yen Chen, « A genetic algorithm for the simultaneous delivery and pickup problems with time window », *in*: *Computers & Industrial Engineering* 62.*1* (2012), pp. 84–95.

[WG23] Hongguang Wu and Yuelin Gao, « An ant colony optimization based on local search for the vehicle routing problem with simultaneous pickup–delivery and time window », *in*: *Applied Soft Computing* 139 (2023), p. 110203.

[Wil92] Frank Wilcoxon, « Individual comparisons by ranking methods », *in*: *Breakthroughs in statistics: Methodology and distribution*, Springer, 1992, pp. 196–202.

[WWY15] Kai-Cheng Wei, Chao-Chin Wu, and Hui-Liang Yu, « Mapping the simulated annealing algorithm onto CUDA GPUs », *in*: *2015 10th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, IEEE, 2015, pp. 358–365.

[YJC24] Luming Yang, Tao Jiang, and Ran Cheng, « Tensorized ant colony optimization for GPU acceleration », *in*: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2024, pp. 755–758.

[YKT19] Pramod Yelmewad, Aniket Kumar, and Basavaraj Talawar, « MMAS on GPU for large TSP instances », *in*: *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, IEEE, 2019, pp. 1–6.

[YT18] Pramod Yelmewad and Basavaraj Talawar, « Near optimal solution for traveling salesman problem using GPU », *in*: *2018 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, IEEE, 2018, pp. 1–6.

[YT19] Pramod Yelmewad and Basavaraj Talawar, « Parallel iterative hill climbing algorithm to solve TSP on GPU », *in*: *Concurrency and Computation: Practice and Experience* 31.7 (2019), e4974.

[YT20] Pramod Yelmewad and Basavaraj Talawar, « Gpu-based parallel heuristics for capacited vehicle routing problem », *in*: *2020 IEEE International Conference on Electronics, Computing and Communication Technologies*, IEEE, 2020, pp. 1–6.

[YT21] Pramod Yelmewad and Basavaraj Talawar, « Parallel version of local search heuristic algorithm to solve capacitated vehicle routing problem », *in*: *Cluster Computing* 24 (2021), pp. 3671–3692.

[YUY24] Salih Yalcin, Hamdi Burak Usul, and Gulay Yalcin, « CompreCity: accelerating the travelling salesman problem on GPU with data compression », *in*: *Integration* (2024), p. 102333.

[Zha+19] Zizhen Zhang et al., « GMMA: GPU-based multiobjective memetic algorithms for vehicle routing problem with route balancing », *in*: *Applied Intelligence* 49 (2019), pp. 63–78.

[ZHQ16] Yi Zhou, Fazhi He, and Yimin Qiu, « Optimization of parallel iterated local search algorithms on graphics processing unit », *in*: *The Journal of Supercomputing* 72.6 (2016), pp. 2394–2416.

[ZHQ17] Yi Zhou, Fazhi He, and Yimin Qiu, « Dynamic strategy based parallel ant colony optimization on GPUs for TSPs », *in*: *Science China Information Sciences* 60 (2017), pp. 1–3.

[ZHW24] Yuji Zou, Jin-Kao Hao, and Qinghua Wu, « A two-individual evolutionary algorithm for cumulative capacitated vehicle routing with single and multiple depots », *in*: *IEEE Transactions on Evolutionary Computation* (2024).

[ZWJ15] Yudong Zhang, Shuihua Wang, and Genlin Ji, « A comprehensive survey on particle swarm optimization algorithm and its applications », *in*: *Mathematical Problems in Engineering* 2015.1 (2015), p. 931256.

APPENDIX

A Detailed computational results on VRPSPDTW

This section provides the detailed computational results of the proposed MA-FIRD algorithm and the reference algorithms on the WC and JD benchmark instances for the VRPSPDTW.

Tables A.1 and A.2 show a detailed comparison of the proposed algorithm with the BKS and the reference algorithms on an instance-by-instance basis for the small and medium WC instances, where the hierarchical objective function aims to first minimize the number of vehicles and then the travel distance. The results of ALNS-PR and MATE are taken directly from [HS19] and [LTY21] respectively, while MATE* shows the results we obtained by re-running the MATE code on our computer.

The *Instance* column indicates the instance names, and *BKS* shows the best-known solutions in the literature. The M and D columns display the number of vehicles and travel distance obtained by each algorithm. $M\pm std$ and $D\pm std$ respectively show the average values and standard deviations of the number of vehicles and the travel distance obtained by the corresponding algorithm. The t column shows the running time in seconds for each algorithm, while the $\gamma\cdot\ t$ column reports the converted time using the time conversion ratio γ.

Table A.1 – Comparative results on the 9 small WC benchmark instances between MATE, MATE* and MA-FIRD.

Instance	MATE (Avg)				MATE* (Avg)			MA-FIRD (Avg)		
	M ± std	D ± std	t	$\gamma\cdot t$	M ± std	D ± std	t	M ± std	D ± std	t
RCdp10013	3.00±0.00	348.98±0.00	1.00	1.24	3.00±0.00	348.98±0.00	0.06	3.00±0.00	348.98±0.00	0.16
RCdp10042	2.00±0.00	216.69±0.00	1.00	1.24	2.00±0.00	216.69±0.00	0.16	2.00±0.00	216.69±0.00	0.28
RCdp10072	2.00±0.00	310.81±0.00	1.00	1.24	2.00±0.00	310.81±0.00	0.19	2.00±0.00	310.81±0.00	0.26
RCdp25015	5.00±0.00	551.05±0.00	1.00	1.24	5.00±0.00	551.05±0.00	0.25	5.00±0.00	551.05±0.00	0.55
RCdp25044	4.00±0.00	473.46±0.00	1.00	1.24	4.00±0.00	473.46±0.00	1.28	4.00±0.00	473.46±0.00	1.42
RCdp25075	5.00±0.00	540.87±0.00	1.00	1.24	5.00±0.00	540.87±0.00	1.23	5.00±0.00	540.87±0.00	1.25
RCdp50019	9.00±0.00	994.18±0.00	1.00	1.24	9.00±0.00	994.18±0.00	1.38	9.00±0.00	994.18±0.00	2.56
RCdp50046	6.00±0.00	733.21±0.00	9.00	11.16	6.00±0.00	733.21±0.00	9.59	6.00±0.00	733.21±0.00	19.25
RCdp50077	7.00±0.00	809.72±0.00	9.00	11.16	7.00±0.00	809.72±0.00	9.00	7.00±0.00	809.72±0.00	6.51
Mean	4.78±0.00	553.22±0.00	2.78	3.44	4.78±0.00	553.22±0.00	2.57	4.78±0.00	553.22±0.00	3.58

Table A.2 – Comparative results on the 56 medium WC benchmark instances between BKS, ALNS-PR, MATE, MATE* and MA-FIRD.

Instance	BKS		ALNS-PR					MATE					MATE*				MA-FIRD (Best)				MA-FIRD (Avg)			
	M	D	M	D	t	$\gamma \cdot t$	Gap	M	D	t	$\gamma \cdot t$	Gap	M	D	t	Gap	M	D	t	Gap	$M\pm$ std	$D\pm$ std	t	Gap
Cdp101	11	976.04	11	976.04	19.30	23.93	0.00%	11	976.04	102.04	108.16	0.00%	11	976.04	10.08	0.00%	11	976.04	14.46	0.00%	11.00±0.00	977.36±0.53	18.77	0.01%
Cdp102	10	941.49	10	941.49	28.11	34.86	0.00%	10	941.49	78.18	82.87	0.00%	10	941.49	32.23	0.00%	10	941.49	51.50	0.00%	10.00±0.00	941.49±0.00	62.36	0.00%
Cdp103	10	892.98	10	892.98	48.03	59.56	0.00%	10	892.98	78.66	83.38	0.00%	10	892.98	100.54	0.00%	10	892.98	136.24	0.00%	10.00±0.00	893.75±1.11	175.70	0.00%
Cdp104	10	871.40	10	871.40	46.51	57.67	0.00%	10	871.40	79.41	84.17	0.00%	10	871.40	111.53	0.00%	10	871.40	226.19	0.00%	10.00±0.00	871.51±0.60	247.13	0.00%
Cdp105	10	1053.12	10	1053.12	15.97	19.80	0.00%	10	1074.51	67.36	71.40	0.10%	11	980.55	10.00	9.16%	10	1053.12	11.09	0.00%	10.00±0.00	1053.12±0.00	15.67	0.00%
Cdp106	10	963.45	10	967.71	17.36	21.53	0.02%	10	963.45	100.87	106.92	0.00%	10	963.45	18.65	0.00%	10	963.45	19.56	0.00%	10.00±0.00	963.45±0.00	25.72	0.00%
Cdp107	10	987.64	10	987.64	18.06	22.39	0.00%	10	988.60	88.37	93.67	0.00%	10	1072.35	18.49	0.40%	10	987.64	23.07	0.00%	10.00±0.00	988.09±1.41	33.73	0.00%
Cdp108	10	932.50	10	932.88	18.22	22.59	0.00%	10	932.49	76.52	81.11	0.00%	10	932.50	90.64	0.00%	10	932.50	36.99	0.00%	10.00±0.00	932.50±0.00	45.96	0.00%
Cdp109	10	909.27	10	910.95	38.45	47.68	0.01%	10	909.27	78.86	83.59	0.00%	10	909.27	104.62	0.00%	10	909.27	107.32	0.00%	10.00±0.00	909.27±0.00	131.31	0.00%
Cdp201	3	591.56	3	591.56	24.37	30.22	0.00%	3	591.56	90.21	95.62	0.00%	3	591.56	24.27	0.00%	3	591.56	3.98	0.00%	3.00±0.00	591.56±0.00	4.43	0.00%
Cdp202	3	591.56	3	591.56	46.09	57.15	0.00%	3	591.56	158.59	168.11	0.00%	3	591.56	36.36	0.00%	3	591.56	10.09	0.00%	3.00±0.00	591.56±0.00	10.90	0.00%
Cdp203	3	591.17	3	591.17	44.19	54.80	0.00%	3	591.17	62.22	65.95	0.00%	3	591.17	54.60	0.00%	3	591.17	26.50	0.00%	3.00±0.00	591.17±0.00	29.62	0.00%
Cdp204	3	590.60	3	590.6	51.76	64.18	0.00%	3	590.60	78.93	83.67	0.00%	3	590.60	100.62	0.00%	3	590.60	44.40	0.00%	3.00±0.00	590.62±0.10	48.30	0.00%
Cdp205	3	588.88	3	588.88	36.35	45.07	0.00%	3	588.88	134.77	142.86	0.00%	3	588.88	30.74	0.00%	3	588.88	4.42	0.00%	3.00±0.00	588.88±0.00	4.87	0.00%
Cdp206	3	588.49	3	588.49	36.39	45.12	0.00%	3	588.49	176.76	187.37	0.00%	3	588.49	37.96	0.00%	3	588.49	7.29	0.00%	3.00±0.00	588.49±0.00	7.64	0.00%
Cdp207	3	588.29	3	588.29	39.83	49.39	0.00%	3	588.29	196.92	208.74	0.00%	3	588.29	39.85	0.00%	3	588.29	11.51	0.00%	3.00±0.00	588.29±0.00	12.49	0.00%
Cdp208	3	588.32	3	588.32	34.47	42.74	0.00%	3	588.32	215.96	228.92	0.00%	3	588.32	46.42	0.00%	3	588.32	10.33	0.00%	3.00±0.00	588.32±0.00	11.66	0.00%
Rdp101	19	1650.80	19	1650.80	22.86	28.35	0.00%	19	1650.80	53.82	57.05	0.00%	19	1650.80	6.56	0.00%	19	1650.80	10.27	0.00%	19.00±0.00	1651.24±1.04	12.28	0.00%
Rdp102	17	1486.12	17	1486.12	20.89	25.90	0.00%	17	1486.12	49.49	52.46	0.00%	17	1486.12	20.62	0.00%	17	1486.12	22.18	0.00%	17.00±0.00	1488.95±5.14	35.19	0.01%
Rdp103	13	1294.64	13	1297.01	17.83	22.11	0.01%	13	1294.64	78.82	83.55	0.00%	13	1294.64	72.16	0.00%	13	1294.64	131.76	0.00%	13.77±0.43	1233.31±34.96	58.16	5.39%
Rdp104	10	984.81	10	984.81	28.40	35.22	0.00%	10	984.81	79.19	83.94	0.00%	10	984.81	116.38	0.00%	**9**	**1026.42**	73.55	**-9.33%**	**9.97±0.18**	**987.83±7.95**	75.87	**-0.30%**
Rdp105	14	1377.11	14	1377.11	17.58	21.80	0.00%	14	1377.11	105.43	111.76	0.00%	14	1377.11	12.08	0.00%	14	1377.11	10.68	0.00%	14.00±0.00	1377.11±0.00	13.46	0.00%
Rdp106	12	1252.03	12	1252.03	27.54	34.15	0.00%	12	1252.03	78.81	83.54	0.00%	12	1252.03	44.62	0.00%	12	1252.03	103.97	0.00%	12.00±0.00	1257.26±4.05	59.03	0.02%
Rdp107	10	1112.55	10	1121.86	18.79	23.30	0.04%	10	1124.90	78.99	83.73	0.06%	10	1126.94	180.03	0.07%	10	1112.55	107.73	0.00%	10.00±0.00	1121.51±6.38	95.27	0.04%
Rdp108	9	965.22	9	965.54	20.60	25.54	0.00%	9	965.22	79.71	84.49	0.00%	9	965.22	603.82	0.00%	9	965.22	112.70	0.00%	9.00±0.00	970.27±4.84	119.47	0.03%
Rdp109	11	1194.73	11	1194.73	16.08	19.94	0.00%	11	1203.97	76.69	81.29	0.04%	12	1156.84	31.43	8.46%	11	1194.73	36.80	0.00%	11.47±0.51	1183.27±27.94	41.06	3.97%
Rdp110	10	1121.46	10	1148.20	19.13	23.72	0.13%	10	1166.47	78.42	83.13	0.21%	11	1081.56	112.89	9.28%	10	1121.46	68.97	0.00%	10.23±0.43	1131.49±34.63	87.89	2.26%
Rdp111	10	1098.84	10	1098.84	22.15	27.47	0.00%	10	1098.84	79.07	83.81	0.00%	10	1098.84	129.21	0.00%	10	1098.84	101.44	0.00%	10.17±0.38	1096.63±19.15	96.34	1.57%
Rdp112	9	1010.42	9	1010.42	28.63	35.50	0.00%	10	953.63	78.32	83.02	10.22%	10	953.63	174.89	10.22%	**9**	**997.27**	101.98	**-0.07%**	9.93±0.25	961.87±11.65	38.03	9.56%
Rdp201	4	1252.37	4	1253.23	33.44	41.47	0.01%	4	1252.37	78.80	83.53	0.00%	4	1252.37	67.51	0.00%	4	1252.37	22.12	0.00%	4.00±0.00	1253.57±2.29	24.55	0.01%
Rdp202	3	1191.70	3	1191.70	46.71	57.92	0.00%	3	1223.69	77.46	82.11	0.44%	3	1244.54	165.95	0.73%	3	1191.70	39.56	0.00%	3.00±0.00	1193.36±2.53	54.76	0.02%
Rdp203	3	939.58	3	946.28	84.91	105.29	0.10%	3	939.58	73.94	78.38	0.00%	3	939.58	585.88	0.00%	**3**	**939.50**	88.73	**0.00%**	3.00±0.00	941.67±1.49	96.49	0.03%
Rdp204	2	833.09	2	833.09	111.52	138.28	0.00%	2	835.28	78.54	83.25	0.05%	2	860.16	520.65	0.56%	**2**	**825.52**	240.22	**-0.16%**	2.00±0.00	834.61±3.56	124.19	0.03%
Rdp205	3	994.43	3	994.43	80.35	99.63	0.00%	3	994.43	80.49	85.32	0.00%	3	994.43	289.25	0.00%	3	994.43	32.95	0.00%	3.00±0.00	997.59±6.63	48.02	0.05%
Rdp206	3	906.14	3	913.68	89.88	111.45	0.11%	3	906.14	78.23	82.92	0.00%	3	906.14	341.72	0.00%	3	906.14	106.05	0.00%	3.00±0.00	908.81±3.91	84.06	0.04%
Rdp207	2	890.61	2	890.61	82.51	102.31	0.00%	3	811.51	71.13	75.40	39.28%	3	811.51	579.05	39.28%	2	890.61	101.82	0.00%	2.03±0.18	896.76±18.39	134.66	1.49%
Rdp208	2	726.82	2	726.82	100.25	124.31	0.00%	2	726.82	81.23	86.10	0.00%	2	728.22	487.98	0.03%	2	726.82	118.93	0.00%	2.00±0.00	732.89±4.71	90.82	0.13%
Rdp209	3	909.16	3	909.16	86.59	107.37	0.00%	3	909.16	76.27	80.85	0.00%	3	909.16	1060.62	0.00%	3	909.16	102.72	0.00%	3.00±0.00	911.51±2.87	86.53	0.03%
Rdp210	3	939.37	3	939.37	82.84	102.72	0.00%	3	939.37	78.07	82.75	0.00%	3	939.37	449.32	0.00%	3	939.37	106.08	0.00%	3.00±0.00	945.45±5.99	86.71	0.09%
Rdp211	2	904.44	2	904.44	85.87	106.48	0.00%	3	767.82	68.64	72.76	37.99%	3	767.82	412.68	37.99%	**2**	**885.71**	55.53	**-0.38%**	2.03±0.18	905.64±29.15	79.89	1.38%
RCdp101	14	1708.21	14	1776.58	10.59	13.13	0.23%	14	1708.21	27.13	28.76	0.00%	14	1708.21	20.63	0.00%	14	1708.21	15.41	0.00%	14.83±0.38	1647.08±27.81	23.14	5.40%
RCdp102	12	1570.28	12	1583.62	19.19	23.80	0.05%	12	1570.28	78.92	83.66	0.00%	12	1589.91	71.53	0.08%	12	1570.28	114.33	0.00%	12.03±0.18	1575.69±20.05	61.87	0.28%
RCdp103	11	1282.53	11	1283.52	29.08	36.06	0.00%	11	1282.53	78.64	83.36	0.00%	11	1282.53	204.05	0.00%	11	1282.53	111.63	0.00%	11.00±0.00	1285.74±5.18	99.93	0.01%
RCdp104	10	1171.37	10	1171.65	22.57	27.99	0.00%	10	1171.37	79.88	84.67	0.00%	10	1171.37	186.61	0.00%	10	1171.37	100.26	0.00%	10.00±0.00	1171.56±0.51	146.17	0.00%
RCdp105	13	1632.29	14	1548.96	16.95	21.02	6.94%	13	1646.36	72.62	76.98	0.05%	14	1548.36	32.51	6.93%	13	1632.29	39.36	0.00%	13.77±0.43	1570.50±39.01	51.43	5.33%
RCdp106	12	1392.47	12	1392.47	20.47	25.38	0.00%	12	1392.47	78.62	83.34	0.00%	12	1392.47	41.38	0.00%	12	1392.47	37.52	0.00%	12.00±0.00	1393.42±3.32	46.37	0.00%
RCdp107	11	1252.79	11	1255.06	21.22	26.31	0.01%	11	1252.79	78.67	83.39	0.00%	11	1252.79	105.82	0.00%	11	1252.79	101.39	0.00%	11.00±0.00	1256.00±2.84	101.35	0.01%
RCdp108	10	1194.40	10	1198.36	19.52	24.20	0.02%	10	1208.58	79.18	83.93	0.07%	11	1151.71	101.75	9.24%	**10**	**1177.98**	104.70	**-0.08%**	**10.00±0.00**	**1186.63±12.00**	99.92	**-0.04%**
RCdp201	4	1406.94	4	1406.94	26.68	33.08	0.00%	4	1406.94	78.89	83.62	0.00%	4	1406.94	78.58	0.00%	4	1406.94	20.52	0.00%	4.00±0.00	1409.57±9.13	29.81	0.03%
RCdp202	3	1412.52	3	1414.55	39.94	49.53	0.03%	4	1161.29	79.10	83.85	23.59%	4	1161.29	73.45	23.59%	**3**	**1365.65**	107.51	**-0.63%**	**3.00±0.00**	**1372.33±14.13**	66.64	**-0.54%**
RCdp203	3	1050.64	3	1050.64	83.95	104.10	0.00%	3	1056.96	77.17	81.80	0.09%	3	1060.45	1152.32	0.14%	**3**	**1049.62**	60.83	**-0.01%**	3.00±0.00	1058.63±5.98	70.81	0.11%
RCdp204	3	798.46	3	798.46	94.85	117.61	0.00%	3	798.46	71.01	75.27	0.00%	3	798.46	334.63	0.00%	3	798.46	103.14	0.00%	3.00±0.00	799.33±2.76	85.43	0.01%
RCdp205	4	1297.65	4	1297.65	33.58	41.64	0.00%	4	1297.65	77.82	82.49	0.00%	4	1297.65	168.17	0.00%	4	1297.65	100.06	0.00%	4.00±0.00	1298.80±2.14	73.07	0.01%
RCdp206	3	1146.32	3	1146.32	59.36	73.61	0.00%	3	1146.32	78.54	83.25	0.00%	3	1146.32	392.20	0.00%	3	1146.32	37.67	0.00%	3.00±0.00	1152.35±5.86	46.23	0.08%
RCdp207	3	1061.14	3	1061.84	79.06	98.03	0.01%	3	1061.14	84.47	89.54	0.00%	3	1061.14	461.54	0.00%	3	1061.14	104.54	0.00%	3.00±0.00	1061.95±2.28	87.89	0.01%
RCdp208	3	828.14	3	828.14	72.78	90.25	0.00%	3	828.44	72.98	77.36	0.00%	3	829.00	506.23	0.01%	3	828.14	162.63	0.00%	3.00±0.00	830.39±3.36	112.32	0.03%
Mean	7.29	1044.45	7.30	1045.68	42.12	52.23	0.14%	7.36	1037.92	86.03	91.19	2.00%	7.45	1033.92	201.15	2.79%	7.27	1043.34	70.77	-0.19%	7.36±0.06	1041.11±7.13	66.63	0.65%

For the 56 medium WC instances in Table A.2, a weight value of 2000 is assigned to the number of vehicles (M), and 1 to the travel distance (D) following [LTY21]. The *Gap* column shows the relative gap between the solutions and the BKS, calculated using Equation (A.1). A negative gap indicates that the algorithm has found a better solution than the BKS, and these improved values are highlighted in bold.

$$\text{Gap} = 100\% \cdot \frac{(2000 \cdot M + D) - (2000 \cdot M_{\text{BKS}} + D_{\text{BKS}})}{2000 \cdot M_{\text{BKS}} + D_{\text{BKS}}} \tag{A.1}$$

Table A.3 shows the comparison results for the large JD instances between MATE, MATE*, and MA-FIRD. Recall that for the JD instances, the objective is to minimize the total cost, which is the weighted sum of the number of vehicles and the travel distance. The columns f_{Best} and $f_{Avg} \pm std$ show the best and average total cost of each algorithm, including standard deviations. Using MATE's results as a baseline, the columns Gap_{Best}

Table A.3 – Comparative results on the 20 large-scale JD benchmark instances between MATE, MATE* and MA-FIRD.

Instance	MATE		MATE*								MA-FIRD							
	f_{Best}	f_{Avg}± std	M	D	f_{Best}	Gap_{Best}	M± std	D± std	f_{Avg}± std	Gap_{Avg}	M	D	f_{Best}	Gap_{Best}	M± std	D± std	f_{Avg}± std	Gap_{Avg}
F201	65106.00	66097.00±292.00	41	52364.90	64664.90	-0.68%	41.63±0.49	52647.61±213.67	65137.61±213.67	-1.45%	40	52083.40	**64083.40**	**-1.57%**	40.40±0.50	52420.04±374.59	**64540.04±374.59**	**-2.36%**
F202	65012.00	66038.00±422.00	43	51923.00	64823.00	-0.29%	43.00±0.00	52141.76±149.35	65041.76±149.35	-1.51%	42	51778.10	**64378.10**	**-0.98%**	42.20±0.48	52309.00±491.89	**64969.00±491.89**	**-1.62%**
F203	65980.00	67090.00±332.00	41	53293.30	65593.30	-0.59%	41.37±0.49	53636.95±196.48	66046.95±196.48	-1.55%	41	53247.40	**65547.40**	**-0.66%**	40.93±0.25	53762.45±375.53	**66042.45±375.53**	**-1.56%**
F204	64747.00	65851.00±326.00	42	51982.70	64582.70	-0.25%	42.13±0.35	52460.08±180.22	65100.08±180.22	-1.14%	41	51737.30	**64037.30**	**-1.10%**	41.30±0.47	52678.48±539.36	**65068.48±539.36**	**-1.19%**
F401	122319.00	123261.00±446.00	78	98157.70	121557.70	-0.62%	79.30±0.75	98636.59±256.89	122426.59±256.89	-0.68%	75	93376.00	**115876.00**	**-5.27%**	76.17±0.95	96172.30±1566.68	**119022.30±1566.68**	**-3.44%**
F402	126887.00	128091.00±410.00	83	101617.00	126517.00	-0.29%	82.47±0.90	102619.60±418.57	127359.60±418.57	-0.57%	79	98297.00	**121997.00**	**-3.85%**	79.57±0.77	100708.27±1260.18	**124578.27±1260.18**	**-2.74%**
F403	120130.00	122306.00±682.00	78	96729.80	120129.80	0.00%	79.30±0.60	97604.31±403.95	121394.31±403.95	-0.75%	74	93340.00	**115540.00**	**-3.82%**	75.80±0.85	95049.40±1398.61	**117789.40±1398.61**	**-3.69%**
F404	124517.00	125242.00±359.00	81	98897.50	123197.50	-1.06%	81.60±0.77	99879.39±452.51	124359.39±452.51	-0.70%	78	96051.00	**119451.00**	**-4.07%**	79.07±0.78	98081.70±1199.85	**121801.70±1199.85**	**-2.75%**
F601	182504.00	184119.00±608.00	116	147617.00	182417.00	-0.05%	115.92±0.89	148360.73±441.98	183137.65±441.98	-0.53%	109	141408.00	**174108.00**	**-4.60%**	110.97±1.10	144920.43±1783.13	**178210.43±1783.13**	**-3.21%**
F602	187236.00	188891.00±645.00	119	151059.00	186759.00	-0.25%	119.41±0.98	151888.00±591.75	187712.14±591.75	-0.62%	113	144417.00	**178317.00**	**-4.76%**	113.83±1.46	148246.53±2376.39	**182396.53±2376.39**	**-3.44%**
F603	186644.00	188050.00±621.00	118	149843.00	185243.00	-0.75%	118.63±1.01	151448.04±553.83	187036.93±553.83	-0.54%	110	144979.00	**177979.00**	**-4.64%**	113.63±1.25	148734.27±1812.42	**182824.27±1812.42**	**-2.78%**
F604	186289.00	188110.00±790.00	120	150449.00	186449.00	0.09%	119.67±1.12	151388.07±478.76	187288.07±478.76	-0.44%	114	144001.00	**178201.00**	**-4.34%**	115.00±1.55	147555.10±2227.07	**182055.10±2227.07**	**-3.22%**
F801	213661.00	214634.00±561.00	154	167407.00	213607.00	-0.03%	152.17±1.23	168743.57±610.66	214393.57±610.66	-0.11%	147	164225.00	**208325.00**	**-2.50%**	150.03±2.36	167414.03±2867.45	**212424.03±2867.45**	**-1.03%**
F802	212752.00	213276.00±292.00	150	166591.00	211591.00	-0.55%	151.00±1.05	167888.93±541.21	213188.93±541.21	-0.04%	145	162714.00	**206214.00**	**-3.07%**	148.57±1.76	165733.33±2238.98	**210303.33±2238.98**	**-1.39%**
F803	214126.00	214870.00±318.00	155	167790.00	214290.00	0.08%	153.60±1.40	168701.80±545.41	214781.80±545.41	-0.04%	150	165213.00	**210213.00**	**-1.83%**	152.93±2.23	168220.27±2641.75	**214100.27±2641.75**	**-0.36%**
F804	209431.00	210845.00±429.00	150	165302.00	210302.00	0.42%	148.93±1.46	165991.13±404.11	210671.13±404.11	-0.08%	145	160471.00	**203971.00**	**-2.61%**	147.57±2.31	164444.90±2467.06	**208714.90±2467.06**	**-1.01%**
F1001	312606.00	314914.00±1096.00	203	251707.00	312607.00	0.00%	203.41±1.74	253436.22±849.77	314458.44±849.77	-0.14%	195	244613.00	**303113.00**	**-3.04%**	197.27±2.32	248709.20±2547.42	**307889.20±2547.42**	**-2.23%**
F1002	309158.00	311718.00±1153.00	203	248731.00	309631.00	0.15%	202.41±1.50	250399.26±917.50	311121.48±917.50	-0.19%	193	241933.00	**299833.00**	**-3.02%**	195.73±2.83	245324.30±2909.40	**304044.30±2909.40**	**-2.46%**
F1003	311377.00	313089.00±981.00	204	249512.00	310712.00	-0.21%	203.87±1.58	251582.26±947.36	312743.13±947.36	-0.40%	195	242551.00	**301051.00**	**-3.32%**	196.90±2.72	246727.33±2729.98	**305797.33±2729.98**	**-2.61%**
F1004	308816.00	311415.00±943.00	204	248314.00	309514.00	0.23%	203.19±1.41	249761.69±840.35	310719.38±840.35	-0.22%	193	240655.00	**298555.00**	**-3.32%**	196.67±2.52	244625.53±2643.12	**303625.53±2643.12**	**-2.50%**
Mean	179464.90	180940.35±585.30	119.15	143464.40	179209.40	-0.23%	119.15±0.99	144460.80±499.72	180205.95±499.72	-0.59%	113.95	139354.51	**141764.89**	**-3.12%**	115.73±1.47	142091.84±1822.54	**144677.53±1822.54**	**-2.28%**

and Gap_{Avg} show the relative gap for the best and average values obtained by MATE* and MA-FIRD over 30 independent runs. So a negative gap indicates an improved result. The number of vehicles M and the travel distance D are also reported for reference.

B Detailed computational results for TGA

This section provides the detailed computational results of experiments for TGA, including the comparison between the proposed TGA-based algorithms (MA-NTGA and MA-ETGA) and their CPU-based counterparts (MA-N and MA-E) on the X (CVRP), GH (VRPTW), and JD (VRPSPDTW) benchmark instances, respectively, as well as the comparison between the TGA-based algorithms and the state-of-the-art algorithms.

MA-N and MA-E are CPU-based algorithms employing traditional local search evaluation methods, whereas MA-NTGA and MA-ETGA integrate the proposed tensor-based GPU acceleration (TGA). MA-N serves as the baseline algorithm for performance comparison.

Tables B.4-B.8 present detailed computational results based on 10 runs for four algorithm variants, MA-N, MA-E, MA-NTGA, and MA-ETGA, which are evaluated on three benchmark sets: 100 X (CVRP) instances [Uch+17], 90 selected GH (VRPTW) instances [GH99], and 20 JD (VRPSPDTW) instances [LTY21].

Each table reports the instance name (*Instance*), the number of customers (N_C), the average number of vehicles (M), the best objective value (f^*), and the average objective value (f) over 10 independent runs for each algorithm variant. The column "Gap(%)" indicates the percentage improvement in average objective value relative to MA-N, where a negative value corresponds to better performance.

For the X (CVRP) and GH (VRPTW) instances, the unified termination conditions

with time limits from the literature [Vid+12; Vid22; Koo+22] are adopted, and the corresponding limits are reported in column t (in seconds). For the *JD* (VRPSPDTW) instances, compound termination conditions are applied following [LH25], and the actual average computation times are reported in column t.

Tables B.9-B.13 show detailed comparative results between our TGA-based algorithms (MA-NTGA and MA-ETGA) and state-of-the-art methods on the three benchmark sets, respectively. For the CVRP and VRPTW, the BKS values are obtained from the CVRPLIB repository (`http://vrp.galgos.inf.puc-rio.br/`) as of December 25, 2025. For the CVRP, we use results from HGS-2012 [Vid+12] and HGS-CVRP [Vid22]. For the VRPTW, with the objective of minimizing the total travel distance, we report results from the DIMACS competition, including both the official DIMACS reference results and the champion team's algorithm, HGS-DIMACS [Koo+22]. For the VRPSPDTW, we report the best results from the state-of-the-art MA-FIRD method [LH25]. Our MA-NTGA and MA-ETGA results are derived from the best solutions obtained in the Tables B.4-B.8.

Each table reports the instance name (*Instance*), the number of customers (N_C), the best-known solution (BKS), the best objective value (f^*) achieved by each reference algorithm, and the corresponding gap (Gap(%)) relative to the BKS. The time column (t) follows the same definition and settings as in Tables B.4-B.8.

C Detailed computational results on MDVRPs

This section provides the detailed computational results of the proposed MDFIHA and MDFIHA-ETGA, along with the BKS and the results of the reference algorithms, on the benchmark instances of the classic Multi-Depot Vehicle Routing Problem (MDVRP) and two MDVRP variants: the Multi-Depot Vehicle Routing Problem with Time Windows (MDVRPTW) and the Multi-Depot Open Vehicle Routing Problem (MDOVRP). The results are shown in Tables C.14-C.19. The *Instance* column lists the instance names, and N_C, N_D, and N_M denote the number of customers, the number of depots, and the number of available vehicles per depot, respectively. The *BKS* column reports the best-known solutions published in the literature, while the '*' symbol or the *Optima* column indicates instances with known optimal values. For each algorithm, we report the best solution found (f_{best}), the converted computation time in seconds ($\gamma \cdot t$) using the conversion ratio γ, and the percentage gap relative to the BKS (Gap(%)), computed as $\text{Gap}(\%) = \frac{f-\text{BKS}}{\text{BKS}} \times 100$. A negative gap indicates that the algorithm has found a better solution than the BKS,

Table B.4 – Comparative results on the 100 X benchmark instances for MA-N, MA-E, MA-NTGA, and MA-ETGA.

Instance	N_C	t	MA-N			MA-E				MA-NTGA				MA-ETGA			
			M	f^*	f	M	f^*	f	Gap(%)	M	f^*	f	Gap(%)	M	f^*	f	Gap(%)
X-n101-k25	100	240.0	26.0	27591.0	27597.8	26.0	27591.0	27603.4	0.02	26.0	27591.0	27605.0	0.03	26.0	27591.0	27599.8	0.01
X-n106-k14	105	252.0	14.0	26381.0	26481.4	14.0	26392.0	26435.4	-0.17	14.0	26403.0	26464.2	-0.06	14.0	26403.0	26464.6	-0.06
X-n110-k13	109	261.6	13.0	14971.0	14988.9	13.0	14971.0	14992.6	0.02	13.0	14971.0	14999.8	0.07	13.0	14971.0	15001.4	0.08
X-n115-k10	114	273.6	10.0	12747.0	12758.1	10.0	12747.0	12773.9	0.12	10.0	12747.0	12751.7	-0.05	10.0	12747.0	12754.8	-0.03
X-n120-k6	119	285.6	6.0	13332.0	13397.8	6.0	13332.0	13380.5	-0.13	6.0	13332.0	13373.8	-0.18	6.0	13332.0	13363.6	-0.26
X-n125-k30	124	297.6	30.0	55562.0	55641.0	30.0	55551.0	55636.3	-0.01	30.0	55626.0	55721.0	0.14	30.0	55600.0	55682.4	0.07
X-n129-k18	128	307.2	18.0	29015.0	29259.0	18.0	29033.0	29138.5	-0.41	18.0	29012.0	29361.4	0.35	18.0	29020.0	29247.7	-0.04
X-n134-k13	133	319.2	13.0	10918.0	10955.8	13.0	10934.0	10955.9	0.00	13.0	10920.0	10953.1	-0.02	13.0	10918.0	10972.8	0.15
X-n139-k10	138	331.2	10.1	13590.0	13604.4	10.2	13590.0	13612.9	0.06	10.2	13590.0	13610.8	0.05	10.1	13590.0	13603.9	0.00
X-n143-k7	142	340.8	7.0	15700.0	15757.8	7.0	15740.0	15813.4	0.35	7.0	15726.0	15765.4	0.05	7.0	15740.0	15814.7	0.36
X-n148-k46	147	352.8	46.8	43449.0	43550.5	46.5	43541.0	43674.3	0.28	46.4	43449.0	43614.4	0.15	46.7	43461.0	43693.4	0.33
X-n153-k22	152	364.8	23.0	21234.0	21315.5	23.0	21250.0	21284.4	-0.15	23.0	21262.0	21326.5	0.05	23.1	21282.0	21382.2	0.31
X-n157-k13	156	374.4	13.0	16876.0	16921.2	13.0	16888.0	16926.9	0.03	13.0	16876.0	16917.5	-0.02	13.0	16876.0	16906.1	-0.09
X-n162-k11	161	386.4	11.0	14138.0	14164.4	11.0	14138.0	14191.9	0.19	11.0	14138.0	14161.7	-0.02	11.0	14147.0	14179.9	0.11
X-n167-k10	166	398.4	10.0	20583.0	20720.4	10.0	20583.0	20687.8	-0.16	10.0	20583.0	20709.2	-0.05	10.0	20557.0	20782.7	0.30
X-n172-k51	171	410.4	53.0	45705.0	45835.7	53.0	45629.0	45783.0	-0.11	53.1	45610.0	45739.7	-0.21	53.0	45616.0	45735.5	-0.22
X-n176-k26	175	420.0	26.0	47843.0	48065.6	26.0	47840.0	48015.3	-0.10	26.0	47915.0	48033.9	-0.07	26.0	47899.0	48060.4	-0.01
X-n181-k23	180	432.0	23.0	25593.0	25677.1	23.0	25597.0	25664.2	-0.05	23.0	25644.0	25685.4	0.03	23.0	25593.0	25667.7	-0.04
X-n186-k15	185	444.0	15.0	24173.0	24311.2	15.0	24196.0	24315.6	0.02	15.0	24154.0	24354.5	0.18	15.0	24180.0	24300.9	-0.04
X-n190-k8	189	453.6	8.0	16995.0	17130.6	8.0	17050.0	17178.0	0.28	8.0	17057.0	17121.1	-0.06	8.0	17015.0	17116.5	-0.08
X-n195-k51	194	465.6	52.4	44321.0	44502.5	52.5	44392.0	44557.4	0.12	52.7	44361.0	44447.0	-0.12	52.8	44313.0	44440.5	-0.14
X-n200-k36	199	477.6	36.0	58674.0	58796.4	36.0	58669.0	58733.4	-0.11	36.0	58730.0	58825.1	0.05	36.0	58711.0	58805.8	0.02
X-n204-k19	203	487.2	19.0	19700.0	19741.2	19.0	19598.0	19720.1	-0.11	19.0	19610.0	19707.8	-0.17	19.0	19570.0	19699.5	-0.21
X-n209-k16	208	499.2	16.0	30673.0	30847.6	16.0	30730.0	30850.4	0.01	16.0	30736.0	30928.4	0.26	16.0	30756.0	30910.5	0.20
X-n214-k11	213	511.2	11.0	11030.0	11067.6	11.0	10886.0	10974.4	-0.84	11.0	10927.0	11020.9	-0.42	11.0	11005.0	11055.8	-0.11
X-n219-k73	218	523.2	73.0	117616.0	117693.2	73.0	117598.0	117695.0	0.00	73.0	117624.0	117706.7	0.01	73.0	117616.0	117716.0	0.02
X-n223-k34	222	532.8	34.0	40665.0	40821.8	34.0	40600.0	40777.3	-0.11	34.0	40587.0	40803.0	-0.05	34.0	40627.0	40858.8	0.09
X-n228-k23	227	544.8	23.0	25865.0	26065.8	23.0	25843.0	25917.5	-0.57	23.0	25870.0	26030.3	-0.14	23.0	25881.0	25954.1	-0.43
X-n233-k16	232	556.8	17.0	19341.0	19491.9	17.0	19357.0	19498.3	0.03	17.0	19333.0	19405.2	-0.44	17.0	19337.0	19426.2	-0.34
X-n237-k14	236	566.4	14.0	27320.0	27538.1	14.0	27186.0	27515.0	-0.08	14.0	27318.0	27496.0	-0.15	14.0	27346.0	27495.8	-0.15
X-n242-k48	241	578.4	48.0	83334.0	83566.8	48.0	83106.0	83384.3	-0.22	48.0	83443.0	83510.0	-0.07	48.0	83319.0	83714.8	0.18
X-n247-k50	246	590.4	51.0	37440.0	37510.1	51.0	37392.0	37491.4	-0.05	51.0	37387.0	37435.9	-0.20	51.5	37387.0	37592.8	0.22
X-n251-k28	250	600.0	28.0	39008.0	39171.9	28.0	38927.0	39013.3	-0.40	28.0	38940.0	39099.1	-0.19	28.0	38966.0	39156.0	-0.04
X-n256-k16	255	612.0	17.0	18984.0	19073.2	16.1	18915.0	18975.9	-0.51	16.9	18920.0	19039.9	-0.17	16.8	18929.0	19028.7	-0.23
X-n261-k13	260	624.0	13.0	26805.0	27045.8	13.0	26683.0	26896.1	-0.55	13.0	26752.0	26895.6	-0.56	13.0	26672.0	26982.8	-0.23
X-n266-k58	265	636.0	58.3	76281.0	76450.2	58.7	76204.0	76966.4	0.68	58.4	75942.0	76170.8	-0.37	58.5	76109.0	76392.1	-0.08
X-n270-k35	269	645.6	36.0	35888.0	35993.4	36.0	36047.0	36215.1	0.62	36.0	35525.0	35597.1	-1.10	36.0	35543.0	35690.4	-0.84
X-n275-k28	274	657.6	28.0	21480.0	21545.8	28.0	21394.0	21515.3	-0.14	28.0	21452.0	21506.3	-0.18	28.0	21412.0	21502.2	-0.20
X-n280-k17	279	669.6	17.0	33738.0	33990.0	17.0	33653.0	33876.1	-0.34	17.0	33656.0	34022.0	0.09	17.0	33689.0	33912.8	-0.23
X-n284-k15	283	679.2	15.0	20723.0	20988.4	15.0	20623.0	20785.6	-0.97	15.0	20632.0	20938.1	-0.24	15.0	20724.0	20998.9	0.05
X-n289-k60	288	691.2	61.6	97395.0	97898.3	61.6	97156.0	97593.3	-0.31	61.4	96330.0	96598.7	-1.33	61.5	96271.0	96690.6	-1.23
X-n294-k50	293	703.2	51.0	47910.0	48202.0	51.0	47988.0	48283.3	0.17	51.0	47421.0	47559.1	-1.33	51.0	47531.0	47612.6	-1.22
X-n298-k31	297	712.8	31.0	34661.0	34825.1	31.0	34407.0	34691.1	-0.38	31.0	34416.0	34770.6	-0.16	31.0	34552.0	34769.6	-0.16
X-n303-k21	302	724.8	21.0	21943.0	22009.2	21.0	21931.0	22021.3	0.05	21.0	21868.0	21980.1	-0.13	21.0	21951.0	22000.3	-0.04
X-n308-k13	307	736.8	13.0	25993.0	26304.3	13.0	25993.0	26194.1	-0.42	13.0	26044.0	26213.5	-0.35	13.0	25924.0	26168.4	-0.52
X-n313-k71	312	748.8	72.2	96003.0	96299.8	72.6	95676.0	96004.5	-0.31	72.6	94891.0	95104.5	-1.24	72.9	94800.0	95212.2	-1.13
X-n317-k53	316	758.4	53.0	78521.0	78568.8	53.0	78478.0	78517.6	-0.07	53.0	78515.0	78562.8	-0.01	53.0	78479.0	78575.4	0.01
X-n322-k28	321	770.4	28.5	30527.0	30639.7	28.0	30075.0	30155.7	-1.58	28.1	30082.0	30185.0	-1.48	28.3	30097.0	30219.5	-1.37
X-n327-k20	326	782.4	20.0	27910.0	28039.1	20.0	27657.0	27895.3	-0.51	20.0	27738.0	27953.4	-0.31	20.0	27850.0	27997.3	-0.15
X-n331-k15	330	792.0	15.0	31398.0	31642.5	15.0	31278.0	31541.3	-0.32	15.0	31408.0	31650.0	0.02	15.0	31340.0	31532.3	-0.35
X-n336-k84	335	804.0	85.7	142668.0	143056.7	85.5	142707.0	143265.8	0.15	86.1	140544.0	140962.2	-1.46	86.2	140645.0	141100.6	-1.37
X-n344-k43	343	823.2	43.4	42823.0	43094.0	43.0	42501.0	42660.1	-1.01	43.2	42468.0	42617.4	-1.11	43.4	42509.0	42710.4	-0.89
X-n351-k40	350	840.0	40.7	26973.0	27065.4	41.0	26995.0	27162.8	0.36	41.0	26247.0	26376.9	-2.54	41.0	26381.0	26471.1	-2.20
X-n359-k29	358	859.2	29.0	52268.0	52568.2	29.0	51951.0	52352.6	-0.41	29.0	52090.0	52325.9	-0.46	29.0	52178.0	52405.3	-0.31
X-n367-k17	366	878.4	17.0	23100.0	23268.3	17.0	22968.0	23215.5	-0.23	17.0	22986.0	23189.4	-0.34	17.0	22902.0	23172.0	-0.41
X-n376-k94	375	900.0	94.0	147893.0	148041.4	94.0	147821.0	148159.3	0.08	94.0	147838.0	147943.0	-0.07	94.0	147873.0	147994.2	-0.03
X-n384-k52	383	919.2	53.2	67889.0	68359.2	53.0	67978.0	68673.2	0.46	53.0	66906.0	67111.2	-1.83	53.0	67103.0	67321.1	-1.52
X-n393-k38	392	940.8	38.0	38838.0	39007.4	38.0	38494.0	38757.8	-0.64	38.0	38596.0	38766.0	-0.62	38.0	38620.0	38799.0	-0.53
X-n401-k29	400	960.0	29.0	66754.0	66913.8	29.0	66527.0	66865.2	-0.07	29.0	66671.0	66840.6	-0.11	29.0	66511.0	66713.8	-0.30
X-n411-k19	410	984.0	19.0	20003.0	20165.6	19.0	19943.0	20041.1	-0.62	19.0	19914.0	20095.3	-0.35	19.0	20022.0	20137.7	-0.14
X-n420-k130	419	1005.6	130.1	109131.0	109670.8	130.0	108436.0	108664.0	-0.92	130.0	108294.0	108639.6	-0.94	130.0	108558.0	108753.5	-0.84
X-n429-k61	428	1027.2	61.8	67802.0	67995.3	61.4	67022.0	67817.0	-0.26	61.8	66112.0	66459.7	-2.26	61.7	66286.0	66579.9	-2.08
X-n439-k37	438	1051.2	37.0	36661.0	36951.7	37.0	36603.0	36788.0	-0.44	37.0	36664.0	36852.5	-0.27	37.0	36600.0	36865.6	-0.23
X-n449-k29	448	1075.2	29.0	57556.0	57900.6	29.0	55980.0	56882.8	-1.76	29.0	56185.0	56398.9	-2.59	29.0	56349.0	56490.7	-2.44
X-n459-k26	458	1099.2	26.0	24819.0	24981.5	26.0	24445.0	24657.2	-1.30	26.0	24467.0	24601.5	-1.52	26.0	24395.0	24530.1	-1.81
X-n469-k138	468	1123.2	140.0	226110.0	227111.5	139.5	225788.0	226470.2	-0.28	139.9	223454.0	223927.8	-1.40	140.1	224239.0	224592.6	-1.11
X-n480-k70	479	1149.6	70.0	90956.0	91251.6	70.2	90604.0	91356.4	0.11	70.0	90269.0	90556.4	-0.76	70.0	90341.0	90672.3	-0.63
X-n491-k59	490	1176.0	59.3	68970.0	69496.3	59.5	68344.0	69422.0	-0.11	59.1	67578.0	67738.7	-2.53	59.3	67629.0	68003.2	-2.15
X-n502-k39	501	1202.4	39.0	69658.0	69702.5	39.0	69584.0	69657.3	-0.06	39.0	69566.0	69644.5	-0.08	39.0	69564.0	69632.7	-0.10
X-n513-k21	512	1228.8	21.0	24676.0	24981.5	21.0	24523.0	24618.4	-1.45	21.0	24429.0	24541.7	-1.76	21.0	24509.0	24605.7	-1.50
X-n524-k153	523	1255.2	154.5	155854.0	156229.2	155.1	155306.0	155615.7	-0.39	155.1	154764.0	154855.1	-0.88	155.8	155331.0	155589.2	-0.41
X-n536-k96	535	1284.0	96.4	98029.0	98761.4	96.3	96656.0	97181.2	-1.60	96.8	95622.0	96144.3	-2.65	97.0	96355.0	96649.3	-2.14
X-n548-k50	547	1312.8	50.0	87605.0	88013.0	50.0	87194.0	87780.0	-0.26	50.0	87426.0	87830.1	-0.21	50.0	87699.0	87904.2	-0.12
X-n561-k42	560	1344.0	42.0	43697.0	44026.9	42.0	43288.0	43462.2	-1.28	42.0	43317.0	43569.3	-1.04	42.0	43388.0	43557.7	-1.07
X-n573-k30	572	1372.8	30.0	51476.0	51763.3	30.0	51001.0	51272.5	-0.95	30.0	51461.0	51598.4	-0.32	30.0	51379.0	51527.5	-0.46
X-n586-k159	585	1404.0	160.5	196519.0	197834.8	159.2	192888.0	193819.4	-2.03	159.5	191836.0	192148.6	-2.87	159.6	192258.0	192706.1	-2.59
X-n599-k92	598	1435.2	94.2	112220.0	112918.7	94.3	113478.0	114071.3	1.02	94.0	110447.0	110693.4	-1.97	94.1	110534.0	110993.6	-1.70

i.e., an improved upper bound. Note that we report the original computation time (t) for CGL [CGL97], as its conversion ratio could not be determined due to unavailable CPU

Table B.5 – Comparative results on the 100 X benchmark instances for MA-N, MA-E, MA-NTGA, and MA-ETGA (continued).

Instance	N_C	t	MA-N			MA-E				MA-NTGA				MA-ETGA			
			M	f^*	f	M	f^*	f	Gap(%)	M	f^*	f	Gap(%)	M	f^*	f	Gap(%)
X-n613-k62	612	1468.8	62.0	62921.0	63335.5	62.0	61330.0	62086.2	-1.97	62.0	60571.0	60698.4	-4.16	62.0	60687.0	60861.2	-3.91
X-n627-k43	626	1502.4	43.5	64169.0	64712.4	43.3	63621.0	64700.9	-0.02	43.0	63015.0	63320.0	-2.15	43.0	63096.0	63305.3	-2.17
X-n641-k35	640	1536.0	35.0	65633.0	65826.2	35.0	64905.0	65975.3	0.23	35.0	64516.0	64847.8	-1.49	35.0	64684.0	64848.6	-1.49
X-n655-k131	654	1569.6	131.0	107273.0	107394.4	131.0	107343.0	107908.7	0.48	131.0	107248.0	107338.8	-0.05	131.0	107269.0	107354.4	-0.04
X-n670-k130	669	1605.6	133.3	149871.0	150288.7	133.7	148928.0	149451.6	-0.56	133.0	147491.0	147731.1	-1.70	134.8	147803.0	148602.4	-1.12
X-n685-k75	684	1641.6	75.2	72087.0	72821.3	75.6	70656.0	71193.7	-2.24	75.6	69319.0	69634.2	-4.38	75.9	69510.0	69859.7	-4.07
X-n701-k44	700	1680.0	44.0	84492.0	84977.4	44.0	83091.0	85201.1	0.26	44.0	83308.0	83691.9	-1.51	44.0	83523.0	83884.9	-1.29
X-n716-k35	715	1716.0	35.0	45189.0	45474.4	35.0	44554.0	45146.1	-0.72	35.0	44188.0	44385.2	-2.40	35.0	44119.0	44416.6	-2.33
X-n733-k159	732	1756.8	159.0	144775.0	146324.9	159.5	139958.0	141074.4	-3.59	160.0	137896.0	138373.5	-5.43	160.0	138426.0	138629.7	-5.26
X-n749-k98	748	1795.2	98.0	81196.0	82731.1	98.2	80324.0	81056.9	-2.02	98.4	78836.0	79020.1	-4.49	98.8	78970.0	79310.6	-4.13
X-n766-k71	765	1836.0	71.0	121067.0	122634.3	71.0	118294.0	119175.5	-2.82	71.0	117135.0	117724.3	-4.00	71.0	116629.0	117443.4	-4.23
X-n783-k48	782	1876.8	48.0	75520.0	76614.0	48.0	76064.0	76842.8	0.30	48.0	73679.0	74168.9	-3.19	48.0	73718.0	74257.3	-3.08
X-n801-k40	800	1920.0	40.0	75041.0	75537.1	40.0	74504.0	75187.5	-0.46	40.0	74533.0	74693.4	-1.12	40.0	74639.0	74936.4	-0.80
X-n819-k171	818	1963.2	174.2	163641.0	164406.7	173.6	163549.0	164922.8	0.31	172.5	159878.0	160581.7	-2.33	173.0	161000.0	161310.7	-1.88
X-n837-k142	836	2006.4	143.7	202047.0	203973.9	143.3	200161.0	201612.7	-1.16	142.6	196109.0	196812.1	-3.51	142.9	196834.0	197951.2	-2.95
X-n856-k95	855	2052.0	95.0	90428.0	90581.7	95.0	90042.0	91351.7	0.85	95.0	89939.0	90175.2	-0.45	95.0	90023.0	90185.2	-0.44
X-n876-k59	875	2100.0	59.0	102264.0	103002.7	59.0	101369.0	103433.1	0.42	59.0	100718.0	101072.3	-1.87	59.0	100931.0	101196.8	-1.75
X-n895-k37	894	2145.6	37.3	57758.0	58860.9	37.9	58238.0	58654.9	-0.35	38.0	55170.0	55486.2	-5.73	38.0	55189.0	55592.3	-5.55
X-n916-k207	915	2196.0	208.7	342404.0	343969.5	207.0	336363.0	337671.2	-1.83	208.0	332728.0	333422.3	-3.07	208.0	334515.0	335608.6	-2.43
X-n936-k151	935	2244.0	161.0	138669.0	139042.5	160.9	136899.0	137666.4	-0.99	160.0	134944.0	135214.0	-2.75	160.8	135026.0	135596.0	-2.48
X-n957-k87	956	2294.4	87.0	87218.0	87386.1	87.0	87100.0	88270.2	1.01	87.0	86748.0	86950.7	-0.50	87.0	86780.0	87044.4	-0.39
X-n979-k58	978	2347.2	58.0	122751.0	123705.7	58.1	122219.0	123003.6	-0.57	58.0	120156.0	120769.3	-2.37	58.0	120106.0	120659.8	-2.46
X-n1001-k43	1000	2400.0	43.0	76258.0	77086.4	43.0	74715.0	76189.8	-1.16	43.0	74271.0	74704.4	-3.09	43.0	74212.0	74612.7	-3.21
MEAN	-	989.3	50.8	64611.6	64962.8	50.8	64182.9	64607.9	-0.38	50.8	63716.1	63926.8	-1.05	50.8	63810.9	64043.9	-0.94

Table B.6 – Comparative results on the 90 GH benchmark instances for MA-N, MA-E, MA-NTGA, and MA-ETGA.

Instance	N_C	t	MA-N			MA-E				MA-NTGA				MA-ETGA			
			M	f^*	f	M	f^*	f	Gap(%)	M	f^*	f	Gap(%)	M	f^*	f	Gap(%)
C1_2_1	200	1800.0	20.0	2698.6	2698.6	20.0	2698.6	2698.6	0.00	20.0	2698.6	2698.6	0.00	20.0	2698.6	2698.6	0.00
C1_2_5	200	1800.0	20.0	2694.9	2694.9	20.0	2694.9	2694.9	0.00	20.0	2694.9	2694.9	0.00	20.0	2694.9	2694.9	0.00
C1_2_10	200	1800.0	19.0	2624.7	2625.6	19.0	2624.7	2626.5	0.04	19.0	2624.7	2625.6	0.00	19.0	2624.7	2625.5	0.00
C2_2_1	200	1800.0	6.9	1922.1	1922.3	7.0	1922.1	1922.1	-0.01	7.0	1922.1	1922.3	0.00	7.0	1922.1	1922.1	-0.01
C2_2_5	200	1800.0	7.0	1869.6	1869.6	7.0	1869.6	1869.6	0.00	6.9	1869.6	1869.9	0.01	7.0	1869.6	1869.6	0.00
C2_2_10	200	1800.0	6.9	1791.2	1793.8	7.0	1791.2	1792.0	-0.10	6.9	1791.2	1792.6	-0.07	7.0	1791.2	1792.1	-0.09
R1_2_1	200	1800.0	23.0	4667.2	4669.1	23.0	4667.2	4667.4	-0.03	23.0	4667.2	4672.6	0.08	22.9	4667.2	4668.1	-0.02
R1_2_5	200	1800.0	20.0	4053.2	4053.9	20.0	4053.2	4054.4	0.01	20.0	4053.2	4054.3	0.01	20.0	4053.2	4053.7	-0.01
R1_2_10	200	1800.0	18.0	3293.1	3300.6	18.0	3293.1	3298.7	-0.06	18.0	3293.1	3300.7	0.00	18.0	3293.8	3303.4	0.08
R2_2_1	200	1800.0	9.6	3482.0	3551.1	9.1	3497.1	3573.3	0.63	9.4	3507.5	3566.2	0.43	10.1	3478.4	3516.3	-0.98
R2_2_5	200	1800.0	7.8	3065.2	3072.0	7.2	3061.1	3087.2	0.49	7.6	3069.1	3086.8	0.48	7.2	3063.6	3089.4	0.57
R2_2_10	200	1800.0	5.4	2556.7	2608.9	5.8	2555.7	2599.1	-0.37	6.0	2557.8	2589.4	-0.75	6.0	2556.1	2592.9	-0.61
RC1_2_1	200	1800.0	20.0	3516.9	3518.2	19.8	3516.9	3518.9	0.02	20.0	3516.9	3517.4	-0.02	19.9	3516.9	3517.0	-0.03
RC1_2_5	200	1800.0	19.0	3325.6	3328.6	19.0	3326.3	3328.1	-0.01	19.0	3325.6	3327.5	-0.03	19.0	3326.3	3328.2	-0.01
RC1_2_10	200	1800.0	18.6	2997.6	3010.9	18.4	2996.5	3008.2	-0.09	18.3	2993.8	3003.6	-0.24	18.3	2996.9	3005.1	-0.19
RC2_2_1	200	1800.0	8.8	2797.4	2806.7	8.7	2799.0	2806.9	0.01	9.0	2797.4	2806.2	-0.02	9.1	2797.4	2801.3	-0.19
RC2_2_5	200	1800.0	7.2	2491.4	2503.7	7.1	2492.1	2502.3	-0.05	7.2	2491.4	2503.7	0.00	7.2	2492.1	2500.5	-0.12
RC2_2_10	200	1800.0	5.0	1989.3	2005.1	4.8	1992.1	2004.4	-0.03	5.0	2000.0	2005.6	0.03	4.9	1997.0	2007.9	0.14
C1_4_1	400	3600.0	40.0	7138.8	7138.8	40.0	7138.8	7138.8	0.00	40.0	7138.8	7138.8	0.00	40.0	7138.8	7138.8	0.00
C1_4_5	400	3600.0	40.0	7138.8	7138.8	40.0	7138.8	7138.8	0.00	40.0	7138.8	7138.8	0.00	40.0	7138.8	7138.8	0.00
C1_4_10	400	3600.0	37.0	6830.1	6835.6	37.0	6826.7	6828.2	-0.11	37.0	6825.7	6826.9	-0.13	37.0	6826.5	6827.9	-0.11
C2_4_1	400	3600.0	12.0	4100.3	4100.3	12.0	4100.3	4100.3	0.00	12.0	4100.3	4100.3	0.00	12.0	4100.3	4100.3	0.00
C2_4_5	400	3600.0	12.0	3923.2	3926.3	12.0	3927.5	3927.5	0.03	12.0	3927.5	3927.5	0.03	12.0	3923.2	3924.9	-0.03
C2_4_10	400	3600.0	12.1	3665.2	3675.4	12.1	3665.1	3671.8	-0.10	12.0	3665.1	3675.1	-0.01	12.0	3665.1	3669.9	-0.15
R1_4_1	400	3600.0	41.6	10339.5	10370.7	41.3	10331.9	10354.2	-0.16	41.3	10313.8	10323.9	-0.45	41.5	10337.6	10356.9	-0.13
R1_4_5	400	3600.0	37.9	9221.0	9255.6	37.9	9204.4	9223.3	-0.35	38.0	9208.7	9220.9	-0.37	37.6	9198.4	9216.7	-0.42
R1_4_10	400	3600.0	37.0	8231.6	8267.4	37.0	8151.2	8206.5	-0.74	36.9	8109.5	8155.2	-1.36	37.0	8138.7	8171.4	-1.16
R2_4_1	400	3600.0	17.2	7556.1	7679.3	17.7	7525.2	7601.2	-1.02	17.1	7559.7	7661.0	-0.24	18.7	7526.8	7547.5	-1.72
R2_4_5	400	3600.0	12.6	6580.0	6658.2	12.4	6578.5	6685.2	0.41	12.4	6568.1	6678.4	0.30	12.5	6577.1	6661.9	0.06
R2_4_10	400	3600.0	10.3	5707.6	5750.2	9.8	5699.4	5752.7	0.04	10.2	5690.2	5738.3	-0.21	10.1	5694.0	5737.6	-0.22
RC1_4_1	400	3600.0	37.0	8568.9	8605.2	37.0	8535.2	8559.1	-0.54	37.6	8524.0	8580.0	-0.29	37.0	8527.2	8557.1	-0.56
RC1_4_5	400	3600.0	37.0	8255.2	8292.9	37.0	8206.1	8236.3	-0.68	37.0	8196.4	8211.3	-0.98	37.0	8206.7	8226.8	-0.80
RC1_4_10	400	3600.0	36.8	7719.9	7795.8	36.8	7670.0	7730.9	-0.83	36.8	7634.5	7697.4	-1.26	36.5	7634.0	7686.3	-1.40
RC2_4_1	400	3600.0	16.1	6153.7	6173.7	15.8	6147.3	6171.4	-0.04	16.4	6156.2	6172.8	-0.01	15.9	6148.7	6175.4	0.03
RC2_4_5	400	3600.0	13.0	5402.9	5419.0	13.0	5401.5	5418.6	-0.01	13.3	5400.1	5418.7	-0.01	13.0	5394.7	5408.0	-0.20
RC2_4_10	400	3600.0	8.2	4284.4	4334.3	8.1	4276.5	4302.1	-0.74	8.5	4266.6	4295.4	-0.90	8.5	4261.6	4300.0	-0.79
C1_6_1	600	3600.0	60.0	14076.6	14076.6	60.0	14076.6	14076.6	0.00	60.0	14076.6	14076.6	0.00	60.0	14076.6	14076.6	0.00
C1_6_5	600	3600.0	60.0	14066.8	14067.2	60.0	14066.8	14066.9	0.00	60.0	14066.8	14066.8	0.00	60.0	14066.8	14066.9	0.00
C1_6_10	600	3600.0	56.4	13733.7	13800.5	56.0	13650.3	13665.4	-0.98	56.0	13626.3	13651.3	-1.08	56.0	13640.1	13654.9	-1.05
C2_6_1	600	3600.0	18.3	7752.2	7762.3	18.0	7752.2	7753.6	-0.11	18.3	7752.2	7760.2	-0.03	18.0	7752.2	7753.2	-0.12
C2_6_5	600	3600.0	18.0	7553.8	7554.1	18.1	7553.8	7561.4	0.10	18.4	7553.8	7576.9	0.30	18.0	7553.8	7554.2	0.00
C2_6_10	600	3600.0	18.0	7138.1	7155.0	18.0	7132.2	7143.0	-0.17	18.0	7131.5	7148.0	-0.10	18.0	7136.0	7142.8	-0.17

Table B.7 – Comparative results on the 90 *GH* benchmark instances for MA-N, MA-E, MA-NTGA, and MA-ETGA (continued).

Instance	N_C	t	MA-N			MA-E				MA-NTGA				MA-ETGA			
			M	f^*	f	M	f^*	f	Gap(%)	M	f^*	f	Gap(%)	M	f^*	f	Gap(%)
R1_6_1	600	3600.0	61.2	21711.0	21858.9	59.1	21377.2	21515.9	-1.57	58.1	21314.5	21352.3	-2.32	58.2	21315.5	21385.5	-2.17
R1_6_5	600	3600.0	56.6	19884.8	19992.1	56.2	19552.0	19641.4	-1.75	56.0	19435.8	19523.9	-2.34	56.0	19463.8	19545.8	-2.23
R1_6_10	600	3600.0	55.2	18226.7	18355.5	55.0	17958.7	18040.4	-1.72	55.0	17711.0	17799.5	-3.03	55.0	17795.1	17861.1	-2.69
R2_6_1	600	3600.0	20.0	15654.4	16012.2	21.4	15259.8	15595.6	-2.60	21.4	15274.1	15693.4	-1.99	23.4	15194.9	15418.9	-3.71
R2_6_5	600	3600.0	15.7	14519.5	14786.5	17.3	13962.7	14226.6	-3.79	18.2	13895.0	14120.2	-4.51	18.3	13926.5	14150.4	-4.30
R2_6_10	600	3600.0	13.0	12465.8	12633.6	13.1	12024.2	12245.4	-3.07	13.8	12010.4	12084.8	-4.34	13.6	11988.6	12090.3	-4.30
RC1_6_1	600	3600.0	57.2	17448.6	17523.0	56.8	17179.5	17215.4	-1.76	56.4	17037.9	17111.6	-2.35	56.7	17069.3	17113.1	-2.34
RC1_6_5	600	3600.0	56.7	17123.0	17237.3	56.3	16872.9	16916.0	-1.86	56.0	16682.9	16744.9	-2.86	56.1	16696.2	16755.3	-2.80
RC1_6_10	600	3600.0	55.8	16394.2	16535.9	55.6	16043.1	16160.4	-2.27	55.7	15870.0	15943.1	-3.58	55.6	15915.0	16018.5	-3.13
RC2_6_1	600	3600.0	21.5	12118.8	12205.5	21.8	11982.7	12037.9	-1.37	22.1	12005.7	12062.6	-1.17	22.0	11986.7	12047.7	-1.29
RC2_6_5	600	3600.0	19.6	11290.7	11419.1	19.1	11114.8	11192.7	-1.98	19.4	11111.9	11188.1	-2.02	19.3	11109.5	11156.0	-2.30
RC2_6_10	600	3600.0	12.6	9219.8	9394.8	12.6	9040.2	9114.0	-2.99	12.5	9028.3	9075.7	-3.40	12.5	9036.1	9077.4	-3.38
C1_8_1	800	3600.0	80.0	25156.9	25156.9	80.0	25156.9	25157.3	0.00	80.0	25156.9	25156.9	0.00	80.0	25156.9	25157.3	0.00
C1_8_5	800	3600.0	80.0	25138.6	25138.7	80.0	25138.6	25139.3	0.00	80.0	25138.6	25138.6	0.00	80.0	25138.6	25139.4	0.00
C1_8_10	800	3600.0	73.5	24908.6	25203.7	73.0	24251.3	24376.8	-3.28	73.0	24094.3	24141.2	-4.22	73.0	24091.5	24152.7	-4.17
C2_8_1	800	3600.0	24.0	11631.9	11632.6	24.0	11631.9	11634.1	0.01	24.0	11631.9	11632.0	0.00	24.0	11631.9	11634.5	0.02
C2_8_5	800	3600.0	24.0	11399.6	11411.2	24.0	11395.6	11402.8	-0.07	24.0	11395.6	11398.0	-0.12	24.0	11395.6	11399.2	-0.11
C2_8_10	800	3600.0	24.3	11103.0	11310.4	24.0	10961.3	10994.4	-2.79	24.0	10950.2	10989.0	-2.84	24.0	10957.6	10972.2	-2.99
R1_8_1	800	3600.0	81.7	37482.8	37712.0	79.9	37038.5	37171.0	-1.43	79.7	36641.8	36720.4	-2.63	79.1	36704.2	36847.0	-2.29
R1_8_5	800	3600.0	75.3	34841.2	34988.0	74.5	34243.4	34433.5	-1.58	74.0	33815.4	33937.0	-3.00	74.2	33849.3	34060.3	-2.65
R1_8_10	800	3600.0	72.9	32847.2	32979.8	73.2	31919.5	32049.6	-2.82	73.0	31436.6	31594.0	-4.20	73.0	31573.6	31738.5	-3.76
R2_8_1	800	3600.0	21.8	26502.0	26865.9	25.1	25228.8	25575.5	-4.80	26.5	25237.3	25426.9	-5.36	25.7	25129.0	25405.7	-5.44
R2_8_5	800	3600.0	17.5	24738.1	24984.2	21.6	23131.2	23708.4	-5.11	23.2	23026.9	23211.1	-7.10	24.0	22929.3	23132.6	-7.41
R2_8_10	800	3600.0	16.0	21497.2	21696.9	18.3	20638.9	20959.2	-3.40	19.3	20154.0	20274.6	-6.55	18.3	20230.8	20389.4	-6.03
RC1_8_1	800	3600.0	75.7	31037.5	31241.0	74.9	30503.6	30670.5	-1.83	74.6	30186.5	30325.2	-2.93	74.3	30193.5	30382.3	-2.75
RC1_8_5	800	3600.0	74.8	30316.2	30506.7	74.2	29794.4	30042.0	-1.52	74.0	29514.3	29578.7	-3.04	74.0	29556.2	29644.5	-2.83
RC1_8_10	800	3600.0	73.3	29634.7	30175.1	73.0	29136.8	29295.5	-2.92	73.0	28564.8	28644.9	-5.07	73.1	28786.5	28876.6	-4.30
RC2_8_1	800	3600.0	26.0	19796.1	19931.0	25.4	19263.7	19338.9	-2.97	25.6	19279.2	19341.9	-2.96	25.9	19239.5	19309.8	-3.12
RC2_8_5	800	3600.0	23.0	18175.5	18316.8	22.7	17609.3	17714.7	-3.29	22.6	17540.8	17611.1	-3.85	22.5	17595.1	17666.8	-3.55
RC2_8_10	800	3600.0	16.4	15182.1	15356.6	16.5	14684.0	14826.1	-3.45	16.0	14470.6	14531.7	-5.37	16.0	14478.0	14533.1	-5.36
C1_10_1	1000	7200.0	100.0	42444.8	42444.9	100.0	42445.8	42447.2	0.01	100.0	42444.8	42444.8	0.00	100.0	42444.8	42446.2	0.00
C1_10_5	1000	7200.0	100.0	42435.5	42436.1	100.0	42435.5	42437.0	0.00	100.0	42434.8	42435.7	0.00	100.0	42435.5	42436.7	0.00
C1_10_10	1000	7200.0	91.7	41332.2	41790.0	91.7	40425.3	40645.7	-2.74	90.9	40030.1	40253.0	-3.68	91.2	40056.8	40231.0	-3.73
C2_10_1	1000	7200.0	30.0	16841.1	16841.1	30.0	16841.1	16841.1	0.00	30.0	16841.1	16842.2	0.01	30.0	16841.1	16841.3	0.00
C2_10_5	1000	7200.0	30.0	16532.2	16553.7	30.0	16525.6	16538.2	-0.09	30.0	16521.9	16533.2	-0.12	30.0	16522.9	16538.5	-0.09
C2_10_10	1000	7200.0	30.0	15965.0	16158.4	30.0	15747.6	15790.8	-2.28	30.0	15773.0	15786.3	-2.30	30.0	15750.8	15775.0	-2.37
R1_10_1	1000	7200.0	98.1	55067.3	55329.0	95.9	53996.7	54106.9	-2.21	94.9	53488.9	53610.4	-3.11	94.7	53522.0	53634.4	-3.06
R1_10_5	1000	7200.0	94.2	52497.3	52757.8	93.7	51523.6	51756.3	-1.90	93.0	50971.1	51099.8	-3.14	93.0	50895.6	51060.4	-3.22
R1_10_10	1000	7200.0	93.0	49984.0	50525.7	93.0	49220.4	49360.5	-2.31	92.6	48015.0	48390.5	-4.23	92.7	48454.5	48854.9	-3.31
R2_10_1	1000	7200.0	26.0	39831.9	40458.1	30.5	37241.1	38080.2	-5.88	32.2	37323.2	37719.2	-6.77	31.2	37079.2	37649.1	-6.94
R2_10_5	1000	7200.0	21.7	37403.0	37695.6	26.8	34597.7	35281.2	-6.40	25.8	34497.3	35102.8	-6.88	27.6	34422.9	34800.6	-7.68
R2_10_10	1000	7200.0	20.1	32256.9	32525.9	22.2	31012.4	31171.7	-4.16	23.3	30118.6	30250.6	-7.00	22.7	30109.1	30319.4	-6.78
RC1_10_1	1000	7200.0	93.6	47760.7	47994.4	92.2	46798.5	47043.7	-1.98	91.5	46285.6	46415.6	-3.29	91.7	46362.2	46639.6	-2.82
RC1_10_5	1000	7200.0	92.2	47019.3	47412.2	91.6	46257.7	46448.5	-2.03	91.1	45593.0	45718.9	-3.57	91.4	45795.4	45971.7	-3.04
RC1_10_10	1000	7200.0	90.5	46181.7	46622.4	91.0	45140.7	45367.2	-2.69	90.8	44251.9	44403.5	-4.76	91.0	44627.1	44802.3	-3.90
RC2_10_1	1000	7200.0	28.7	29458.1	29634.6	29.4	28355.3	28514.0	-3.78	29.5	28324.6	28440.2	-4.03	29.1	28275.9	28349.2	-4.34
RC2_10_5	1000	7200.0	25.4	26869.9	27242.3	26.0	26111.1	26277.3	-3.54	25.9	25983.3	26076.8	-4.28	25.7	25961.7	26064.3	-4.32
RC2_10_10	1000	7200.0	19.9	22910.0	23103.9	20.3	22234.7	22409.7	-3.00	19.9	21912.3	21987.3	-4.83	20.1	21885.4	22003.5	-4.76
MEAN	-	3960.0	37.1	16857.9	16977.3	37.2	16529.0	16625.3	-1.32	37.3	16410.3	16479.4	-1.80	37.3	16423.1	16495.1	-1.80

Table B.8 – Comparative results on the 20 *JD* benchmark instances for MA-N, MA-E, MA-NTGA, and MA-ETGA.

Instance	N_C	MA-N				MA-E					MA-NTGA					MA-ETGA				
		M	f^*	f	t	M	f^*	f	t	Gap(%)	M	f^*	f	t	Gap(%)	M	f^*	f	t	Gap(%)
F201	200	41.0	64458.1	64690.3	908.5	41.0	64437.6	64705.0	439.5	0.02	41.0	64297.8	64550.4	424.0	-0.22	41.1	64483.1	64812.3	516.1	0.19
F202	200	42.8	64687.0	64944.5	1089.2	42.9	64848.5	65002.1	376.6	0.09	42.9	64758.0	64875.4	415.1	-0.11	42.8	64698.3	64969.9	608.1	0.04
F203	200	41.2	65667.3	65994.2	1067.7	41.2	65619.3	66042.1	511.8	0.07	41.2	65526.9	65967.3	495.1	-0.04	41.0	65664.1	65854.0	614.9	-0.21
F204	200	41.7	64309.7	64755.8	1164.3	41.6	64461.7	64780.3	442.2	0.04	41.5	64179.1	64629.0	532.0	-0.20	41.8	64310.7	64835.5	641.4	0.12
F401	400	76.4	117113.0	117937.6	6489.8	76.6	117068.0	117779.9	2422.3	-0.13	76.4	116999.0	117512.5	841.8	-0.36	76.8	117057.0	118032.7	1056.2	0.08
F402	400	80.2	122364.0	123627.1	6798.0	79.8	122606.0	123308.8	2674.6	-0.26	80.0	122429.0	123059.2	865.5	-0.46	79.9	122369.0	123023.9	1567.1	-0.49
F403	400	76.4	116591.0	117077.2	7046.5	76.6	116762.0	117098.9	2176.6	0.02	75.9	115920.0	116589.7	874.3	-0.42	76.6	116693.0	117369.2	1109.8	0.25
F404	400	79.6	120232.0	121007.1	7008.3	79.4	119546.0	120633.0	2782.8	-0.31	79.0	119242.0	120287.4	882.7	-0.59	79.5	119777.0	120558.1	1456.1	-0.37
F601	600	112.1	175304.0	177418.2	7205.8	110.9	174113.0	175018.4	5890.7	-1.35	110.4	173141.0	174054.6	1731.3	-1.90	111.2	173896.0	175616.6	1571.3	-1.02
F602	600	115.9	181240.0	182229.8	7205.8	114.1	178435.0	180070.3	5572.2	-1.19	113.5	177893.0	178792.2	1503.2	-1.89	114.5	179052.0	179960.1	1836.8	-1.25
F603	600	115.1	180712.0	181790.1	7205.4	113.4	178135.0	178972.0	5898.9	-1.55	112.5	177320.0	178291.0	1378.0	-1.92	113.8	177997.0	178974.8	1567.1	-1.55
F604	600	117.3	180749.0	182080.4	7206.2	115.0	177918.0	178753.8	6859.3	-1.83	113.7	176527.0	177811.5	1651.1	-2.34	114.7	177806.0	179063.2	1712.8	-1.66
F801	800	157.1	215645.0	217342.7	7217.0	150.1	208302.0	209804.6	7201.6	-3.47	147.1	205657.0	207008.4	2362.7	-4.75	148.8	207300.0	208363.2	2589.9	-4.13
F802	800	154.5	212788.0	214773.9	7218.6	148.8	206510.0	208008.9	7202.0	-3.15	146.4	204164.0	205171.9	2302.7	-4.47	147.0	205251.0	206450.9	2696.9	-3.88
F803	800	158.3	214338.0	217564.7	7219.9	152.5	209854.0	211410.9	7201.6	-2.83	150.6	207664.0	208869.3	2167.3	-4.00	150.5	207585.0	209159.9	2737.8	-3.86
F804	800	155.2	212164.0	214123.8	7217.9	147.5	204671.0	205643.9	7201.7	-3.96	145.4	202271.0	203238.5	2165.4	-5.08	146.6	203116.0	204404.2	2546.4	-4.54
F1001	1000	204.8	311597.0	312801.1	7305.1	197.8	301712.0	303817.7	7204.1	-2.87	193.6	297090.0	297435.0	4617.2	-4.91	194.1	297325.0	299324.2	3434.4	-4.31
F1002	1000	204.3	307184.0	309877.0	7335.6	197.2	299313.0	300400.0	7204.2	-3.06	192.0	292779.0	294447.6	4010.4	-4.98	194.0	294712.0	296616.4	3511.3	-4.28
F1003	1000	204.8	309496.0	311153.3	7378.4	199.0	300427.0	302536.1	7204.9	-2.77	192.7	292581.0	295103.3	4556.9	-5.16	193.5	294946.0	297179.9	3187.9	-4.49
F1004	1000	205.0	308060.0	310114.8	7315.7	197.8	298988.0	300791.0	7204.4	-3.01	192.7	292222.0	293919.6	4875.1	-5.22	193.8	293905.0	296191.1	3223.3	-4.49
MEAN	-	119.2	177235.0	178565.2	5930.2	116.2	173686.4	174728.9	4683.6	-1.57	114.4	171633.0	172580.7	1932.6	-2.45	115.1	172397.2	173538.0	1909.3	-1.99

Table B.9 – Comparative results on the 100 X benchmark instances for HGS-2012, HGS-CVRP, MA-NTGA, and MA-ETGA.

Instance	N_C	BKS	t	HGS-2012		HGS-CVRP		MA-NTGA		MA-ETGA	
				f^*	Gap(%)	f^*	Gap(%)	f^*	Gap(%)	f^*	Gap(%)
X-n101-k25	100	27591.0	240.0	27591.0	0.00	27591.0	0.00	27591.0	0.00	27591.0	0.00
X-n106-k14	105	26362.0	252.0	26387.0	0.09	26364.0	0.01	26403.0	0.16	26403.0	0.16
X-n110-k13	109	14971.0	261.6	14971.0	0.00	14971.0	0.00	14971.0	0.00	14971.0	0.00
X-n115-k10	114	12747.0	273.6	12747.0	0.00	12747.0	0.00	12747.0	0.00	12747.0	0.00
X-n120-k6	119	13332.0	285.6	13332.0	0.00	13332.0	0.00	13332.0	0.00	13332.0	0.00
X-n125-k30	124	55539.0	297.6	55539.0	0.00	55539.0	0.00	55626.0	0.16	55600.0	0.11
X-n129-k18	128	28940.0	307.2	28940.0	0.00	28940.0	0.00	29012.0	0.25	29020.0	0.28
X-n134-k13	133	10916.0	319.2	10916.0	0.00	10916.0	0.00	10920.0	0.04	10918.0	0.02
X-n139-k10	138	13590.0	331.2	13590.0	0.00	13590.0	0.00	13590.0	0.00	13590.0	0.00
X-n143-k7	142	15700.0	340.8	15700.0	0.00	15700.0	0.00	15726.0	0.17	15740.0	0.25
X-n148-k46	147	43448.0	352.8	43448.0	0.00	43448.0	0.00	43449.0	0.00	43461.0	0.03
X-n153-k22	152	21220.0	364.8	21220.0	0.00	21225.0	0.02	21262.0	0.20	21282.0	0.29
X-n157-k13	156	16876.0	374.4	16876.0	0.00	16876.0	0.00	16876.0	0.00	16876.0	0.00
X-n162-k11	161	14138.0	386.4	14138.0	0.00	14138.0	0.00	14138.0	0.00	14147.0	0.06
X-n167-k10	166	20557.0	398.4	20557.0	0.00	20557.0	0.00	20583.0	0.13	20557.0	0.00
X-n172-k51	171	45607.0	410.4	45607.0	0.00	45607.0	0.00	45610.0	0.01	45616.0	0.02
X-n176-k26	175	47812.0	420.0	47812.0	0.00	47812.0	0.00	47915.0	0.22	47899.0	0.18
X-n181-k23	180	25569.0	432.0	25569.0	0.00	25569.0	0.00	25644.0	0.29	25593.0	0.09
X-n186-k15	185	24145.0	444.0	24145.0	0.00	24145.0	0.00	24154.0	0.04	24180.0	0.14
X-n190-k8	189	16980.0	453.6	16986.0	0.04	16980.0	0.00	17057.0	0.45	17015.0	0.21
X-n195-k51	194	44225.0	465.6	44225.0	0.00	44225.0	0.00	44361.0	0.31	44313.0	0.20
X-n200-k36	199	58578.0	477.6	58578.0	0.00	58578.0	0.00	58730.0	0.26	58711.0	0.23
X-n204-k19	203	19565.0	487.2	19565.0	0.00	19565.0	0.00	19610.0	0.23	19570.0	0.03
X-n209-k16	208	30656.0	499.2	30656.0	0.00	30656.0	0.00	30736.0	0.26	30756.0	0.33
X-n214-k11	213	10856.0	511.2	10856.0	0.00	10856.0	0.00	10927.0	0.65	11005.0	1.37
X-n219-k73	218	117595.0	523.2	117595.0	0.00	117595.0	0.00	117624.0	0.02	117616.0	0.02
X-n223-k34	222	40437.0	532.8	40437.0	0.00	40437.0	0.00	40587.0	0.37	40627.0	0.47
X-n228-k23	227	25742.0	544.8	25742.0	0.00	25742.0	0.00	25870.0	0.50	25881.0	0.54
X-n233-k16	232	19230.0	556.8	19230.0	0.00	19230.0	0.00	19333.0	0.54	19337.0	0.56
X-n237-k14	236	27042.0	566.4	27044.0	0.01	27042.0	0.00	27318.0	1.02	27346.0	1.12
X-n242-k48	241	82751.0	578.4	82771.0	0.02	82771.0	0.02	83443.0	0.84	83319.0	0.69
X-n247-k50	246	37274.0	590.4	37278.0	0.01	37274.0	0.00	37387.0	0.30	37387.0	0.30
X-n251-k28	250	38684.0	600.0	38699.0	0.04	38684.0	0.00	38940.0	0.66	38966.0	0.73
X-n256-k16	255	18839.0	612.0	18880.0	0.22	18839.0	0.00	18920.0	0.43	18929.0	0.48
X-n261-k13	260	26558.0	624.0	26570.0	0.05	26558.0	0.00	26752.0	0.73	26672.0	0.43
X-n266-k58	265	75478.0	636.0	75608.0	0.17	75478.0	0.00	75942.0	0.61	76109.0	0.84
X-n270-k35	269	35291.0	645.6	35303.0	0.03	35303.0	0.03	35525.0	0.66	35543.0	0.71
X-n275-k28	274	21245.0	657.6	21245.0	0.00	21245.0	0.00	21452.0	0.97	21412.0	0.79
X-n280-k17	279	33503.0	669.6	33542.0	0.12	33506.0	0.01	33656.0	0.46	33689.0	0.56
X-n284-k15	283	20215.0	679.2	20225.0	0.05	20231.0	0.08	20632.0	2.06	20724.0	2.52
X-n289-k60	288	95151.0	691.2	95181.0	0.03	95242.0	0.10	96330.0	1.24	96271.0	1.18
X-n294-k50	293	47161.0	703.2	47210.0	0.10	47167.0	0.01	47421.0	0.55	47531.0	0.78
X-n298-k31	297	34231.0	712.8	34231.0	0.00	34231.0	0.00	34416.0	0.54	34552.0	0.94
X-n303-k21	302	21736.0	724.8	21754.0	0.08	21739.0	0.01	21868.0	0.61	21951.0	0.99
X-n308-k13	307	25859.0	736.8	25865.0	0.02	25862.0	0.01	26044.0	0.72	25924.0	0.25
X-n313-k71	312	94043.0	748.8	94050.0	0.01	94045.0	0.00	94891.0	0.90	94800.0	0.80
X-n317-k53	316	78355.0	758.4	78355.0	0.00	78355.0	0.00	78515.0	0.20	78479.0	0.16
X-n322-k28	321	29834.0	770.4	29857.0	0.08	29834.0	0.00	30082.0	0.83	30097.0	0.88
X-n327-k20	326	27532.0	782.4	27576.0	0.16	27532.0	0.00	27738.0	0.75	27850.0	1.16
X-n331-k15	330	31102.0	792.0	31107.0	0.02	31102.0	0.00	31408.0	0.98	31340.0	0.77
X-n336-k84	335	139111.0	804.0	139305.0	0.14	139205.0	0.07	140544.0	1.03	140645.0	1.10
X-n344-k43	343	42050.0	823.2	42067.0	0.04	42061.0	0.03	42468.0	0.99	42509.0	1.09
X-n351-k40	350	25896.0	840.0	25948.0	0.20	25924.0	0.11	26247.0	1.36	26381.0	1.87
X-n359-k29	358	51505.0	859.2	51598.0	0.18	51566.0	0.12	52090.0	1.14	52178.0	1.31
X-n367-k17	366	22814.0	878.4	22814.0	0.00	22814.0	0.00	22986.0	0.75	22902.0	0.39
X-n376-k94	375	147713.0	900.0	147713.0	0.00	147713.0	0.00	147838.0	0.08	147873.0	0.11
X-n384-k52	383	65928.0	919.2	66088.0	0.24	65997.0	0.10	66906.0	1.48	67103.0	1.78
X-n393-k38	392	38260.0	940.8	38260.0	0.00	38260.0	0.00	38596.0	0.88	38620.0	0.94
X-n401-k29	400	66154.0	960.0	66256.0	0.15	66209.0	0.08	66671.0	0.78	66511.0	0.54
X-n411-k19	410	19712.0	984.0	19718.0	0.03	19716.0	0.02	19914.0	1.02	20022.0	1.57
X-n420-k130	419	107798.0	1005.6	107831.0	0.03	107810.0	0.01	108294.0	0.46	108558.0	0.71
X-n429-k61	428	65449.0	1027.2	65524.0	0.11	65484.0	0.05	66112.0	1.01	66286.0	1.28
X-n439-k37	438	36391.0	1051.2	36413.0	0.06	36395.0	0.01	36664.0	0.75	36600.0	0.57
X-n449-k29	448	55233.0	1075.2	55484.0	0.45	55306.0	0.13	56185.0	1.72	56349.0	2.02
X-n459-k26	458	24139.0	1099.2	24182.0	0.18	24139.0	0.00	24467.0	1.36	24395.0	1.06
X-n469-k138	468	221824.0	1123.2	222131.0	0.14	221916.0	0.04	223454.0	0.73	224239.0	1.09
X-n480-k70	479	89449.0	1149.6	89561.0	0.13	89498.0	0.05	90269.0	0.92	90341.0	1.00
X-n491-k59	490	66483.0	1176.0	66653.0	0.26	66569.0	0.13	67578.0	1.65	67629.0	1.72

benchmark score. For our proposed algorithms, we report the best solution found (f_{best}), the average solution (f_{avg}) and average computation time ($\gamma \cdot t$) over 10 independent runs. The best solutions are highlighted in bold. The improved upper bounds (a total of 70) are underlined.

Table B.10 – Comparative results on the 100 X benchmark instances for HGS-2012, HGS-CVRP, MA-NTGA, and MA-ETGA (continued).

Instance	N_C	BKS	t	HGS-2012		HGS-CVRP		MA-NTGA		MA-ETGA	
				f^*	Gap(%)	f^*	Gap(%)	f^*	Gap(%)	f^*	Gap(%)
X-n502-k39	501	69226.0	1202.4	69264.0	0.05	69230.0	0.01	69566.0	0.49	69564.0	0.49
X-n513-k21	512	24201.0	1228.8	24201.0	0.00	24201.0	0.00	24429.0	0.94	24509.0	1.27
X-n524-k153	523	154593.0	1255.2	154713.0	0.08	154646.0	0.03	154764.0	0.11	155331.0	0.48
X-n536-k96	535	94846.0	1284.0	95125.0	0.29	95040.0	0.20	95622.0	0.82	96355.0	1.59
X-n548-k50	547	86700.0	1312.8	86888.0	0.22	86710.0	0.01	87426.0	0.84	87699.0	1.15
X-n561-k42	560	42717.0	1344.0	42738.0	0.05	42726.0	0.02	43317.0	1.40	43388.0	1.57
X-n573-k30	572	50673.0	1372.8	50797.0	0.24	50757.0	0.17	51461.0	1.56	51379.0	1.39
X-n586-k159	585	190316.0	1404.0	190576.0	0.14	190470.0	0.08	191836.0	0.80	192258.0	1.02
X-n599-k92	598	108451.0	1435.2	108712.0	0.24	108605.0	0.14	110447.0	1.84	110534.0	1.92
X-n613-k62	612	59535.0	1468.8	59698.0	0.27	59636.0	0.17	60571.0	1.74	60687.0	1.93
X-n627-k43	626	62164.0	1502.4	62377.0	0.34	62238.0	0.12	63015.0	1.37	63096.0	1.50
X-n641-k35	640	63682.0	1536.0	63976.0	0.46	63782.0	0.16	64516.0	1.31	64684.0	1.57
X-n655-k131	654	106780.0	1569.6	106841.0	0.06	106785.0	0.00	107248.0	0.44	107269.0	0.46
X-n670-k130	669	146332.0	1605.6	146833.0	0.34	146640.0	0.21	147491.0	0.79	147803.0	1.01
X-n685-k75	684	68205.0	1641.6	68414.0	0.31	68288.0	0.12	69319.0	1.63	69510.0	1.91
X-n701-k44	700	81923.0	1680.0	82302.0	0.46	82075.0	0.19	83308.0	1.69	83523.0	1.95
X-n716-k35	715	43373.0	1716.0	43541.0	0.39	43459.0	0.20	44188.0	1.88	44119.0	1.72
X-n733-k159	732	136187.0	1756.8	136405.0	0.16	136323.0	0.10	137896.0	1.25	138426.0	1.64
X-n749-k98	748	77269.0	1795.2	77671.0	0.52	77563.0	0.38	78836.0	2.03	78970.0	2.20
X-n766-k71	765	114417.0	1836.0	114834.0	0.36	114679.0	0.23	117135.0	2.38	116629.0	1.93
X-n783-k48	782	72386.0	1876.8	72797.0	0.57	72704.0	0.44	73679.0	1.79	73718.0	1.84
X-n801-k40	800	73305.0	1920.0	73678.0	0.51	73396.0	0.12	74533.0	1.68	74639.0	1.82
X-n819-k171	818	158121.0	1963.2	158639.0	0.33	158391.0	0.17	159878.0	1.11	161000.0	1.82
X-n837-k142	836	193737.0	2006.4	194444.0	0.36	194103.0	0.19	196109.0	1.22	196834.0	1.60
X-n856-k95	855	88965.0	2052.0	89110.0	0.16	88986.0	0.02	89939.0	1.09	90023.0	1.19
X-n876-k59	875	99299.0	2100.0	99765.0	0.47	99596.0	0.30	100718.0	1.43	100931.0	1.64
X-n895-k37	894	53860.0	2145.6	54134.0	0.51	54023.0	0.30	55170.0	2.43	55189.0	2.47
X-n916-k207	915	329179.0	2196.0	329918.0	0.22	329572.0	0.12	332728.0	1.08	334515.0	1.62
X-n936-k151	935	132715.0	2244.0	133102.0	0.29	133121.0	0.31	134944.0	1.68	135026.0	1.74
X-n957-k87	956	85465.0	2294.4	85709.0	0.29	85506.0	0.05	86748.0	1.50	86780.0	1.54
X-n979-k58	978	118976.0	2347.2	119353.0	0.32	119180.0	0.17	120156.0	0.99	120106.0	0.95
X-n1001-k43	1000	72355.0	2400.0	72824.0	0.65	72678.0	0.45	74271.0	2.65	74212.0	2.57
MEAN	-	63106.7	989.3	63222.3	0.13	63166.6	0.06	63716.1	0.83	63810.9	0.92

Table B.11 – Comparative results on the 90 GH benchmark instances for DIMACS, HGS-DIMACS, MA-NTGA, and MA-ETGA.

Instance	N_C	BKS	t	DIMACS		HGS-DIMACS		MA-NTGA		MA-ETGA	
				f^*	Gap(%)	f^*	Gap(%)	f^*	Gap(%)	f^*	Gap(%)
C1_2_1	200	2698.6	1800.0	2698.6	0.00	2698.6	0.00	2698.6	0.00	2698.6	0.00
C1_2_5	200	2694.9	1800.0	2694.9	0.00	2694.9	0.00	2694.9	0.00	2694.9	0.00
C1_2_10	200	2624.7	1800.0	2624.7	0.00	2624.7	0.00	2624.7	0.00	2624.7	0.00
C2_2_1	200	1922.1	1800.0	1922.1	0.00	1922.1	0.00	1922.1	0.00	1922.1	0.00
C2_2_5	200	1869.6	1800.0	1869.6	0.00	1869.6	0.00	1869.6	0.00	1869.6	0.00
C2_2_10	200	1791.2	1800.0	1791.2	0.00	1791.2	0.00	1791.2	0.00	1791.2	0.00
R1_2_1	200	4667.2	1800.0	4667.2	0.00	4667.2	0.00	4667.2	0.00	4667.2	0.00
R1_2_5	200	4053.2	1800.0	4053.2	0.00	4053.2	0.00	4053.2	0.00	4053.2	0.00
R1_2_10	200	3293.1	1800.0	3293.1	0.00	3293.1	0.00	3293.1	0.00	3293.8	0.02
R2_2_1	200	3468.0	1800.0	3468.0	0.00	3468.0	0.00	3507.5	1.14	3478.4	0.30
R2_2_5	200	3061.1	1800.0	3061.1	0.00	3063.8	0.09	3069.1	0.26	3063.6	0.08
R2_2_10	200	2549.4	1800.0	2549.4	0.00	2549.4	0.00	2557.8	0.33	2556.1	0.26
RC1_2_1	200	3516.9	1800.0	3516.9	0.00	3516.9	0.00	3516.9	0.00	3516.9	0.00
RC1_2_5	200	3325.6	1800.0	3325.6	0.00	3325.6	0.00	3325.6	0.00	3326.3	0.02
RC1_2_10	200	2990.5	1800.0	2990.5	0.00	2994.9	0.15	2993.8	0.11	2996.9	0.21
RC2_2_1	200	2797.4	1800.0	2797.4	0.00	2797.4	0.00	2797.4	0.00	2797.4	0.00
RC2_2_5	200	2491.4	1800.0	2491.4	0.00	2491.4	0.00	2491.4	0.00	2492.1	0.03
RC2_2_10	200	1989.2	1800.0	1989.2	0.00	1989.2	0.00	2000.0	0.54	1997.0	0.39
C1_4_1	400	7138.8	3600.0	7138.8	0.00	7138.8	0.00	7138.8	0.00	7138.8	0.00
C1_4_5	400	7138.8	3600.0	7138.8	0.00	7138.8	0.00	7138.8	0.00	7138.8	0.00
C1_4_10	400	6825.4	3600.0	6825.4	0.00	6825.4	0.00	6825.7	0.00	6826.5	0.02
C2_4_1	400	4100.3	3600.0	4100.3	0.00	4100.3	0.00	4100.3	0.00	4100.3	0.00
C2_4_5	400	3923.2	3600.0	3923.2	0.00	3923.2	0.00	3927.5	0.11	3923.2	0.00
C2_4_10	400	3665.1	3600.0	3665.1	0.00	3665.1	0.00	3665.1	0.00	3665.1	0.00

Table B.12 – Comparative results on the 90 *GH* benchmark instances for DIMACS, HGS-DIMACS, MA-NTGA, and MA-ETGA (continued).

Instance	N_C	BKS	t	DIMACS		HGS-DIMACS		MA-NTGA		MA-ETGA	
				f^*	Gap(%)	f^*	Gap(%)	f^*	Gap(%)	f^*	Gap(%)
R1_4_1	400	10305.8	3600.0	10305.8	0.00	10305.8	0.00	10313.8	0.08	10337.6	0.31
R1_4_5	400	9184.6	3600.0	9188.6	0.04	9190.3	0.06	9208.7	0.26	9198.4	0.15
R1_4_10	400	8077.8	3600.0	8086.4	0.11	8081.9	0.05	8109.5	0.39	8138.7	0.75
R2_4_1	400	7520.7	3600.0	7520.7	0.00	7520.7	0.00	7559.7	0.52	7526.8	0.08
R2_4_5	400	6567.9	3600.0	6567.9	0.00	6567.9	0.00	6568.1	0.00	6577.1	0.14
R2_4_10	400	5638.1	3600.0	5645.9	0.14	5657.7	0.35	5690.2	0.92	5694.0	0.99
RC1_4_1	400	8522.9	3600.0	8524.0	0.01	8524.0	0.01	8524.0	0.01	8527.2	0.05
RC1_4_5	400	8152.3	3600.0	8172.4	0.25	8154.6	0.03	8196.4	0.54	8206.7	0.67
RC1_4_10	400	7581.2	3600.0	7585.3	0.05	7589.0	0.10	7634.5	0.70	7634.0	0.70
RC2_4_1	400	6147.3	3600.0	6147.3	0.00	6147.3	0.00	6156.2	0.14	6148.7	0.02
RC2_4_5	400	5392.3	3600.0	5396.9	0.09	5392.3	0.00	5400.1	0.14	5394.7	0.04
RC2_4_10	400	4252.3	3600.0	4252.3	0.00	4252.3	0.00	4266.6	0.34	4261.6	0.22
C1_6_1	600	14076.6	3600.0	14076.6	0.00	14076.6	0.00	14076.6	0.00	14076.6	0.00
C1_6_5	600	14066.8	3600.0	14066.8	0.00	14066.8	0.00	14066.8	0.00	14066.8	0.00
C1_6_10	600	13617.5	3600.0	13636.5	0.14	13636.6	0.14	13626.3	0.06	13640.1	0.17
C2_6_1	600	7752.2	3600.0	7752.2	0.00	7752.2	0.00	7752.2	0.00	7752.2	0.00
C2_6_5	600	7553.8	3600.0	7553.8	0.00	7553.8	0.00	7553.8	0.00	7553.8	0.00
C2_6_10	600	7123.9	3600.0	7123.9	0.00	7128.0	0.06	7131.5	0.11	7136.0	0.17
R1_6_1	600	21274.2	3600.0	21274.2	0.00	21312.5	0.18	21314.5	0.19	21315.5	0.19
R1_6_5	600	19294.9	3600.0	19305.7	0.06	19322.6	0.14	19435.8	0.73	19463.8	0.88
R1_6_10	600	17583.7	3600.0	17648.1	0.37	17603.2	0.11	17711.0	0.72	17795.1	1.20
R2_6_1	600	15145.3	3600.0	15145.3	0.00	15145.3	0.00	15274.1	0.85	15194.9	0.33
R2_6_5	600	13790.2	3600.0	13809.7	0.14	13803.7	0.10	13895.0	0.76	13926.5	0.99
R2_6_10	600	11837.0	3600.0	11851.6	0.12	11852.9	0.13	12010.4	1.46	11988.6	1.28
RC1_6_1	600	16944.2	3600.0	16981.3	0.22	16961.1	0.10	17037.9	0.55	17069.3	0.74
RC1_6_5	600	16536.3	3600.0	16580.4	0.27	16597.4	0.37	16682.9	0.89	16696.2	0.97
RC1_6_10	600	15651.3	3600.0	15672.8	0.14	15737.1	0.55	15870.0	1.40	15915.0	1.68
RC2_6_1	600	11966.1	3600.0	11979.3	0.11	11966.1	0.00	12005.7	0.33	11986.7	0.17
RC2_6_5	600	11080.7	3600.0	11082.5	0.02	11080.7	0.00	11111.9	0.28	11109.5	0.26
RC2_6_10	600	8973.3	3600.0	9014.8	0.46	8976.1	0.03	9028.3	0.61	9036.1	0.70
C1_8_1	800	25156.9	3600.0	25156.9	0.00	25156.9	0.00	25156.9	0.00	25156.9	0.00
C1_8_5	800	25138.6	3600.0	25138.6	0.00	25138.6	0.00	25138.6	0.00	25138.6	0.00
C1_8_10	800	24026.7	3600.0	24030.4	0.02	24046.3	0.08	24094.3	0.28	24091.5	0.27
C2_8_1	800	11631.9	3600.0	11631.9	0.00	11632.7	0.01	11631.9	0.00	11631.9	0.00
C2_8_5	800	11395.6	3600.0	11395.6	0.00	11395.6	0.00	11395.6	0.00	11395.6	0.00
C2_8_10	800	10946.0	3600.0	10946.0	0.00	10946.7	0.01	10950.2	0.04	10957.6	0.11
R1_8_1	800	36345.0	3600.0	36393.3	0.13	36465.0	0.33	36641.8	0.82	36704.2	0.99
R1_8_5	800	33494.0	3600.0	33640.0	0.44	33658.8	0.49	33815.4	0.96	33849.3	1.06
R1_8_10	800	30918.3	3600.0	31059.8	0.46	31138.4	0.71	31436.6	1.68	31573.6	2.12
R2_8_1	800	24963.8	3600.0	24986.8	0.09	24998.9	0.14	25237.3	1.10	25129.0	0.66
R2_8_5	800	22795.6	3600.0	22802.0	0.03	22808.2	0.06	23026.9	1.01	22929.3	0.59
R2_8_10	800	19964.2	3600.0	20016.5	0.26	20007.8	0.22	20154.0	0.95	20230.8	1.34
RC1_8_1	800	29952.8	3600.0	30039.5	0.29	30058.9	0.35	30186.5	0.78	30193.5	0.80
RC1_8_5	800	29219.9	3600.0	29306.4	0.30	29327.6	0.37	29514.3	1.01	29556.2	1.15
RC1_8_10	800	28168.5	3600.0	28210.7	0.15	28283.1	0.41	28564.8	1.41	28786.5	2.19
RC2_8_1	800	19201.3	3600.0	19213.2	0.06	19213.4	0.06	19279.2	0.41	19239.5	0.20
RC2_8_5	800	17466.1	3600.0	17493.3	0.16	17466.1	0.00	17540.8	0.43	17595.1	0.74
RC2_8_10	800	14370.9	3600.0	14445.3	0.52	14376.2	0.04	14470.6	0.69	14478.0	0.75
C1_10_1	1000	42444.8	7200.0	42444.8	0.00	42444.8	0.00	42444.8	0.00	42444.8	0.00
C1_10_5	1000	42434.8	7200.0	42434.8	0.00	42434.8	0.00	42434.8	0.00	42435.5	0.00
C1_10_10	1000	39816.8	7200.0	39874.5	0.14	40149.5	0.84	40030.1	0.54	40056.8	0.60
C2_10_1	1000	16841.1	7200.0	16841.1	0.00	16841.1	0.00	16841.1	0.00	16841.1	0.00
C2_10_5	1000	16521.3	7200.0	16521.3	0.00	16521.3	0.00	16521.9	0.00	16522.9	0.01
C2_10_10	1000	15728.6	7200.0	15728.6	0.00	15728.6	0.00	15773.0	0.28	15750.8	0.14
R1_10_1	1000	53026.1	7200.0	53233.9	0.39	53341.7	0.60	53488.9	0.87	53522.0	0.94
R1_10_5	1000	50406.7	7200.0	50487.8	0.16	50596.5	0.38	50971.1	1.12	50895.6	0.97
R1_10_10	1000	47364.6	7200.0	47570.6	0.43	47690.5	0.69	48015.0	1.37	48454.5	2.30
R2_10_1	1000	36881.0	7200.0	36899.4	0.05	36891.3	0.03	37323.2	1.20	37079.2	0.54
R2_10_5	1000	34132.8	7200.0	34233.0	0.29	34195.3	0.18	34497.3	1.07	34422.9	0.85
R2_10_10	1000	29840.5	7200.0	29956.9	0.39	29882.3	0.14	30118.6	0.93	30109.1	0.90
RC1_10_1	1000	45790.7	7200.0	45948.5	0.34	46047.3	0.56	46285.6	1.08	46362.2	1.25
RC1_10_5	1000	45028.1	7200.0	45235.5	0.46	45349.6	0.71	45593.0	1.25	45795.4	1.70
RC1_10_10	1000	43533.7	7200.0	43745.6	0.49	43635.1	0.23	44251.9	1.65	44627.1	2.51
RC2_10_1	1000	28122.6	7200.0	28190.8	0.24	28192.1	0.25	28324.6	0.72	28275.9	0.55
RC2_10_5	1000	25797.5	7200.0	25892.3	0.37	25862.2	0.25	25983.3	0.72	25961.7	0.64
RC2_10_10	1000	21731.2	7200.0	21848.3	0.54	21862.3	0.60	21912.3	0.83	21885.4	0.71
MEAN	-	16297.8	3960.0	16328.2	0.11	16337.2	0.13	16410.3	0.45	16423.1	0.48

Table B.13 – Comparative results on the 20 *JD* benchmark instances for MA-FIRD, MA-NTGA, and MA-ETGA.

Instance	N_C	BKS	MA-FIRD			MA-NTGA			MA-ETGA		
			f^*	t	Gap(%)	f^*	t	Gap(%)	f^*	t	Gap(%)
F201	200	64083.4	64083.4	1360.4	0.00	64297.8	424.0	0.33	64483.1	516.1	0.62
F202	200	64378.1	64378.1	1236.1	0.00	64758.0	415.1	0.59	64698.3	608.1	0.50
F203	200	65547.4	65547.4	1349.0	0.00	65526.9	495.1	-0.03	65664.1	614.9	0.18
F204	200	64037.3	64037.3	1429.9	0.00	64179.1	532.0	0.22	64310.7	641.4	0.43
F401	400	115876.0	115876.0	7005.8	0.00	116999.0	841.8	0.97	117057.0	1056.2	1.02
F402	400	121997.0	121997.0	7082.8	0.00	122429.0	865.5	0.35	122369.0	1567.1	0.30
F403	400	115540.0	115540.0	6729.3	0.00	115920.0	874.3	0.33	116693.0	1109.8	1.00
F404	400	119451.0	119451.0	7129.2	0.00	119242.0	882.7	-0.17	119777.0	1456.1	0.27
F601	600	174108.0	174108.0	7203.1	0.00	173141.0	1731.3	-0.56	173896.0	1571.3	-0.12
F602	600	178317.0	178317.0	7202.7	0.00	177893.0	1503.2	-0.24	179052.0	1836.8	0.41
F603	600	177979.0	177979.0	7202.7	0.00	177320.0	1378.0	-0.37	177997.0	1567.1	0.01
F604	600	178201.0	178201.0	7202.8	0.00	176527.0	1651.1	-0.94	177806.0	1712.8	-0.22
F801	800	208325.0	208325.0	7208.8	0.00	205657.0	2362.7	-1.28	207300.0	2589.9	-0.49
F802	800	206214.0	206214.0	7207.2	0.00	204164.0	2302.7	-0.99	205251.0	2696.9	-0.47
F803	800	210213.0	210213.0	7207.6	0.00	207664.0	2167.3	-1.21	207585.0	2737.8	-1.25
F804	800	203971.0	203971.0	7206.4	0.00	202271.0	2165.4	-0.83	203116.0	2546.4	-0.42
F1001	1000	303113.0	303113.0	7218.1	0.00	297090.0	4617.2	-1.99	297325.0	3434.4	-1.91
F1002	1000	299833.0	299833.0	7216.6	0.00	292779.0	4010.4	-2.35	294712.0	3511.3	-1.71
F1003	1000	301051.0	301051.0	7211.7	0.00	292581.0	4556.9	-2.81	294946.0	3187.9	-2.03
F1004	1000	298555.0	298555.0	7219.0	0.00	292222.0	4875.1	-2.12	293905.0	3223.3	-1.56
MEAN	-	173539.5	173539.5	5991.5	0.00	171633.0	1932.6	-0.66	172397.2	1909.3	-0.27

Table C.14 – Comparative results on the 33 *C97* MDVRP benchmark instances between BKS, CGL, PR, VCGLR, ELTG, SCA and MDFIHA.

Instance	N_C	N_D	N_V	BKS	CGL			PR			VCGLR			ELTG			SCA			MDFIHA (Ours)			
					f_{Best}	t	Gap (%)	f_{Best}	$\gamma \cdot t$	Gap (%)	f_{Best}	$\gamma \cdot t$	Gap (%)	f_{Best}	$\gamma \cdot t$	Gap (%)	f_{Best}	$\gamma \cdot t$	Gap (%)	f_{Best}	f_{Avg}	$\gamma \cdot t$	Gap (%)
p01	50	4	4	576.87*	**576.87**	194.00	**0.00**	**576.87**	7.83	**0.00**	**576.87**	3.78	**0.00**	**576.87**	2.17	**0.00**	**576.87**	33.18	**0.00**	**576.87**	576.87	7.97	**0.00**
p02	50	4	2	473.53*	473.87	208.00	0.07	**473.53**	7.56	**0.00**	**473.53**	3.51	**0.00**	**473.53**	1.86	**0.00**	**473.53**	21.03	**0.00**	**473.53**	473.53	18.43	**0.00**
p03	75	5	3	641.19*	645.15	340.00	0.62	**641.19**	17.28	**0.00**	**641.19**	7.02	**0.00**	**641.19**	8.99	**0.00**	**641.19**	78.69	**0.00**	**641.19**	641.60	14.01	**0.00**
p04	100	2	8	1001.04*	1006.66	467.00	0.56	**1001.04**	23.76	**0.00**	**1001.04**	31.32	**0.00**	**1001.04**	27.90	**0.00**	**1001.04**	85.59	**0.00**	**1001.04**	1004.04	70.91	**0.00**
p05	100	2	5	750.03*	753.34	493.00	0.44	751.26	32.40	0.16	**750.03**	17.28	**0.00**	**750.03**	8.06	**0.00**	**750.03**	126.86	**0.00**	**750.03**	750.14	45.39	**0.00**
p06	100	3	6	876.50*	877.84	459.00	0.15	876.70	25.11	0.02	**876.50**	18.36	**0.00**	**876.50**	31.93	**0.00**	**876.50**	175.74	**0.00**	**876.50**	878.70	29.14	**0.00**
p07	100	4	4	881.97*	891.95	463.00	1.13	**881.97**	23.76	**0.00**	**881.97**	25.11	**0.00**	884.66	32.86	0.30	**881.97**	208.58	**0.00**	**881.97**	886.33	80.74	**0.00**
p08	249	2	14	4369.95	4482.44	1526.00	2.57	4390.80	89.91	0.48	4372.78	162.00	0.06	4371.66	88.35	0.04	**4369.95**	198.87	**0.00**	4372.78	4384.34	278.04	0.06
p09	249	3	12	3858.66	3920.85	1604.00	1.61	3873.64	97.47	0.39	**3858.66**	153.90	**0.00**	3880.85	79.36	0.58	3881.31	338.69	0.59	**3858.66**	3868.83	470.09	**0.00**
p10	249	4	8	3629.60	3714.65	1530.00	2.34	3650.04	98.01	0.56	3631.11	159.03	0.04	**3629.60**	82.77	**0.00**	3644.51	361.99	0.41	3631.11	3640.81	256.83	0.04
p11	249	5	6	3545.18	3580.84	1555.00	1.01	3546.06	96.39	0.02	3546.06	115.56	0.02	**3545.18**	59.52	**0.00**	3563.28	289.53	0.51	3546.06	3548.75	276.50	0.02
p12	80	2	5	1318.95*	**1318.95**	334.00	**0.00**	**1318.95**	20.25	**0.00**	**1318.95**	8.37	**0.00**	**1318.95**	1.86	**0.00**	**1318.95**	74.90	**0.00**	**1318.95**	1318.95	21.40	**0.00**
p13	80	2	5	1318.95*	**1318.95**	335.00	**0.00**	**1318.95**	16.20	**0.00**	**1318.95**	9.18	**0.00**	**1318.95**	2.17	**0.00**	**1318.95**	4.41	**0.00**	**1318.95**	1318.95	23.90	**0.00**
p14	80	2	5	1360.12*	**1360.12**	326.00	**0.00**	**1360.12**	15.66	**0.00**	**1360.12**	8.91	**0.00**	**1360.12**	1.86	**0.00**	**1360.12**	3.16	**0.00**	**1360.12**	1360.68	33.01	**0.00**
p15	160	4	5	2505.42*	2534.13	844.00	1.15	**2505.42**	68.31	**0.00**	**2505.42**	31.05	**0.00**	**2505.42**	35.34	**0.00**	**2505.42**	498.14	**0.00**	**2505.42**	2508.05	78.59	**0.00**
p16	160	4	5	2572.23*	**2572.23**	843.00	**0.00**	**2572.23**	50.76	**0.00**	**2572.23**	31.86	**0.00**	**2572.23**	36.58	**0.00**	**2572.23**	20.05	**0.00**	**2572.23**	2572.23	86.40	**0.00**
p17	160	4	5	2709.09*	2720.23	822.00	0.41	**2709.09**	48.33	**0.00**	**2709.09**	34.56	**0.00**	**2709.09**	33.48	**0.00**	2731.37	11.02	0.82	**2709.09**	2709.09	115.51	**0.00**
p18	240	6	5	3702.85*	3710.49	1491.00	0.21	**3702.85**	113.13	**0.00**	**3702.85**	73.17	**0.00**	**3702.85**	86.18	**0.00**	3728.49	433.42	0.69	**3702.85**	3705.41	226.93	**0.00**
p19	240	6	5	3827.06*	**3827.06**	1512.00	**0.00**	**3827.06**	85.05	**0.00**	**3827.06**	68.04	**0.00**	**3827.06**	79.36	**0.00**	**3827.06**	25.66	**0.00**	**3827.06**	3827.06	161.36	**0.00**
p20	240	6	5	4058.07*	**4058.07**	1483.00	**0.00**	**4058.07**	81.00	**0.00**	**4058.07**	70.74	**0.00**	**4058.07**	82.77	**0.00**	4097.06	16.01	0.96	**4058.07**	4069.83	213.20	**0.00**
p21	360	9	5	5474.84	5535.99	2890.00	1.12	**5474.84**	157.14	**0.00**	**5474.84**	162.00	**0.00**	**5474.84**	83.08	**0.00**	5506.26	1479.96	0.57	**5474.84**	5481.71	565.48	**0.00**
p22	360	9	5	5702.16	5716.01	2934.00	0.24	**5702.16**	124.74	**0.00**	**5702.16**	162.00	**0.00**	**5702.16**	81.22	**0.00**	**5702.16**	51.34	**0.00**	**5702.16**	5702.16	289.67	**0.00**
p23	360	9	5	6078.75	6139.73	2872.00	1.00	**6078.75**	119.61	**0.00**	**6078.75**	162.00	**0.00**	6095.46	88.35	0.27	6145.58	30.54	1.10	**6078.75**	6114.51	666.65	**0.00**
pr01	48	4	2	861.32	**861.32**	242.00	**0.00**	**861.32**	8.10	**0.00**	**861.32**	2.70	**0.00**	**861.32**	1.24	**0.00**	**861.32**	8.24	**0.00**	**861.32**	861.32	3.36	**0.00**
pr02	96	4	4	1296.25	1314.99	505.00	1.45	1307.34	27.81	0.86	1307.34	12.42	0.86	1311.11	3.41	1.15	**1296.25**	132.71	**0.00**	**1296.25**	1297.22	32.28	**0.00**
pr03	144	4	6	1803.80	1815.62	854.00	0.66	1806.60	57.78	0.16	**1803.80**	31.05	**0.00**	**1803.80**	36.58	**0.00**	**1803.80**	100.43	**0.00**	**1803.80**	1803.88	116.61	**0.00**
pr04	192	4	8	2058.31	2094.24	1158.00	1.75	2060.93	79.92	0.13	2058.31	84.51	0.00	2064.11	38.44	0.28	2060.74	605.31	0.12	**2042.45**	2054.55	246.49	**-0.77**
pr05	240	4	10	2331.20	2408.10	1529.00	3.30	2337.84	100.44	0.28	2331.20	154.98	0.00	2349.63	66.03	0.79	2346.72	1046.48	0.67	**2327.59**	2329.24	679.93	**-0.15**
pr06	288	4	12	2674.07	2768.13	2007.00	3.52	2687.60	125.55	0.51	2676.30	162.00	0.08	2710.30	72.54	1.35	2674.07	1275.84	0.00	**2663.56**	2666.02	874.44	**-0.39**
pr07	72	6	3	1082.93	1092.12	412.00	0.85	1089.56	15.66	0.61	1089.56	5.40	0.61	1089.56	3.41	0.61	1082.93	14.43	0.00	**1075.12**	1076.60	6.77	**-0.72**
pr08	144	6	6	1664.60	1676.26	906.00	0.70	1664.85	55.89	0.02	1664.85	33.21	0.02	1665.50	20.46	0.05	1664.60	339.80	0.00	**1658.23**	1662.12	233.30	**-0.38**
pr09	216	6	9	2133.20	2176.79	1462.00	2.04	2136.42	94.50	0.15	2133.20	98.82	0.00	2151.45	48.36	0.86	2133.20	250.60	0.00	**2131.70**	2139.99	212.73	**-0.07**
pr10	288	6	12	2820.88	3089.62	2105.00	9.53	2889.82	122.85	2.44	2868.20	162.00	1.68	2910.78	93.62	3.19	2820.88	1342.67	0.00	**2805.53**	2808.77	1141.00	**-0.54**
Mean	-	-	-	2423.02	2455.56	1112.27	1.16	2428.30	63.88	0.21	2425.22	68.66	0.10	2430.12	43.03	0.29	2430.86	293.45	0.20	2421.33	2425.52	229.61	-0.09

Table C.15 – Comparative results on the 10 *C97-T* MDVRP benchmark instances between Optima, SCA and MDFIHA.

Instance	N_C	N_D	N_V	Optima	SCA			MDFIHA (Ours)			
					f_{Best}	$\gamma \cdot t$	Gap (%)	f_{Best}	f_{Avg}	$\gamma \cdot t$	Gap (%)
pr01	48	4	1	861.32	**861.32**	19.58	**0.00**	**861.32**	861.32	4.12	**0.00**
pr02	96	4	2	1307.34	**1307.34**	140.68	**0.00**	**1307.34**	1308.00	28.52	**0.00**
pr03	144	4	3	1803.80	**1803.80**	107.60	**0.00**	**1803.80**	1804.13	95.96	**0.00**
pr04	192	4	4	2058.31	2060.93	609.62	0.13	**2058.31**	2066.65	235.54	**0.00**
pr05	240	4	5	2331.20	2346.05	1038.39	0.64	**2331.20**	2339.81	797.08	**0.00**
pr06	288	4	6	2676.30	**2676.30**	1288.80	**0.00**	**2676.30**	2680.60	898.26	**0.00**
pr07	72	6	1	1089.56	**1089.56**	10.00	**0.00**	**1089.56**	1089.56	11.35	**0.00**
pr08	144	6	2	1664.85	**1664.85**	339.67	**0.00**	**1664.85**	1665.52	145.92	**0.00**
pr09	216	6	3	2133.20	**2133.20**	240.21	**0.00**	**2133.20**	2140.07	287.91	**0.00**
pr10	288	6	4	2867.26	2868.20	1344.02	0.03	**2867.26**	2875.50	1317.23	**0.00**
Mean	-	-	-	1879.31	1881.16	513.86	0.08	1879.31	1883.12	382.19	0.00

Table C.16 – Comparative results on the 20 *C01* MDVRPTW benchmark instances between BKS, VCGP, SCA and MDFIHA.

Instance	N_C	N_D	N_V	BKS	VCGP			SCA			MDFIHA (Ours)			
					f_{Best}	$\gamma \cdot t$	Gap (%)	f_{Best}	$\gamma \cdot t$	Gap (%)	f_{Best}	f_{Avg}	$\gamma \cdot t$	Gap (%)
pr01	48	4	2	1074.12	**1074.12**	13.39	**0.00**	**1074.12**	67.74	**0.00**	**1074.12**	1074.12	14.09	**0.00**
pr02	96	4	3	1762.21	**1762.21**	49.68	**0.00**	**1762.21**	337.46	**0.00**	**1762.21**	1762.29	91.89	**0.00**
pr03	144	4	4	2373.65	**2373.65**	75.60	**0.00**	**2373.65**	520.38	**0.00**	**2373.65**	2380.24	166.76	**0.00**
pr04	192	4	5	2814.34	2815.75	254.45	0.05	**2814.34**	3454.26	**0.00**	**2814.34**	2817.89	669.76	**0.00**
pr05	240	4	6	2964.65	2964.65	374.98	0.00	2965.18	915.85	0.02	**2962.25**	2967.52	1494.27	**-0.08**
pr06	288	4	7	3588.78	**3588.78**	580.18	**0.00**	3590.58	4026.62	0.05	**3588.78**	3594.42	1516.47	**0.00**
pr07	72	6	2	1418.22	**1418.22**	22.03	**0.00**	**1418.22**	146.58	**0.00**	**1418.22**	1418.93	56.07	**0.00**
pr08	144	6	3	2096.73	**2096.73**	103.25	**0.00**	**2096.73**	501.21	**0.00**	**2096.73**	2100.42	239.77	**0.00**
pr09	216	6	4	2712.56	**2712.56**	224.64	**0.00**	2717.69	1539.15	0.19	**2712.56**	2713.54	437.81	**0.00**
pr10	288	6	5	3465.92	3465.92	657.50	0.00	3469.29	4658.51	0.10	**3465.54**	3477.00	1334.49	**-0.01**
pr11	48	4	1	1005.73	**1005.73**	22.03	**0.00**	**1005.73**	13.82	**0.00**	**1005.73**	1005.73	34.08	**0.00**
pr12	96	4	2	1464.50	**1464.50**	72.58	**0.00**	**1464.50**	480.07	**0.00**	**1464.50**	1465.29	104.60	**0.00**
pr13	144	4	3	2001.81	**2001.81**	127.01	**0.00**	**2001.81**	314.08	**0.00**	**2001.81**	2001.82	170.77	**0.00**
pr14	192	4	4	2195.33	**2195.33**	282.96	**0.00**	**2195.33**	1924.17	**0.00**	**2195.33**	2197.15	248.47	**0.00**
pr15	240	4	5	2433.15	**2433.15**	542.59	**0.00**	2434.94	1583.78	0.07	**2433.15**	2449.23	1685.11	**0.00**
pr16	288	4	6	2836.67	**2836.67**	689.90	**0.00**	2850.69	2521.00	0.49	2837.85	2842.35	2877.91	0.04
pr17	72	6	1	1236.24	**1236.24**	45.36	**0.00**	**1236.24**	79.47	**0.00**	**1236.24**	1236.62	68.23	**0.00**
pr18	144	6	2	1788.18	**1788.18**	142.56	**0.00**	**1788.18**	378.07	**0.00**	**1788.18**	1788.71	197.17	**0.00**
pr19	216	6	3	2261.08	2261.08	371.09	0.00	2263.74	1304.18	0.12	**2257.13**	2266.69	552.22	**-0.17**
pr20	288	6	4	2993.31	2993.31	958.18	0.00	2995.08	2850.71	0.06	**2983.78**	2996.78	2694.24	**-0.32**
Mean	-	-	-	2224.42	2224.43	280.50	0.00	2225.91	1380.86	0.05	2223.61	2227.84	732.71	-0.03

Table C.17 – Comparative results on the 10 *C01-R* MDVRPTW benchmark instances between BKS, VCGP and MDFIHA.

Instance	N_C	N_D	N_V	BKS	VCGP			MDFIHA (Ours)			
					f_{Best}	$\gamma \cdot t$	Gap (%)	f_{Best}	f_{Avg}	$\gamma \cdot t$	Gap (%)
pr11	48	4	2	1005.73	1005.73	22.03	0.00	**922.44**	922.44	9.05	**-8.28**
pr12	96	4	3	1464.50	1464.50	72.58	0.00	**1430.29**	1430.29	55.31	**-2.34**
pr13	144	4	4	2001.81	**2001.81**	127.01	**0.00**	**2001.81**	2002.33	175.36	**0.00**
pr14	192	4	5	2195.33	2195.33	282.96	0.00	**2191.06**	2192.77	285.07	**-0.19**
pr15	240	4	6	2433.15	2433.15	542.59	0.00	**2419.88**	2425.73	900.55	**-0.55**
pr16	288	4	7	2836.67	2836.67	689.90	0.00	**2823.68**	2833.65	1432.80	**-0.46**
pr17	72	6	2	1236.24	1236.24	45.36	0.00	**1167.20**	1167.20	32.11	**-5.58**
pr18	144	6	3	1788.18	1788.18	142.56	0.00	**1775.25**	1780.84	242.02	**-0.72**
pr19	216	6	4	2261.08	2261.08	371.09	0.00	**2256.11**	2266.39	1103.20	**-0.22**
pr20	288	6	5	2993.31	2993.31	958.18	0.00	**2900.86**	2903.34	1307.47	**-3.09**
Mean	-	-	-	2021.60	2021.60	325.43	0.00	1988.86	1992.50	554.29	-2.14

Table C.18 – Comparative results on the 28 *V13* MDVRPTW benchmark instances between VCGP, MDFIHA and MDFIHA-ETGA.

Instance	N_C	N_D	N_V	VCGP			MDFIHA (Ours)				MDFIHA-ETGA (Ours)			
				f_{Best}	f_{Avg}	$\gamma \cdot t$	f_{Best}	f_{Avg}	$\gamma \cdot t$	Gap (%)	f_{Best}	f_{Avg}	$\gamma \cdot t$	Gap (%)
pr11a	360	4	10	6720.71	6772.00	726.19	**6685.36**	6702.83	2868.55	**-0.53**	**6683.00**	6697.60	5167.52	**-0.56**
pr12a	480	4	13	8179.80	8259.57	1296.00	8208.50	8225.16	7099.69	0.35	**8164.72**	8191.44	6226.49	**-0.18**
pr13a	600	4	16	9667.20	9751.22	2369.52	**9602.32**	9662.21	7201.87	**-0.67**	**9571.71**	9616.67	7015.26	**-0.99**
pr14a	720	4	19	11124.01	11235.13	2836.08	**11089.24**	11117.63	7202.28	**-0.31**	**11062.70**	11110.91	6919.64	**-0.55**
pr15a	840	4	22	13013.97	13078.26	5721.41	13019.83	13080.84	7205.23	0.05	**12895.63**	12941.05	7201.95	**-0.91**
pr16a	960	4	26	14299.87	14415.89	5772.82	14450.22	14489.24	7208.66	1.05	**14223.38**	14319.36	7089.71	**-0.53**
pr17a	360	6	7	6304.30	6340.66	744.34	6309.20	6321.09	3615.48	0.08	6314.89	6335.83	4280.81	0.17
pr18a	520	6	10	8308.32	8381.71	1911.60	**8228.31**	8247.81	7202.66	**-0.96**	**8238.25**	8267.51	6809.99	**-0.84**
pr19a	700	6	13	10677.61	10734.60	3214.94	10679.74	10728.21	7203.97	0.02	**10663.75**	10675.57	6429.51	**-0.13**
pr20a	880	6	16	11963.91	12142.60	4638.38	**11929.23**	11983.04	7208.83	**-0.29**	**11839.92**	11864.35	7201.67	**-1.04**
pr21a	420	12	4	6260.53	6321.20	1209.60	6261.83	6275.48	4780.65	0.02	**6258.84**	6271.13	5903.00	**-0.03**
pr22a	600	12	6	7985.37	8047.87	3285.36	**7919.77**	7955.53	7203.30	**-0.82**	**7935.78**	7965.98	6742.34	**-0.62**
pr23a	780	12	8	9937.43	9984.75	5949.50	**9893.39**	9960.04	7204.03	**-0.44**	**9870.23**	9891.89	7203.07	**-0.68**
pr24a	960	12	10	11923.72	11971.74	8517.74	12031.09	12063.55	7210.03	0.90	**11861.22**	11882.24	7204.22	**-0.52**
pr11b	360	4	8	4839.44	4852.67	779.33	**4832.98**	4842.14	3257.81	**-0.13**	**4831.74**	4842.88	5057.87	**-0.16**
pr12b	480	4	11	6063.26	6084.33	1256.69	**6038.38**	6053.57	7018.54	**-0.41**	**6033.93**	6063.03	6157.79	**-0.48**
pr13b	600	4	14	7254.17	7282.25	3066.77	**7245.29**	7268.61	7204.84	**-0.12**	**7231.95**	7274.66	6866.29	**-0.31**
pr14b	720	4	17	8732.29	8796.77	4273.34	8772.44	8809.61	7203.77	0.46	8761.07	8783.31	7200.69	0.33
pr15b	840	4	20	10439.72	10496.39	5593.54	**10425.82**	10488.71	7204.47	**-0.13**	**10417.15**	10441.08	6845.19	**-0.22**
pr16b	960	4	23	11483.22	11565.39	7357.39	11568.44	11595.45	7210.55	0.74	**11394.50**	11457.72	7204.34	**-0.77**
pr17b	360	6	6	4806.01	4847.58	681.70	**4790.27**	4809.84	2545.24	**-0.33**	**4797.43**	4811.54	4601.24	**-0.18**
pr18b	520	6	9	6526.72	6555.95	1704.24	**6467.78**	6498.29	6939.66	**-0.90**	**6484.49**	6505.49	5536.92	**-0.65**
pr19b	700	6	12	8227.25	8295.30	3479.76	**8183.28**	8214.23	7202.20	**-0.53**	**8164.09**	8176.36	6999.77	**-0.77**
pr20b	880	6	15	10325.80	10378.54	6511.97	**10274.63**	10305.06	7206.93	**-0.50**	**10162.84**	10196.89	7202.17	**-1.58**
pr21b	420	12	4	4866.57	4887.57	1587.60	**4830.90**	4882.99	3005.66	**-0.73**	**4834.49**	4847.04	5767.18	**-0.66**
pr22b	600	12	6	6488.50	6537.15	3165.70	**6474.00**	6480.90	7039.33	**-0.22**	**6446.58**	6479.22	6494.99	**-0.65**
pr23b	780	12	7	8523.41	8603.85	7084.37	**8491.96**	8516.51	7203.59	**-0.37**	**8417.24**	8453.39	7201.55	**-1.25**
pr24b	960	12	8	10890.08	10997.66	12895.63	**10721.40**	10806.23	7203.38	**-1.55**	**10665.16**	10667.91	7201.68	**-2.07**
Mean	-	-	-	8779.76	8843.52	3843.98	8765.20	8799.46	6352.19	-0.22	8722.38	8751.15	6490.46	-0.60

Table C.19 – Comparative results on the 24 *L14* MDOVRP benchmark instances between BKS, JB, RM, SCA and MDFIHA.

Instance	N_C	N_D	BKS	JB			RM			SCA			MDFIHA (Ours)			
				f_{Best}	$\gamma \cdot t$	Gap (%)	f_{Best}	$\gamma \cdot t$	Gap (%)	f_{Best}	$\gamma \cdot t$	Gap (%)	f_{Best}	f_{Avg}	$\gamma \cdot t$	Gap (%)
p01	50	4	386.18*	**386.18**	6.84	**0.00**	**386.18**	0.92	**0.00**	**386.18**	22.01	**0.00**	**386.18**	386.91	3.83	**0.00**
p02	50	4	375.93*	376.44	7.83	0.14	**375.93**	0.20	**0.00**	**375.93**	29.15	**0.00**	**375.93**	375.93	2.60	**0.00**
p03	75	5	474.57*	475.79	14.58	0.26	**474.57**	1.39	**0.00**	**474.57**	53.74	**0.00**	**474.57**	474.57	13.65	**0.00**
p04	100	2	662.22	**662.22**	27.54	**0.00**	**662.22**	7272.00	**0.00**	**662.22**	21.26	**0.00**	**662.22**	662.22	19.46	**0.00**
p05	100	2	607.53*	609.04	19.98	0.25	**607.53**	73.68	**0.00**	**607.53**	41.67	**0.00**	**607.53**	608.39	21.90	**0.00**
p06	100	3	611.99*	**611.99**	19.80	**0.00**	**611.99**	416.08	**0.00**	**611.99**	31.48	**0.00**	**611.99**	611.99	17.19	**0.00**
p07	100	4	608.28*	609.60	27.54	0.22	**608.28**	1041.73	**0.00**	**608.28**	38.68	**0.00**	**608.28**	608.77	29.11	**0.00**
p08	249	2	2776.12	2794.10	133.29	0.65	2870.21	7272.00	3.39	**2776.12**	24.25	**0.00**	2776.35	2781.67	320.77	0.01
p09	249	3	2578.49	2582.65	136.44	0.16	2660.45	7272.00	3.18	2587.04	132.65	0.33	**2576.17**	2581.95	212.03	**-0.09**
p10	249	4	2482.32	2500.43	103.77	0.73	2528.98	7272.00	1.88	2482.32	253.03	0.00	**2476.12**	2479.58	167.36	**-0.25**
p11	249	5	2468.45	2478.31	119.97	0.40	2499.25	7272.00	1.25	2472.85	338.26	0.18	**2454.01**	2458.01	166.29	**-0.58**
p12	80	2	953.26*	**953.26**	9.36	**0.00**	**953.26**	0.46	**0.00**	**953.26**	56.77	**0.00**	**953.26**	953.26	6.41	**0.00**
p15	160	4	1885.81*	**1885.81**	32.94	**0.00**	**1885.81**	3.67	**0.00**	**1885.81**	213.04	**0.00**	**1885.81**	1885.81	25.28	**0.00**
p18	240	6	2818.36*	**2818.36**	68.40	**0.00**	**2818.36**	14.83	**0.00**	**2818.36**	482.29	**0.00**	**2818.36**	2818.36	61.62	**0.00**
pr01	48	4	647.03*	**647.03**	4.50	**0.00**	**647.03**	0.14	**0.00**	**647.03**	57.36	**0.00**	**647.03**	647.03	2.48	**0.00**
pr02	96	4	979.82*	**979.82**	17.19	**0.00**	**979.82**	1.99	**0.00**	**979.82**	114.72	**0.00**	**979.82**	980.33	9.66	**0.00**
pr03	144	4	1423.48*	1429.38	37.98	0.41	**1423.48**	17.01	**0.00**	**1423.48**	203.78	**0.00**	**1423.48**	1423.82	24.56	**0.00**
pr04	192	4	1514.07*	1524.18	54.45	0.67	**1514.07**	1755.56	**0.00**	**1514.07**	278.91	**0.00**	**1514.07**	1514.96	64.50	**0.00**
pr05	240	4	1699.40	1700.93	116.91	0.09	1716.02	7272.00	0.98	1699.40	376.49	0.00	**1692.61**	1695.94	189.34	**-0.40**
pr06	288	4	1978.46	1992.26	161.37	0.70	1978.46	7272.00	0.00	1982.13	417.65	0.19	**1977.30**	1979.97	191.71	**-0.06**
pr07	72	6	821.25*	**821.25**	10.62	**0.00**	**821.25**	0.30	**0.00**	**821.25**	130.64	**0.00**	**821.25**	821.25	5.10	**0.00**
pr08	144	6	1254.45*	1258.64	29.16	0.33	**1254.45**	8.53	**0.00**	**1254.45**	252.46	**0.00**	**1254.45**	1255.83	32.68	**0.00**
pr09	216	6	1591.78*	1592.77	77.13	0.06	**1591.78**	329.48	**0.00**	**1591.78**	414.37	**0.00**	**1591.78**	1591.78	112.89	**0.00**
pr10	288	6	1969.35	1975.36	157.05	0.31	1997.96	7272.00	1.45	1969.35	597.89	0.00	**1968.67**	1972.83	195.31	**-0.03**
Mean	-	-	1398.69	1402.74	58.11	0.22	1411.14	2576.75	0.51	1399.38	190.94	0.03	1397.39	1398.80	78.99	-0.06

List of Figures

1.1 Illustration of the Traveling Salesman Problem. 16
1.2 Illustration of the Vehicle Routing Problem. 17

2.1 Taxonomy of metaheuristics for routing problems and their corresponding GPU parallelism paradigms. 38

3.1 Illustration of the route descent procedure. The solution has four routes. First, the route with the fewest nodes 4-12 is removed. Then, the nodes 4 and 12 are inserted into other routes using the infeasible regret insertion strategy. 65
3.2 Illustration of operators and concatenation operation. 67
3.3 Illustration of the ARIX crossover. 72
3.4 Examples of time-to-target analysis between MA-FIRD and MATE. The x-axis represents the running time to reach the target value and the y-axis indicates the cumulative probability of reaching the given target value. . . 78
3.5 Comparison of MA-FIRD with its variants. The x-axis denotes the instance names and the y-axis illustrates the relative gap between the solutions obtained by variants compared to those of MA-FIRD. A positive gap indicates that the variant performs worse than MA-FIRD, and vice versa. 80
3.6 Two solutions on the instance *RCdp101* with the distance of 32.17. 81
3.7 Two solutions on the instance *Rdp211* with the distance of 72.82. 81
3.8 Comparison of the MA-FIRD algorithm and its variants. The x-axis denotes the instance groups and the y-axis illustrates the relative gap between the solutions obtained by variants compared to those of MA-FIRD. The mean relative gap for each group is indicated by a solid line, while the 0.95 confidence interval is represented by the corresponding shadow. 83

4.1 Illustration of operators and their concatenation operations. Dotted boxes represent the route subsequences. Symbols $\oplus$ denote the concatenation operations. 89

4.2 Illustration of a route of length $|R| = 5$ and its corresponding attribute matrix. The first (0) and last (4) nodes represent the depot. 91

4.3 Workflow of the TGA framework. 93

4.4 Illustration of the solution tensor representation. 95

4.5 Example of the extraction operation for *2-opt*. For clarity, we illustrate a single-attribute slice of the solution tensor and show a 3D tensor with dimensions $J \times K \times K = 2 \times 10 \times 10$, omitting the first dimension of $\mathbf{T}_s$. . 96

4.6 Example of the extraction operation for *Intra-Relocate*. For clarity, we illustrate a single-attribute slice of the solution tensor and show a 3D tensor with dimensions $J \times K \times K = 1 \times 10 \times 10$, omitting the first dimension of $\mathbf{T}_s$ and setting $J = 1$. 97

4.7 Example of the extraction operation for *Intra-Swap* with $N_1 = 1, N_2 = 2$. For clarity, we illustrate a single-attribute slice of the solution tensor and show a 3D tensor with dimensions $J \times K \times K = 1 \times 10 \times 10$, omitting the first dimension of $\mathbf{T}_s$ and setting $J = 1$. 98

4.8 Example of the route-based *3-Seq* operation with $N = 1$ for inter-route operators. For clarity, we illustrate a single-attribute slice of the solution tensor and show a 3D tensor with dimensions $J \times K \times K = 2 \times 10 \times 10$. Note that the length of the routes affects the tail sequence $\mathbf{S}_t$. 99

4.9 Example of the node-based *3-Seq* operation. To illustrate the differences between node-based and route-based implementations, we reuse the same setting as in Figure 4.8. In this example, five nodes are distributed across two routes: the 2nd, 3rd, and 6th nodes on route R_1, and the 3rd and 5th nodes on route R_2. The figure shows how the head, middle, and tail sequences are extracted based on node positions, and highlights their relationship to those obtained through the route-based implementation. 100

4.10 Comparative performance of MA-N, MA-E, MA-NTGA, and MA-ETGA on three benchmark sets. The x-axis shows runtime (s), and the y-axis shows average gap (%) relative to the baseline MA-N. 111

4.11 Relationship between instance size and speedup ratio across benchmark instances. The x-axis denotes the instance size, while the y-axis reports the speedup of TGA-based algorithms over their CPU-based counterparts. . . . 112

4.12 Speedup ratios of local search operators across benchmark instances, including inter-route operators (*Inter-Relocate*, *Inter-Swap*, *2-opt**) with node-based (XR(N), XS(N), TOS(N)) and edge-based implementations (XR(E), XS(E), TOS(E)), as well as route-based intra-route operators (*Intra-Relocate* (IR), *Intra-Swap* (IS), *2-opt* (TO)). The x-axis denotes instance size and the y-axis the corresponding speedup ratio. 114
4.13 Time analysis of TGA's key steps for varying instance sizes. The x-axis indicates instance size, and the y-axis represents computation time for each step. 115
4.14 Comparative performance of TGA-based algorithms and state-of-the-art methods on benchmark instances. The x-axis shows runtime (s), and the y-axis shows gap (%) relative to the BKS. 117

5.1 Illustration of the two dedicated depot operators *Depot-Insert* and *Depot-Replace*, along with their concatenation operations. Yellow nodes represent the original depot nodes, while "H" and "T" denote the head and tail of the route, respectively. The blue "D" nodes represent the new depot being inserted or used for replacement. 128
5.2 Illustration of the diversity-controlled route-exchange crossover. "A" and "B" are depot nodes. Blue indicates the routes added or nodes retained, while yellow represents the routes or nodes that are removed. 134
5.3 Performance comparison of MDFIHA and its variants. The x-axis indicates instances, and the y-axis shows the relative gap to the baseline MDFIHA. Colored lines denote mean gaps with 0.95 confidence intervals. 142
5.4 Evolution of population diversity across generations for MDFIHA and its variants MDFIHA-SREX and MDFIHA-RARI. The diversity is measured by the solution distances within population. 143
5.5 Performance comparison of MDFIHA and its variants. The x-axis indicates algorithm variants, and the y-axis shows the relative gap to the baseline MDFIHA. 144
5.6 Illustration of depot-related operators. Red triangles denote depots, blue points represent customers. Blue edges indicate unchanged routes, yellow edges highlight modified routes. 145
5.7 Performance comparison between MDFIHA-ETGA and MDFIHA-ETGA-S, with VCGP [Vid+13] serving as the baseline. 146

5.8 ETGA speedup analysis on *V13* instances. The x-axis indicates the instance sizes, y-axis represents the speedup ratio. 147

List of Tables

1.1 Summary of the algorithms for VRPSPDTW in the literature. 21

2.1 Publications on GPU-based metaheuristics for routing problems by year (2010-2024) 45
2.2 Comprehensive summary of GPU-based metaheuristics for routing problems (2010-2025). 47

3.1 Parameters tuning results. 75
3.2 CPU information and time conversion ratio γ. 76
3.3 Summary of comparative results on benchmark instances between MA-FIRD and the references. 77

4.1 Comparison of NVIDIA GPUs: V100, A100, H100. 87
4.2 Per-step space and time complexity of tensor-based move evaluation for intra-route operators (route-based), and inter-route operators (route-based, node-based, and edge-based). 105
4.3 Characteristics of the four algorithm variants. 109
4.4 Parameter tuning results. 110
4.5 Processor specifications and corresponding CPU benchmark scores. 116

5.1 Parameter tuning results. 138
5.2 Reference algorithms for benchmark sets. 139
5.3 Processor information and conversion ratio. 139
5.4 Summary of comparative results on benchmarks between the proposed algorithms and the reference algorithms. 140

A.1 Comparative results on the 9 small WC benchmark instances between MATE, MATE* and MA-FIRD. 169
A.2 Comparative results on the 56 medium WC benchmark instances between BKS, ALNS-PR, MATE, MATE* and MA-FIRD. 170

A.3 Comparative results on the 20 large-scale JD benchmark instances between MATE, MATE* and MA-FIRD. . . . 171
B.4 Comparative results on the 100 *X* benchmark instances for MA-N, MA-E, MA-NTGA, and MA-ETGA. . . . 173
B.5 Comparative results on the 100 *X* benchmark instances for MA-N, MA-E, MA-NTGA, and MA-ETGA (continued). . . . 174
B.6 Comparative results on the 90 *GH* benchmark instances for MA-N, MA-E, MA-NTGA, and MA-ETGA. . . . 174
B.7 Comparative results on the 90 *GH* benchmark instances for MA-N, MA-E, MA-NTGA, and MA-ETGA (continued). . . . 175
B.8 Comparative results on the 20 *JD* benchmark instances for MA-N, MA-E, MA-NTGA, and MA-ETGA. . . . 175
B.9 Comparative results on the 100 *X* benchmark instances for HGS-2012, HGS-CVRP, MA-NTGA, and MA-ETGA. . . . 176
B.10 Comparative results on the 100 *X* benchmark instances for HGS-2012, HGS-CVRP, MA-NTGA, and MA-ETGA (continued). . . . 177
B.11 Comparative results on the 90 *GH* benchmark instances for DIMACS, HGS-DIMACS, MA-NTGA, and MA-ETGA. . . . 177
B.12 Comparative results on the 90 *GH* benchmark instances for DIMACS, HGS-DIMACS, MA-NTGA, and MA-ETGA (continued). . . . 178
B.13 Comparative results on the 20 *JD* benchmark instances for MA-FIRD, MA-NTGA, and MA-ETGA. . . . 179
C.14 Comparative results on the 33 *C97* MDVRP benchmark instances between BKS, CGL, PR, VCGLR, ELTG, SCA and MDFIHA. . . . 179
C.15 Comparative results on the 10 *C97-T* MDVRP benchmark instances between Optima, SCA and MDFIHA. . . . 180
C.16 Comparative results on the 20 *C01* MDVRPTW benchmark instances between BKS, VCGP, SCA and MDFIHA. . . . 180
C.17 Comparative results on the 10 *C01-R* MDVRPTW benchmark instances between BKS, VCGP and MDFIHA. . . . 180
C.18 Comparative results on the 28 *V13* MDVRPTW benchmark instances between VCGP, MDFIHA and MDFIHA-ETGA. . . . 181
C.19 Comparative results on the 24 *L14* MDOVRP benchmark instances between BKS, JB, RM, SCA and MDFIHA. . . . 181

List of Publications

Published/accepted papers

- **Zhenyu Lei**, and Jin-Kao Hao. An effective memetic algorithm for the close-enough traveling salesman problem. *Applied Soft Computing* 153 (2024): 111266.
- **Zhenyu Lei**, and Jin-Kao Hao. A Memetic Algorithm for Vehicle Routing with Simultaneous Pickup and Delivery and Time Windows. *IEEE Transactions on Evolutionary Computation* 29.5 (2025): 1924-1936.

Submitted papers

- **Zhenyu Lei**, Jin-Kao Hao, and Qinghua Wu. Speeding up Local Optimization in Vehicle Routing with Tensor-based GPU Acceleration. *Transportation Science*, initial submission, May 2025; revised and re-submitted, February 2026.
- **Zhenyu Lei**, and Jin-Kao Hao. GPU-Accelerated Hybrid Method for a Class of Multi-Depot Vehicle Routing Problems. *European Journal of Operational Research*, submitted, June 2026.

COLLEGE ED MATHEMATIQUES
DOCTORAL STIC
PAYS DE LA LOIRE

université angers

Titre : Algorithmes mémétiques et accélération GPU pour les problèmes de routage multi-attributs

Mot clés : Problème de routage de véhicules, Métaheuristiques, Algorithme mémétique, Accélération GPU

Résumé : Cette thèse présente l'état de l'art des métaheuristiques sur GPU et des problèmes de routage de véhicules multi-attributs, puis propose des algorithmes mémétiques et des techniques d'accélération GPU pour les résoudre efficacement. Pour le VRPSPDTW à dépôt unique, caractérisé par de multiples contraintes complexes, nous proposons MA-FIRD, un algorithme mémétique intégrant une recherche locale efficace. Afin d'améliorer l'efficacité, nous introduisons un cadre d'accélération GPU basé sur les tenseurs, exploitant le parallélisme des GPU pour accélérer l'évaluation des voisinages en recherche locale. TGA est extensible à diverses variantes du VRP et s'intègre facilement à différents cadres algorithmiques, tout en offrant des gains substantiels par rapport aux implémentations CPU. Pour une classe de VRP multi-dépôts, nous concevons MDFIHA ainsi que sa version améliorée MDFIHA-ETGA, intégrant la TGA basée sur les arêtes et une stratégie de mise à jour multi-mouvements afin d'accroître encore l'efficacité. Des expérimentations approfondies sur des instances de référence démontrent leur compétitivité face aux méthodes de l'état de l'art. Les contributions de cette thèse font progresser l'optimisation du routage de véhicules et offrent une perspective fondée sur le GPU pour la résolution de variantes complexes et de grande taille du VRP.

Title: Memetic Algorithms and GPU Acceleration for Multi-Attributes Routing Problems

Keywords: Vehicle routing problem, Metaheuristics, Memetic algorithm, GPU acceleration

Abstract: This thesis reviews state-of-the-art for GPU-based metaheuristics and representative multi-attribute vehicle routing problems, and proposes memetic algorithms and GPU acceleration techniques to solve them. For the single-depot VRPSPDTW with multiple complex constraints, we propose MA-FIRD, a memetic algorithm with an effective local search component. To improve efficiency, we introduce a tensor-based GPU acceleration framework that leverages GPU parallelism to speed up neighborhood evaluation in local search. TGA is extensible to various VRP variants and adaptable across algorithmic frameworks, demonstrating its substantial efficiency gains over CPU implementations. For a class of multi-depot VRPs, we design MDFIHA and its enhanced version MDFIHA-ETGA, which incorporates the edge-implementd TGA and a multi-move update strategy to further improve efficiency. Extensive experiments on benchmark instances demonstrate their competitiveness against state-of-the-art methods in both solution quality and efficiency. The contributions of this thesis advance vehicle routing optimization and provides a GPU-based perspective for solving large-scale and complex VRP variants.