MOHAMED BIN ZAYED
UNIVERSITY OF
ARTIFICIAL INTELLIGENCE
LIBRARY

Machine Learning for Combinatorial Optimization

Authors	Iklassov, Zangir
Citation	Z. Iklassov, "Machine Learning for Combinatorial Optimization", PhD Dissertation, Machine Learning, MBZUAI, Abu Dhabi, UAE, 2024
Download date	2026-11-01 12:55:40
Link to Item	https://hdl.handle.net/20.500.14634/521

Machine Learning for Combinatorial Optimization

by

Zangir Iklassov

Submitted in partial fulfillment of the requirement for
the degree of
Doctor of Philosophy
in
Department of Machine Learning

Mohamed bin Zayed University of Artificial Intelligence
Abu Dhabi, UAE

December 2024
Copyright © 2024, Zangir Iklassov
All Rights Reserved

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

Supervisor:	Dr. Martin Takáč Deputy Department Chair of Machine Learning Associate Professor, Department of Machine Learning Mohamed bin Zayed University of Artificial Intelligence (MBZUAI)
External Member:	Dr. Lawrence V. Snyder Deputy Provost for Faculty Affairs Professor of Industrial and Systems Engineering Lehigh University
Internal Member:	Dr. Bin Gu Assistant Professor, Department of Machine Learning Mohamed bin Zayed University of Artificial Intelligence (MBZUAI)
Internal Member:	Dr. Zhiqiang Xu Assistant Professor, Department of Machine Learning Mohamed bin Zayed University of Artificial Intelligence (MBZUAI)
Internal Member:	Dr. Karthik Nandakumar Associate Professor, Department of Computer Vision Mohamed bin Zayed University of Artificial Intelligence (MBZUAI)

Author's Declaration

I hereby declare that I am the author of this thesis. Some chapters are based on work from jointly-authored publications: Chapter 2 is based on the paper [74], Chapter 3 is derived from the papers [76], [77], and [78], and Chapter 4 is built upon the paper [73]. All sources of information have been duly acknowledged, and this thesis has not been submitted previously, in whole or in part, for any degree or diploma at any other institution.

I understand that this paper may be made electronically available to the public.

Abstract

Combinatorial optimization problems (CPs) represent a significant challenge in various fields, including logistics, resource management, and scheduling, due to their inherent NP-hardness and computational complexity. Traditional methods such as exact algorithms, heuristics, and metaheuristics often fall short in handling large-scale instances and adapting to dynamic environments. This thesis explores the application of machine learning, particularly neural combinatorial optimization and reinforcement learning, to address these limitations and improve the efficiency and effectiveness of solving CPs.

The first part of this thesis provides a comprehensive overview of combinatorial optimization problems, highlighting their importance and the challenges associated with solving them. It then reviews the literature on traditional methods and modern machine learning approaches, identifying gaps and opportunities for advancement.

The core of the thesis is based on insights from five key research papers, each contributing novel methodologies and significant findings to the field:

- Curriculum Learning for Job Shop Scheduling: This study introduces a reinforced adaptive staircase curriculum learning strategy for job shop scheduling, significantly improving the optimality gap over state-of-the-art models by dynamically adjusting difficulty levels and focusing on problematic instances.
- Reinforcement Learning for Vehicle Routing with Problem with Stochastic Correlated Demands: This research presents an RL-based framework for the vehicle routing problem with stochastic demands, incorporating correlated demands through observable variables. The proposed model outperforms state-of-the-art metaheuristics, demonstrating robustness in various environmental conditions.
- RL for Vehicle Routing Problem with Stochastic Travel Times: This paper extends the application of RL to solving the stochastic vehicle routing problem (SVRP), addressing key sources of stochasticity and achieving a 3.43% reduction in travel costs compared to existing methods, while exhibiting robustness across different SVRP settings.
- RL for SVRP with Time Windows: A novel RL framework is introduced for the SVRP with time windows, focusing on reducing travel costs while adhering to customer time constraints. The model demonstrates a 1.73% reduction in travel costs and robustness in diverse environments, making it a valuable benchmark for future SVRP studies.
- Self-Guiding Exploration for Combinatorial Problems: This paper proposes the Self-Guiding Exploration (SGE) strategy, a new prompting method for large language models (LLMs) designed to enhance CP-solving performance. SGE generates multiple thought trajectories, decomposes them into actionable subtasks, and refines the results, outperforming existing prompting strategies by 27.84% in optimization performance.

Each chapter dedicated to these papers details the methodologies, experimental setups, results, and insights gained from the studies, highlighting their contributions to the field of combinatorial optimization.

The thesis concludes with a synthesis of findings from the five papers, providing a comparative analysis of the different approaches and their performance. It also discusses the overall contributions to machine learning and combinatorial optimization, emphasizing the advancements made through this research. Finally, it outlines potential future research directions, including enhancing model efficiency and applicability to a broader range of problems.

This thesis demonstrates that machine learning, particularly reinforcement learning and advanced prompting strategies for LLMs, offers significant potential in solving complex combinatorial optimization problems, paving the way for more efficient and adaptable solutions in various real-world applications.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my advisor, Professor Martin Takáč. His unwavering support, guidance, and dedication have been the cornerstone of my academic journey. His commitment to excellence and his enthusiasm for research have profoundly influenced my approach to both academics and life. I am truly fortunate to have had the opportunity to work under his mentorship, and his wisdom and encouragement have been instrumental in the completion of this thesis.

I would also like to acknowledge the exceptional faculty and staff of the university. Their dedication to creating an enriching and supportive academic environment has made my journey not only successful but also enjoyable. Their hard work and commitment to the students' well-being have made it easy to live and study here, and their efforts are greatly appreciated.

My sincere appreciation goes to my fellow researchers, M.S. students Dmitrii Medvedev, Ikboljon Sobirov, Farkhad Akimov, Dr. Ruben Solozabal, and Professor Yali Du. Their brilliance, camaraderie, and unwavering support have been invaluable throughout the course of this research. Collaborating with such talented and dedicated individuals has not only enriched this work but also made the research journey a memorable and rewarding experience. Their contributions and friendship have been a source of strength and inspiration.

I extend my heartfelt thanks to His Highness Mohamed bin Zayed for his visionary leadership and generosity in providing opportunities for students like myself to pursue our studies at such a prestigious institution. His commitment to advancing education and fostering an environment of innovation and excellence has made it possible for many to realize their academic aspirations. The opportunity to study at this advanced university has been a transformative experience, and for this, I am deeply grateful.

Finally, I would like to express my deepest gratitude to my family, especially my mother and my grandmother. Their unconditional support, love, and encouragement have been the bedrock upon which I have built my academic and personal achievements. Their invaluable help and unwavering belief in my abilities have given me the strength and determination to persevere through the challenges of this journey. I am forever indebted to them for their sacrifices and for always being there for me.

In conclusion, this thesis is a testament to the collective efforts, guidance, and support of many remarkable individuals. I am profoundly grateful to all who have contributed to this journey, and I dedicate this work to them with sincere appreciation and respect.

Lay Summary of Thesis

This research focuses on solving complex decision-making problems that are important in everyday areas like transportation, managing resources, and planning schedules. Traditional methods often struggle to handle these challenges, especially when dealing with uncertainty or large-scale tasks. The study uses advanced machine learning techniques, including artificial intelligence models that learn and adapt, to overcome these issues. Highlights include improving scheduling by teaching models to tackle harder tasks step-by-step and developing smarter systems to plan vehicle routes even when conditions are unpredictable. The research also explores how AI language models can help break down and solve these problems more efficiently. Overall, this work shows how machine learning can make these processes faster, smarter, and more flexible in real-world situations.

Table of Contents

List of Tables **xi**

List of Figures **xiii**

1 Introduction **2**

1.1 Overview of Combinatorial Optimization Problems 2

1.1.1 Definition and Significance of Combinatorial Optimization Problems (CPs) 2

1.1.2 Examples of Common Combinatorial Optimization Problems 5

1.2 Traditional Approaches and Their Limitations: Exact Methods, Heuristics, and Metaheuristics 9

1.2.1 Exact Methods 9

1.2.2 Heuristics: Nearest Neighbor, Greedy Algorithms 10

1.2.3 Metaheuristics: Genetic Algorithms, Simulated Annealing, Ant-Colony Optimization 12

1.3 Role of Machine Learning in Combinatorial Optimization 14

1.3.1 Introduction to Machine Learning Techniques in Combinatorial Optimization Problems 14

1.3.2 Neural Combinatorial Optimization (NCO) 16

1.3.3 Reinforcement Learning (RL) Techniques in CPs 16

1.3.4 Curriculum Learning (CL) Strategies 18

1.3.5 Generative Models 19

1.3.6 Large Language Models (LLMs) 20

1.3.7 Representation of Problems as Markov Decision Processes (MDPs) . . . 21

1.4 Overcoming Challenges in Combinatorial Optimization Research 23

1.5 Scope and Contributions of This Thesis 26

1.5.1 Objectives and Research Questions 26

1.5.2 Overview of the Five Key Papers and Their Contributions to the Field . . 27

1.5.3 Structure of the Thesis 29

2 Learning Algorithm for Job-Shop Scheduling Problem **32**

2.1 Introduction . . . 32

2.2 Related Work . . . 33

2.3 Method . . . 35

2.3.1 Learning the Dispatching Policy . . . 35

2.3.2 Inference Strategies . . . 37

2.4 Architecture Details . . . 37

2.5 Learning Using a Curriculum Strategy . . . 38

2.6 Experiments . . . 40

2.7 Conclusions and Future Work . . . 44

2.8 Models and Configurations . . . 45

2.9 Additional Tables . . . 46

3 Learning Algorithm for Stochastic Vehicle Routing Problem **51**

3.1 Learning Algorithm for Stochastic Vehicle Routing Problem with Correlated Demands . . . 52

3.1.1 Introduction . . . 52

3.1.2 Related Work . . . 53

3.1.3 Background . . . 55

3.1.4 Method . . . 58

3.1.5 Architecture Details . . . 63

3.1.6 Experiments . . . 64

3.1.7 Discussion and Conclusion . . . 70

3.2 Learning Algorithm for Vehicle Routing Problem with Stochastic Travel Times . 71

3.2.1 Introduction . . . 71

3.2.2 Method . . . 73

3.2.3 Environment Settings . . . 74

3.2.4 Architecture Details . . . 76

3.2.5 Experiments . . . 76

3.2.6 Conclusion . . . 80

3.2.7 Implementation . . . 81

3.2.8 Results . . . 82

3.2.9 Training Curves . . . 82

3.3 Learning Algorithm for Stochastic Vehicle Routing Problem with Time Windows 83
3.3.1 Introduction . . . 83
3.3.2 Method . . . 85
3.3.3 Architecture . . . 89
3.3.4 Experiments . . . 91
3.3.5 Conclusion . . . 96
3.3.6 Implementation . . . 96
3.3.7 Training Curves . . . 97

4 Large Language Models for Combinatorial Problems 98
4.1 Introduction . . . 98
4.2 Related Work . . . 99
4.3 Preliminaries . . . 100
4.4 Method . . . 102
4.4.1 Algorithm . . . 102
4.4.2 Baseline Prompting Techniques . . . 103
4.4.3 Combinatorial Problems . . . 104
4.5 Experiments . . . 106
4.5.1 Setup . . . 107
4.5.2 Results on CP Tasks . . . 107
4.5.3 Results on Reasoning Tasks . . . 112
4.6 Conclusion . . . 113

5 Conclusions and Future Work 124
5.1 Summary of Key Findings . . . 124
5.1.1 Recap of Results and Contributions to the Field . . . 124
5.2 Future Research Directions . . . 126

References 128

APPENDICES 128

List of Tables

2.1 Results on (a) Taillard's instances. 42
2.2 Results on (b) DMU instances. 43
2.3 Inference strategies for RASCL on Taillard's instances. 43
2.4 Hyper-parameter values. 45
2.5 Results on Taillard's instances. 47
2.6 Results on DMU instances. 48
2.7 Curriculum strategies on Taillard's instances. 49
2.8 Inference strategies for RASCL on Taillard's instances. 50

3.1 Results of ACO algorithm. 65
3.2 Results of baseline and the RL models. 66
3.3 Results of different inference strategies. 66
3.4 Results of different types of demand estimates. 66
3.5 Results of different types of customer positions. 68
3.6 Results of different proportions of demand terms. 68
3.7 Results of different filling rates. 68
3.8 Results of different supply types. 68
3.9 Results of a varying number of vehicles. 69
3.10 Average inference time 69
3.11 The performance of the ACO algorithm. 78
3.12 The results of the baseline and RL models. 78
3.13 The performance of different inference strategies. 79
3.14 The results from two different types of variable estimates. 79
3.15 The results of various stochasticity sources on the SVRPTT test dataset. 79
3.16 The results obtained from various signal ratios on the SVRPTT test dataset. . . . 80
3.17 The results obtained from two types of positions on the SVRPTT test dataset. . . 80

3.18 The results obtained from various fill rates on the SVRPTT test dataset. 80
3.19 The results obtained from various number of vehicles on the SVRP test dataset. . 81
3.20 Inference time results in seconds on the SVRPTT test dataset. 81
3.21 The results obtained from various delivery types on the SVRPTT test dataset. . . 82
3.22 The performance of the ACO algorithm. 92
3.23 The results obtained from the baseline and RL models on the SVRPTW test dataset. 92
3.24 The effectiveness of inference strategies. 92
3.25 The results of different stochasticity settings on the SVRPTW test dataset. 93
3.26 The results derived from different signal ratios on the SVRPTW test dataset. . . . 93
3.27 The results derived from two positioning approaches on the SVRPTW test dataset. 93
3.28 The results derived from different fill rates on the SVRPTW test dataset. 94
3.29 The results obtained from various delivery types on the SVRPTW test dataset. . . 95
3.30 The results obtained with different numbers of vehicles on the SVRPTW test dataset. 95
3.31 Results for the inference time in seconds on the SVRPTW test dataset. 96

4.1 Percentage performance improvement compared to IO on CP tasks. 108
4.2 Optimality gap of prompting method. 110
4.3 Results for reasoning tasks. 112
4.4 Performance vs efficiency of prompting methods. 112
4.5 Results for reasoning tasks using gpt-3.5. 113
4.6 Comparison of various combinatorial problems using gpt-3.5. 114
4.7 Comparison of various combinatorial problems using gpt-4. 115
4.8 Comparison of various combinatorial problems using gemini-1.5. 116
4.9 Comparison of various combinatorial problems using llama-2-7b. 117
4.10 Comparison of various combinatorial problems using llama-2-70b. 118

List of Figures

2.1 Model Architecture. 36
2.2 Optimatlity gap on the (a) curriculum strategies and (b) inference strategies. . . . 39
2.3 Increment in the optimality gap (in percent) on Taillard's instances. 41
2.4 Average makespan observed during the training. 44

3.1 Correlated demands example. 59
3.2 Network architecture. 63
3.3 Average travel cost gap. 67
3.4 Average travel cost during training. 70
3.5 Average travel cost during training. 70
3.6 Average travel cost during training. 71
3.7 Network architecture. 77
3.8 Percentage gap in travel times. 82
3.9 The travel cost during the training with varying variable estimates. 83
3.10 The travel cost during the training with varying customer positions. 83
3.11 The travel cost during the training phase with varying delivery types 83
3.12 Vehicle routing problem example. 84
3.13 Model architecture. 90
3.14 Percentage difference in travel times. 94
3.15 The travel cost for varying customer positioning approaches. 97
3.16 The travel cost for varying product delivering approaches. 97

4.1 Self-Guiding Exploration. 101
4.2 Example of SGE inference. 109
4.3 Effect of problem size on performance improvement. 110
4.4 Effect of model choice on performance improvement. 111

List of Abbreviations

ACO	Ant Colony Optimization
DMU	Demirkol's dataset
DRL	Deep Reinforcement Learning
JSP	Job-Shop Scheduling Problem
LSTM	Long short-term memory neural network
ML	Machine Learning
RL	Reinforcement Learning
rLSTM	Reverse LSTM
SVRP	Stochastic Vehicle Routing Problem
UCL	Uniform Curriculum Learning
VRP	Vehicle Routing Problem
VRPSD	Vehicle Routing Problem with Stochastic Demands
VRPSTT	Vehicle Routing Problem with Stochastic Travel Times

Chapter 1

Introduction

1.1 Overview of Combinatorial Optimization Problems

1.1.1 Definition and Significance of Combinatorial Optimization Problems (CPs)

Combinatorial Optimization Problems are mathematical problems in which the objective is to determine the best possible solution from a finite set of feasible solutions [159]. These problems are characterized by their discrete nature, meaning that the variables involved can take on only a limited number of distinct values, typically integers. Unlike continuous optimization problems, where solutions can vary smoothly within a range, CPs require selecting the optimal combination of elements from a predefined set, subject to specific constraints.

Formally, a combinatorial optimization problem can be defined as follows:

- Let S be a finite set of feasible solutions.
- Let $f : S \to \mathbb{R}$ be an objective function that assigns a real number (representing cost, distance, time, profit, etc.) to each solution in S.
- The goal is to find a solution $s^* \in S$ such that:

  $$f(s^*) = \min_{s \in S} f(s),$$

  or $\max_{s \in S} f(s)$ for maximization problems. Subject to the constraints:

  $$g_i(s) \leq b_i, \quad \text{for } i = 1, 2, \ldots, m,$$

  where m represents the total number of constraints, $g_i(s)$ is the i-th constraint function that depends on the solution s, and b_i is the corresponding bound or limit for the i-th constraint. Each constraint ensures that the solution s adheres to specific requirements or limitations, such as resource availability, capacity limits, or other problem-specific conditions. The solution s^* is considered optimal if it minimizes (or maximizes) the objective function f while satisfying all the given constraints.

Discrete Solution Space. The solution space in combinatorial problems is inherently discrete, consisting of distinct and separate solutions [67]. Each solution typically represents a combination or permutation of elements, such as sequences, sets, or graphs. For example, in the Traveling Salesman Problem (TSP), the solution space is composed of all possible permutations of a given set of cities. The TSP seeks to find the shortest possible route that visits each city exactly once and returns to the starting point. Mathematically, if there are n cities, the number of possible solutions (or tours) is $n!$, representing all permutations of the cities. The goal is to minimize the total distance traveled.

Objective Function. The objective function $f(s)$ is a critical component of any combinatorial optimization problem. It quantitatively evaluates each feasible solution based on specific criteria, such as cost, time, distance, or profit. The objective function serves as the benchmark for determining the quality of solutions. For example, in the TSP, the objective function might measure the total distance traveled, with the goal of minimizing this distance. Mathematically, this can be expressed as:

$$f(\pi) = \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)},$$

where π represents a permutation of the cities, $d_{\pi(i),\pi(i+1)}$ is the distance between consecutive cities in the tour, and n is the total number of cities. The optimal solution s^* is the one that minimizes the objective function:

$$f(s^*) = \min_{\pi \in S} f(\pi),$$

where S is the set of all feasible solutions (permutations of cities) in the TSP.

Constraints. Constraints define the rules or limitations that feasible solutions must adhere to. They restrict the solution space by eliminating combinations that do not satisfy certain conditions [160]. Constraints can take various forms, such as capacity limits, precedence relationships, or resource availability. For example, in the TSP, a common constraint is that each city must be visited exactly once. This can be mathematically expressed as:

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \forall i \in \{1, 2, \ldots, n\},$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad \forall j \in \{1, 2, \ldots, n\},$$

where $x_{ij} = 1$ if the tour visits city j immediately after city i, and $x_{ij} = 0$ otherwise. These constraints ensure that each city is included exactly once in the tour, forming a valid sequence that meets the problem's requirements.

NP-Hardness. Many combinatorial optimization problems are classified as NP-hard, meaning that they are at least as hard as the hardest problems in NP (nondeterministic polynomial time) [46]. This classification implies that there is no known polynomial-time algorithm that can solve

all instances of these problems efficiently. NP-hard problems exhibit exponential growth in computational complexity as the size of the input increases. For example, in the TSP with n cities, as n increases, the number of possible solutions grows factorially, making it computationally prohibitive to evaluate all possible routes. For instance, with just 10 cities, there are $10! = 3,628,800$ possible tours, and this number becomes unimaginably large as more cities are added.

Complexity and Approximation. Due to their complexity, many combinatorial problems are tackled using approximation algorithms or heuristics that aim to find near-optimal solutions within a reasonable timeframe. These methods trade off optimality for computational efficiency [85]. Approximation algorithms provide guarantees on how close the obtained solution is to the optimal one, often expressed as a ratio ρ, where the solution s satisfies:

$$\frac{f(s)}{f(s^*)} \leq \rho,$$

for a minimization problem, with $\rho \geq 1$, where s^* is the optimal solution. Heuristics, on the other hand, rely on rules of thumb or problem-specific strategies to produce good solutions quickly, often without any guarantee of how close the solution is to the optimal one. For example, in the TSP, the nearest neighbor heuristic is a common approach. This heuristic starts at an arbitrary city v_1, then repeatedly visits the nearest unvisited city v_i until all cities have been visited. Mathematically, this can be described as:

$$v_{i+1} = \arg \min_{v_j \in V \setminus \{v_1, \ldots, v_i\}} d(v_i, v_j),$$

where $d(v_i, v_j)$ is the distance between cities v_i and v_j, and V is the set of all cities. Although the nearest neighbor heuristic does not guarantee the optimal solution, it often produces a good tour in a fraction of the time required for an exhaustive search, where the number of possible solutions would be $n!$. In practice, the nearest neighbor heuristic might yield a tour that is longer than the optimal one, but it provides a feasible solution very quickly, making it useful in scenarios where time is a critical factor.

Multimodal Solution Landscape. The solution landscape of combinatorial problems is often multimodal, meaning that there are multiple locally optimal solutions that are better than neighboring solutions but not necessarily the global optimum. Mathematically, a solution s_{local} is considered a local optimum if:

$$f(s_{\text{local}}) \leq f(s),$$

for all s in the neighborhood of s_{local}, where $f(s)$ is the objective function. This characteristic of having multiple local optima poses a significant challenge for optimization algorithms, as they can easily become trapped in these local optima, preventing them from discovering better solutions that might exist elsewhere in the solution space. To overcome this challenge, metaheuristic approaches have been developed, which are specifically designed to navigate such complex landscapes and increase the likelihood of finding the global optimum. For example, simulated annealing employs a probabilistic acceptance criterion that allows the algorithm to occasionally

accept worse solutions, thus providing a mechanism to escape local optima. The acceptance probability P can be defined as:

$$P = \exp\left(\frac{f(s') - f(s)}{T}\right),$$

where s is the current solution, s' is a neighboring solution, $f(s)$ and $f(s')$ are the objective function values, and T is the temperature parameter that gradually decreases during the optimization process. This allows the algorithm to explore the solution space more broadly early on and gradually focus on exploiting the best regions as T decreases.

Intractability. The intractability of NP-hard CPs has led to the development of various heuristic and metaheuristic methods. These approaches provide practical solutions to large-scale problems where exact methods are computationally infeasible. Heuristics, such as the nearest neighbor algorithm for TSP, offer quick, though not necessarily optimal, solutions. Metaheuristics, such as genetic algorithms, tabu search, and simulated annealing, explore the solution space more thoroughly to find high-quality solutions. However, even with these advanced techniques, finding consistently effective solutions can be challenging, especially as the complexity of the problem increases. This is where machine learning algorithms show great promise. By learning from data, machine learning approaches can identify patterns and strategies that may be difficult to capture with traditional methods [10], offering new ways to approximate or even improve upon the solutions generated by heuristics and metaheuristics.

1.1.2 Examples of Common Combinatorial Optimization Problems

Combinatorial optimization problems are integral to numerous applications, spanning industries from logistics to manufacturing, finance, and beyond. These problems are characterized by their discrete nature, where the objective is to find the best possible combination or arrangement of elements within a finite set. Among the many CPs, some are particularly noteworthy due to their wide applicability and the complexity they present. This section explores some of the most prominent examples: the Traveling Salesman Problem, the Vehicle Routing Problem, the Job Shop Scheduling Problem, and others, highlighting their real-world relevance and the challenges they pose.

Traveling Salesman Problem (TSP). The Traveling Salesman Problem is a classic combinatorial optimization problem where a salesman must visit a set of n cities exactly once and return to the starting city [17]. The goal is to find the sequence of city visits (tour) that minimizes the total travel distance (or cost). Let $G = (V, E)$ be a complete graph where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of cities (vertices) and $E = \{e_{ij}\}$ is the set of edges representing the paths between cities. Each edge e_{ij} has an associated distance d_{ij}. The objective is to find a permutation π of the cities that minimizes the total travel distance:

$$\min_{\pi \in \Pi_n} \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)},$$

where Π_n is the set of all possible permutations of n cities.

The constraints ensure that each city must be visited exactly once:

$$\sum_{j=1}^{n} x_{ij} = 1 \quad \forall i \in V,$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad \forall j \in V,$$

The decision variable $x_{ij} = 1$ if the tour visits city j immediately after city i, and $x_{ij} = 0$ otherwise.

Despite its abstract nature, the TSP has numerous practical applications. It is directly applicable in logistics for optimizing delivery routes, in manufacturing for designing circuits on printed circuit boards, and even in biology for DNA sequencing. The TSP also serves as a metaphorical problem for optimizing various sequences and paths in different fields. Over the years, various methods have been developed to tackle the TSP. Exact algorithms, such as branch-and-bound and dynamic programming, are capable of finding the optimal solution but are feasible only for small instances. For larger instances, heuristic and metaheuristic approaches like the nearest neighbor method, genetic algorithms, and simulated annealing are commonly used. These methods aim to find good, if not optimal, solutions within a reasonable amount of time. Recently, machine learning-based approaches have also been explored, where neural networks and reinforcement learning are used to approximate solutions to the TSP, showing promising results in handling complex instances.

Vehicle Routing Problem (VRP). The Vehicle Routing Problem is an extension of the TSP where a fleet of vehicles must deliver goods to a set of customers from a central depot. The objective is to determine the optimal set of routes for the vehicles such that the total travel cost is minimized, while satisfying constraints such as vehicle capacity and customer demands [54]. Let $G = (V, E)$ be a complete graph where $V = \{v_0, v_1, \ldots, v_n\}$ represents the set of nodes (with v_0 being the depot and $v_1, \ldots, v_n$ being the customers), and $E = \{e_{ij}\}$ represents the edges between nodes. Each edge e_{ij} has a travel cost c_{ij}. Let K be the set of vehicles, each with a capacity Q_k. The objective is to minimize the total travel cost:

$$\min \sum_{k \in K} \sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} x_{ij}^{k},$$

where $x_{ij}^{k} = 1$ if vehicle k travels from node i to node j, and $x_{ij}^{k} = 0$ otherwise.

The constraints ensure that each customer is visited exactly once by exactly one vehicle:

$$\sum_{k \in K} \sum_{j=1}^{n} x_{ij}^{k} = 1 \quad \forall i \in V \setminus \{v_0\}.$$

The flow conservation constraint ensures that if a vehicle arrives at a customer, it must also leave:

$$\sum_{j=1}^{n} x_{ij}^{k} = \sum_{j=1}^{n} x_{ji}^{k} \quad \forall i \in V, \forall k \in K.$$

The total demand served by a vehicle must not exceed its capacity:

$$\sum_{i=1}^{n} d_i \sum_{j=1}^{n} x_{ij}^k \leq Q_k \quad \forall k \in K,$$

where d_i is the demand at customer i. The sub-tour elimination constraint ensures that no subset of customers forms a cycle disconnected from the depot. For every subset $S \subseteq V \setminus \{v_0\}$ where $|S| \geq 2$, the total number of edges connecting the nodes in S must be less than the size of S:

$$\sum_{i \in S} \sum_{j \in S} x_{ij}^k \leq |S| - 1 \quad \forall S \subseteq V \setminus \{v_0\}, \forall k \in K.$$

The VRP is critical in logistics, where it is used to optimize delivery and transportation routes. Applications include courier services, where delivery efficiency can significantly impact operating costs, and waste collection services, where optimizing routes can reduce fuel consumption and environmental impact. Exact solutions for the VRP are feasible only for small problem instances. These solutions often involve sophisticated optimization techniques like mixed-integer linear programming. Heuristics and metaheuristics, such as the Clarke-Wright savings algorithm, genetic algorithms, and ant colony optimization, are widely used to find near-optimal solutions for larger instances. Recent advancements include the application of machine learning techniques, where reinforcement learning is employed to learn efficient routing strategies that can adapt to varying conditions.

Job Shop Scheduling Problem (JSP). The Job Shop Scheduling Problem involves scheduling a set of jobs, each consisting of a sequence of operations, on a set of machines. The objective is to minimize the makespan, which is the total time required to complete all jobs [162]. Let $J = \{J_1, J_2, \ldots, J_n\}$ be the set of jobs and $M = \{M_1, M_2, \ldots, M_m\}$ be the set of machines. Each job J_i consists of a sequence of operations $O_i = \{O_{i1}, O_{i2}, \ldots, O_{ik_i}\}$, where each operation O_{ij} requires a specific machine M_{ij} for a processing time p_{ij}. The objective is to minimize the makespan $C_{\max}$, defined as:

$$C_{\max} = \min \max_{i=1,\ldots,n} C_i,$$

where C_i is the completion time of job J_i.

Precedence constraints ensure that operations within a job are processed in a specific order:

$$S_{i,j+1} \geq S_{ij} + p_{ij} \quad \forall i, \forall j,$$

where S_{ij} is the start time of operation O_{ij}. No two operations can be processed on the same machine simultaneously:

$$S_{ij} \geq S_{kl} + p_{kl} \quad \text{or} \quad S_{kl} \geq S_{ij} + p_{ij} \quad \text{if } M_{ij} = M_{kl}.$$

The JSP is crucial in manufacturing, where it is used to optimize production schedules, ensuring that machines are utilized efficiently, and production targets are met. It is also applicable in other areas such as project management, where tasks must be scheduled to meet deadlines

while respecting resource constraints. Exact methods like branch-and-bound and constraint programming can solve small instances of the JSP but become infeasible for larger problems. Heuristic approaches, such as dispatching rules and the shifting bottleneck procedure, provide good solutions for more complex instances. Metaheuristics like simulated annealing and genetic algorithms are also commonly applied. More recently, machine learning techniques, particularly reinforcement learning, have been explored to develop scheduling policies that generalize across different instances, offering promising results for dynamic and complex environments.

Other Combinatorial Optimization Problems. Beyond the TSP, VRP, and JSP, many other combinatorial optimization problems are critical in various fields, each with its own unique challenges and applications.

The Knapsack Problem involves selecting a subset of items, each with a given weight and value, to maximize the total value without exceeding a weight limit. This problem is a classic example of resource allocation where the goal is to make the best use of limited resources. The Knapsack Problem is widely used in finance for portfolio selection, in logistics for load optimization, and in budget management. Dynamic programming provides exact solutions for small instances, while greedy algorithms and approximation methods are used for larger instances.

The Bin Packing Problem requires packing a set of items of varying sizes into a finite number of bins, minimizing the number of bins used. It is a common problem in logistics and manufacturing, where efficient space utilization is critical. This problem is relevant in industries such as shipping, where it is important to pack containers efficiently, and in cloud computing, where virtual machines must be allocated to physical servers. First-fit and best-fit heuristics are commonly used, along with metaheuristics like genetic algorithms for more complex instances.

The Maximum Flow Problem involves finding the maximum flow possible from a source to a sink in a network while respecting capacity constraints on the edges. It is fundamental in network design and traffic management. This problem is used in designing communication networks, managing traffic flow in transportation systems, and optimizing supply chains. Algorithms like the Ford-Fulkerson method and the Edmonds-Karp algorithm are standard for solving maximum flow problems.

The Graph Coloring Problem involves assigning colors to the vertices of a graph such that no two adjacent vertices share the same color, with the goal of minimizing the number of colors used. This problem arises in scheduling and frequency assignment. It is widely applied in timetabling, where exams or classes must be scheduled without conflicts, and in assigning frequencies to transmitters in telecommunications to avoid interference. Greedy algorithms and backtracking are often used, along with more advanced heuristics for larger or more complex graphs.

The Steiner Tree Problem involves finding the minimum tree that spans a given set of vertices in a weighted graph, possibly including additional vertices (Steiner points) to reduce the total edge weight. It is a problem of great interest in network design. This problem is used in designing cost-effective communication networks, circuit layouts, and in evolutionary biology for constructing phylogenetic trees. Exact algorithms are feasible for small instances, while approximation algorithms and heuristics are applied to larger instances.

1.2 Traditional Approaches and Their Limitations: Exact Methods, Heuristics, and Metaheuristics

Traditional approaches to solving combinatorial optimization problems fall into three main categories: exact methods, heuristics, and metaheuristics. Each of these approaches has its strengths and limitations, and understanding these is crucial for selecting the appropriate method for a given problem.

1.2.1 Exact Methods

Exact methods are designed to find the optimal solution to a combinatorial optimization problem by exhaustively exploring the solution space [145, 190, 191]. These methods guarantee that the best possible solution will be found, but they often require significant computational resources, especially for large problem instances. Two of the most well-known exact methods are Branch and Bound and Dynamic Programming.

Branch and Bound (B&B). The Branch and Bound method is an algorithmic technique for solving combinatorial optimization problems by systematically exploring the solution space [146]. The method works by dividing the problem into smaller subproblems (branching) and calculating upper and lower bounds on the optimal solution (bounding). If a subproblem's bound indicates that it cannot yield a better solution than the current best, it is discarded (pruning).

The algorithm begins with the original problem and an initial solution, which may be obtained using a heuristic. This initial solution is set as the current best, with its objective function value denoted as $f(s_{\text{best}})$. The problem is then divided into smaller subproblems, often by making binary decisions, such as whether to include or exclude an element in the solution. For example, in the Traveling Salesman Problem, branching could involve deciding whether to include a specific edge e_{ij} between cities i and j in the tour.

For each subproblem, the algorithm calculates a lower bound $L(s)$ (the minimum possible value) and an upper bound $U(s)$ (the maximum possible value) of the objective function $f(s)$. The lower bound $L(s)$ is often calculated using a relaxed version of the problem. For example, in TSP, the lower bound could be estimated by solving the Minimum Spanning Tree (MST) for the remaining cities:

$$L(s) = \text{MST}(G' \cup \{e_{ij}\}),$$

where G' is the graph of unvisited cities, and e_{ij} is the edge currently being considered. The upper bound $U(s)$ might be calculated by completing the current partial solution using a heuristic, such as the nearest neighbor algorithm, to estimate the total tour length:

$$U(s) = f(s_{\text{current}}) + \text{NN}(G'),$$

where $\text{NN}(G')$ is the tour length estimated by the nearest neighbor heuristic applied to the remaining cities. If the lower bound $L(s)$ of a subproblem is greater than the objective value of the current best solution $f(s_{\text{best}})$, the subproblem is pruned, as it cannot lead to a better solution.

Otherwise, the subproblem is further explored. The algorithm continues this process of branching, bounding, and pruning until all subproblems have been explored or pruned, at which point the current best solution s_{best} is the optimal solution.

For instance, in the TSP, the B&B algorithm would start with the complete graph representing all possible routes. At each branch, the problem is divided by fixing a subset of cities and exploring possible routes from those cities. The algorithm calculates bounds for each subset of routes and prunes those that cannot improve the current best solution, gradually narrowing down the possibilities until the optimal tour is identified. Despite its effectiveness, the main limitation of B&B is its computational complexity, as the number of subproblems can grow exponentially with problem size, making it impractical for large instances.

Dynamic Programming (DP). Dynamic Programming is a method for solving complex problems by breaking them down into simpler subproblems. It is particularly effective for optimization problems that exhibit overlapping subproblems and an optimal substructure, meaning that the optimal solution to a problem can be constructed from the optimal solutions to its subproblems [103]. DP operates on the principle that if an optimal solution to a problem can be composed of optimal solutions to its subproblems, the problem is said to have an optimal substructure. Moreover, if these subproblems overlap—meaning the same subproblems are solved multiple times—DP can store the results of these subproblems to avoid redundant calculations, a process known as memoization.

The algorithm typically follows a bottom-up approach where smaller subproblems are solved first, and their solutions are used to solve larger subproblems. A classic example of DP is the Knapsack Problem, where the objective is to maximize the total value of items placed in a knapsack without exceeding its capacity. The DP approach involves solving subproblems for smaller capacities and fewer items, storing the results, and using them to construct the solution for the full problem. The recursive formula for the 0/1 Knapsack Problem is:

$$V(i, w) = \max\left(V(i-1, w), V(i-1, w-w_i) + v_i\right),$$

where $V(i, w)$ represents the maximum value that can be achieved with the first i items and a capacity w, w_i is the weight of item i, and v_i is its value. While DP is powerful for problems with overlapping subproblems and optimal substructure, its primary limitation is its high memory requirement, as it needs to store solutions for all subproblems. Additionally, DP can be computationally intensive for problems with a large number of subproblems.

1.2.2 Heuristics: Nearest Neighbor, Greedy Algorithms

Heuristics are problem-solving strategies that use rules of thumb or intuitive judgments to find good, but not necessarily optimal, solutions to combinatorial optimization problems [62, 63, 70, 108, 128, 167, 217, 222, 224, 228, 244]. These methods are particularly useful when finding an exact solution is not feasible due to time or computational constraints.

Nearest Neighbor (NN). The Nearest Neighbor heuristic is a simple and intuitive algorithm commonly used to solve routing problems, such as the Traveling Salesman Problem [99]. The basic idea is to start at a random node (or city) and iteratively visit the nearest unvisited node until all nodes have been visited.

The algorithm begins by selecting an arbitrary starting node, denoted as v_1. From the current node v_i, the algorithm chooses the nearest unvisited node v_j, such that the distance $d(v_i, v_j)$ is minimized:

$$v_{i+1} = \arg \min_{v_j \in V \setminus \{v_1, \ldots, v_i\}} d(v_i, v_j),$$

where V is the set of all nodes. The chosen node is then marked as visited, and the algorithm moves to it, repeating the process until all nodes have been visited. Once all nodes are visited, the algorithm returns to the starting node to complete the tour.

For example, in the TSP, the NN algorithm might start at city A, then move to the nearest city B, then from B to the nearest unvisited city C, and so on, until all cities have been visited and the salesman returns to city A. Although the NN heuristic is computationally efficient with a time complexity of $O(n^2)$, it does not guarantee an optimal solution and can produce suboptimal results, particularly for larger or more complex problem instances. The quality of the solution depends heavily on the starting node, which can lead to different outcomes.

Greedy Algorithms. Greedy algorithms are a class of heuristics that make a series of locally optimal choices in the hope of finding a globally optimal solution. At each step, the algorithm selects the option that provides the most immediate benefit without considering the long-term consequences [65]. The algorithm typically starts with an empty solution or an initial state and iteratively adds components that maximize the immediate benefit.

For example, consider the Fractional Knapsack Problem, where the objective is to maximize the total value V of items placed in a knapsack without exceeding its capacity W. The Greedy Algorithm for this problem involves selecting items based on their value-to-weight ratio $\frac{v_i}{w_i}$. The algorithm starts by selecting the item with the highest ratio and continues adding items in decreasing order of their ratios until the knapsack is full. Mathematically, the greedy choice at each step is:

$$\text{Choose } i = \arg\max \left(\frac{v_i}{w_i} \right) \quad \text{such that } \sum_i w_i \leq W,$$

where v_i and w_i represent the value and weight of item i, respectively. Unlike the 0/1 Knapsack Problem, items can be divided, allowing fractional parts to be included in the solution.

While greedy algorithms are simple to implement and run efficiently, making them suitable for large problems where exact methods are impractical, they do not always produce optimal solutions. This limitation is particularly evident in problems where local optimality does not imply global optimality. For instance, in the TSP, a greedy approach that selects the shortest edge at each step may result in a poor overall tour. In summary, heuristics like the Nearest Neighbor and Greedy Algorithms provide efficient, often close-to-optimal solutions for combinatorial optimization problems, especially when the problem size makes exact methods infeasible. However, the lack of a guarantee of global optimality remains a key limitation, particularly as problem complexity increases.

1.2.3 Metaheuristics: Genetic Algorithms, Simulated Annealing, Ant-Colony Optimization

Metaheuristics are higher-level strategies that guide the search process to explore the solution space more effectively. Unlike heuristics, which are typically problem-specific, metaheuristics are designed to be more general and can be applied to a wide range of combinatorial optimization problems [18, 29, 32, 90, 156, 170, 174, 178, 186, 196, 231, 245]. They are particularly useful for solving large and complex problems where exact methods and simple heuristics are inadequate.

Genetic Algorithms (GA). Genetic Algorithms are inspired by the principles of natural selection and genetics [172]. These algorithms operate on a population of candidate solutions, referred to as individuals, and evolve the population over successive generations through the application of evolutionary operators such as selection, crossover (recombination), and mutation.

The process begins with the initialization of a random population of candidate solutions. Each individual represents a potential solution to the problem at hand. The selection phase evaluates the fitness of each individual based on a fitness function $f(s)$, where s represents the solution. Fitter individuals, those with higher fitness values, have a higher probability of being selected for reproduction. In the crossover phase, selected individuals (parents) are combined to produce offspring. For example, in the Traveling Salesman Problem, where each individual represents a possible tour, the crossover operation might involve swapping segments of two parent tours to create new offspring. Mathematically, if two parent tours are $p_1 = [A, B, C, D, E]$ and $p_2 = [E, D, C, B, A]$, a possible offspring might be $o_1 = [A, B, C, B, A]$.

The mutation phase introduces random changes to some individuals to maintain genetic diversity and prevent premature convergence to suboptimal solutions. In the TSP, this could involve swapping two cities in the tour or reversing the order of a segment. After crossover and mutation, the replacement step occurs, where the least fit individuals in the population are replaced by the offspring, thus creating a new generation. This process continues until a stopping criterion is met, such as a fixed number of generations or convergence to a solution. For example, in the TSP, the fitness function $f(s)$ might be defined as the inverse of the total tour length, $f(s) = \frac{1}{\text{total distance}}$, encouraging shorter tours. Although GAs are highly flexible and effective in exploring large and complex solution spaces, they require careful tuning of parameters like population size, crossover rate, and mutation rate. Moreover, GAs do not guarantee finding the global optimum, and their performance can vary depending on the problem and parameter settings.

Simulated Annealing (SA). Simulated Annealing is a metaheuristic inspired by the annealing process in metallurgy, where a material is slowly cooled to remove defects and reach a stable state [98]. In SA, the algorithm explores the solution space by accepting both improving and worsening solutions, with the probability of accepting worsening solutions decreasing over time.

The algorithm starts with an initial solution and an initial temperature T_0. At each iteration, a neighboring solution s' is generated by making a small change to the current solution s. For instance, in the TSP, a neighboring solution might be generated by swapping two cities in the current tour. The algorithm then evaluates the new solution. If the neighboring solution is better

(i.e., has a shorter tour length), it is accepted. Otherwise, it is accepted with a probability P given by:

$$P = \exp\left(\frac{f(s') - f(s)}{T}\right),$$

where $f(s)$ and $f(s')$ are the objective function values (e.g., total tour lengths) of the current and neighboring solutions, respectively, and T is the current temperature. Since $\exp(x)$ is always less than or equal to 1 for $x \leq 0$, the probability P is guaranteed to be between 0 and 1. This probability allows the algorithm to escape local optima by accepting worse solutions early in the process.

As the algorithm progresses, the temperature is gradually reduced according to a cooling schedule, typically $T = T_0 \times \alpha^k$, where α is a cooling factor (e.g., 0.95), and k is the iteration number. The likelihood of accepting worse solutions decreases as the temperature drops, allowing the algorithm to focus on refining the best solutions. SA is particularly effective at escaping local optima, making it suitable for problems with complex solution landscapes. However, its performance depends heavily on the cooling schedule and the choice of initial temperature. If these parameters are not carefully tuned, the algorithm may converge too quickly to a suboptimal solution or take too long to find an optimal one.

Ant Colony Optimization (ACO). Ant Colony Optimization is inspired by the foraging behavior of ants, which find the shortest paths to food sources by laying down pheromones and following pheromone trails [37]. In ACO, artificial ants construct solutions by moving through the problem space and depositing pheromones on promising paths, which guide subsequent ants.

The process begins with initializing a population of ants, where each ant represents a potential solution. The pheromone levels on all paths are also initialized. As each ant constructs a solution, it moves from one decision point to the next, guided by pheromone levels τ_{ij} and a heuristic function η_{ij}, which might represent the desirability of choosing a particular path. The probability P_{ij} that an ant will move from node i to node j is given by:

$$P_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{k \in \text{allowed}} \tau_{ik}^{\alpha} \cdot \eta_{ik}^{\beta}},$$

where α and β are parameters that control the influence of pheromone and heuristic information, respectively.

After all ants have constructed their solutions, the pheromone levels on the paths are updated. Paths that are part of good solutions receive more pheromones, which increases the likelihood that future ants will follow them. This is known as the pheromone update. Additionally, pheromone evaporation is applied to reduce the influence of older paths, preventing convergence to a suboptimal solution. In the TSP, each ant represents a potential tour, and as ants traverse the cities, they deposit pheromones on the paths they take. Shorter tours receive more pheromones, guiding subsequent ants toward these better paths. Over time, the collective behavior of the ants converges on the shortest path.

ACO is robust and adaptable, making it effective at finding good solutions for combinatorial optimization problems with complex search spaces. However, ACO can be computationally intensive, especially for large problems with many decision points. The algorithm's performance also

depends on the careful tuning of parameters, such as pheromone decay rate and the influence of the heuristic function. In summary, metaheuristics like Genetic Algorithms, Simulated Annealing, and Ant Colony Optimization provide powerful tools for solving large and complex combinatorial optimization problems. While they do not guarantee finding the global optimum, they are highly effective at exploring vast solution spaces and escaping local optima, making them indispensable for practical optimization tasks.

1.3 Role of Machine Learning in Combinatorial Optimization

Machine learning (ML) has emerged as a powerful tool for addressing the challenges associated with solving combinatorial optimization problems [13, 24, 38, 41, 47, 71, 86, 94, 104, 129, 155, 219, 223, 229, 233]. Traditional methods, such as exact algorithms, heuristics, and metaheuristics, have limitations, particularly when dealing with large, complex, or dynamic problems. Machine learning offers a novel approach by leveraging data-driven techniques to enhance optimization processes, improve solution quality, and reduce computational costs. This section explores the role of machine learning in combinatorial optimization, starting with an introduction to the various machine learning techniques applied to CPs.

1.3.1 Introduction to Machine Learning Techniques in Combinatorial Optimization Problems

Machine learning techniques have been increasingly integrated into combinatorial optimization to overcome the limitations of traditional methods [21–23, 40, 50, 58, 80, 87, 102, 105, 115, 121, 134, 164, 176, 180, 191, 198, 205, 220]. These techniques can be broadly categorized into several approaches, each with its own strengths and applications in solving CPs.

Supervised Learning. In supervised learning, a model is trained on a labeled dataset where the input-output pairs are known. The model learns to map inputs to outputs, which can then be used to predict outcomes for new, unseen inputs [55]. For instance, in the Knapsack Problem, a supervised learning model might learn to predict which items are most likely to be included in an optimal solution based on features like value-to-weight ratio, size, and availability [166]. The prediction $\hat{y} = f(x)$ is based on minimizing a loss function, such as the mean squared error:

$$\text{Loss} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2,$$

where y_i is the true output, $\hat{y}_i$ is the predicted output, and n is the number of training examples. Neural networks, decision trees, and support vector machines are common supervised learning techniques used in combinatorial optimization. These models can be trained to approximate solution quality or guide the search process in heuristics. For example, in TSP, a neural network could be trained to estimate the quality of different tour configurations, thereby helping to prioritize certain paths during the search.

Reinforcement Learning (RL). Reinforcement learning involves training an agent to make decisions by interacting with an environment. The agent learns to maximize cumulative rewards by taking actions that lead to desired outcomes [182]. RL is particularly well-suited for combinatorial optimization problems, where the objective is to find a sequence of decisions that optimizes a given criterion. The RL process can be formalized by defining a state space S, an action space A, a reward function $R(s, a)$, and a policy $\pi(a|s)$ that maps states to actions. In RL-based optimization, the agent learns a policy that maps states (representations of partial solutions) to actions (decisions) that lead to the construction of complete solutions. RL has been successfully applied to various CPs, such as the Traveling Salesman Problem, Vehicle Routing Problem, and Job Shop Scheduling Problem. For example, in the TSP, an RL agent might learn to construct a tour by sequentially selecting the next city based on the current partial tour and expected future rewards.

Unsupervised Learning. Unsupervised learning involves finding patterns or structures in data without the use of labeled outputs. The goal is to learn representations of the data that can be used for tasks like clustering or dimensionality reduction [14]. In combinatorial optimization, unsupervised learning can be used to identify patterns or structures in the solution space, which can then inform the design of optimization algorithms. For instance, clustering techniques can group similar problem instances, allowing for the development of specialized heuristics tailored to each cluster. Techniques such as k-means clustering can be applied to group problem instances based on certain features, thereby reducing the complexity of the optimization process. The k-means algorithm seeks to minimize the within-cluster variance:

$$\min \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2,$$

where C_i is the set of points in cluster i, and μ_i is the centroid of cluster i. Similarly, dimensionality reduction techniques like Principal Component Analysis (PCA) can be used to simplify the solution space, making it easier to explore and optimize. By reducing the dimensionality of the problem, PCA helps focus the search on the most critical variables, potentially improving the efficiency of the optimization algorithm.

Examples and Limitations. In supervised learning, a neural network might be trained to predict the optimal subset of items for a Knapsack Problem based on historical data [92]. However, the effectiveness of this approach depends on the quality and quantity of the labeled data available for training, which can be a limitation in some cases. In reinforcement learning, an agent could be trained to solve the VRP by learning a policy that balances minimizing travel distance with meeting customer delivery windows. While RL techniques can be powerful, they often require extensive training and can be sensitive to hyperparameters like learning rates and exploration strategies. Unsupervised learning might be used to identify clusters of similar TSP instances, allowing for the development of specialized heuristics for each cluster. However, the challenge lies in the interpretability and applicability of the patterns found by unsupervised learning to actual optimization tasks.

1.3.2 Neural Combinatorial Optimization (NCO)

Neural Combinatorial Optimization is an innovative approach that combines deep learning, particularly neural networks, with combinatorial optimization techniques [6, 20, 32, 44, 53, 68, 83, 95, 116, 123, 136, 150, 154, 158, 163, 173, 201, 216, 227, 232, 238, 240, 249, 253]. The goal of NCO is to learn heuristics or policies that can generate high-quality solutions to combinatorial optimization problems directly from data. Unlike traditional methods that rely on predefined rules or exhaustive searches, NCO leverages the learning capabilities of neural networks to discover patterns and strategies that can be generalized to new instances of the problem. NCO typically involves training a neural network model to solve a specific combinatorial problem. The architecture of the network is carefully designed to capture the problem's structure, and the model is trained using supervised learning, reinforcement learning, or a combination of both. This allows the network to learn not only from direct examples of optimal solutions but also from the exploration of the solution space.

Pointer Networks. One of the foundational architectures in NCO is the Pointer Network, introduced by [198]. Pointer Networks are a type of sequence-to-sequence model where the output is a sequence of pointers to the input elements, rather than the input elements themselves. This is particularly useful for problems like the Traveling Salesman Problem, where the output is a permutation of the input sequence. Let $x = (x_1, x_2, \ldots, x_n)$ be the input sequence (e.g., cities in TSP), and let $y = (y_1, y_2, \ldots, y_n)$ be the output sequence, where each y_i is an index into x. The probability of generating the output sequence y is modeled as:

$$P(y|x) = \prod_{i=1}^{n} P(y_i|y_1, y_2, \ldots, y_{i-1}, x),$$

where the probability $P(y_i|y_1, \ldots, y_{i-1}, x)$ is computed using attention mechanisms.

In the TSP, a Pointer Network is trained to output the sequence of cities that minimizes the tour length [198]. During training, the model learns to focus on the most relevant parts of the input sequence (e.g., the nearest unvisited cities) to generate a high-quality tour. NCO methods can generalize well across different instances of a problem and can produce solutions very quickly once trained. They are particularly effective for large-scale problems where traditional optimization methods struggle. NCO models require large amounts of training data and computational resources. Additionally, the quality of the solutions depends heavily on the quality of the training data and the effectiveness of the neural network architecture.

1.3.3 Reinforcement Learning (RL) Techniques in CPs

Reinforcement Learning (RL) is a machine learning paradigm where an agent learns to make decisions by interacting with an environment and receiving feedback in the form of rewards. In the context of combinatorial optimization, RL is used to learn policies that construct solutions to optimization problems through a sequence of decisions. The optimization problem in RL is modeled as a Markov Decision Process (MDP), which is defined by a tuple (S, A, P, R):

- S represents the set of states, which correspond to partial solutions or configurations of the problem. For instance, in the Vehicle Routing Problem, a state could represent the current location of a vehicle and the customers that have already been served.
- A represents the set of actions, which correspond to possible decisions that can be made at each state. In the VRP, an action could be selecting the next customer to visit.
- $P(s'|s, a)$ is the state transition probability, defining how the state changes after taking action a in state s. This models the environment's dynamics, such as how the vehicle's state changes after serving a customer.
- $R(s, a)$ is the reward function, defining the immediate reward for taking action a in state s. The reward could be, for example, the negative travel distance in VRP, where the goal is to minimize total distance.

The objective of the RL agent is to learn a policy $\pi(a|s)$ that maximizes the expected cumulative reward over time. This is mathematically represented as:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\right],$$

where γ is a discount factor that balances immediate and future rewards, with T representing the time horizon.

Q-Learning. Q-Learning is a model-free RL algorithm where the agent learns the value of taking a particular action in a given state, known as the Q-value [211]. The agent updates the Q-values iteratively using the Bellman equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\left[R(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a)\right],$$

where α is the learning rate. This equation adjusts the current Q-value based on the observed reward and the estimated value of the best future action. Q-Learning does not require a model of the environment, making it suitable for problems where the environment's dynamics are unknown.

Policy Gradient Methods. Policy gradient methods directly optimize the policy $\pi_\theta(a|s)$ by adjusting the parameters θ to maximize the expected reward [183]. The gradient of the expected reward with respect to the policy parameters is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t\right],$$

where G_t is the cumulative reward from time step t. This approach allows for the direct improvement of the policy by following the gradient of the expected reward, making it particularly effective in problems where the action space is continuous or where Q-learning struggles.

Actor-Critic Methods. Actor-Critic methods combine the strengths of policy gradient and value-based approaches [101]. The actor updates the policy parameters θ using policy gradient, while the critic estimates the value function $V(s)$ to reduce the variance of the gradient estimate. The critic evaluates the goodness of the actions taken by the actor, providing feedback that helps stabilize and speed up learning.

Example. In the VRP, an RL agent might be tasked with learning a policy for deciding which customer to visit next, given the current state of the vehicle, such as its location and remaining capacity [147]. The reward function could be designed to minimize the total travel distance, such that each decision that leads to a shorter route is rewarded. As the agent interacts with the environment (e.g., simulating deliveries), it learns to prioritize actions that contribute to a more efficient overall route.

Advantages and Limitations. Reinforcement learning is highly flexible and can be adapted to a wide range of combinatorial problems [105]. It is particularly useful in dynamic environments where the problem parameters may change over time, such as real-time logistics or adaptive scheduling. However, RL often requires a large number of interactions with the environment to learn an effective policy, which can be computationally expensive. Additionally, designing an appropriate reward function that accurately reflects the objectives of the optimization problem can be challenging. If the reward function is not well-designed, the agent may learn suboptimal behaviors that do not align with the desired outcomes.

1.3.4 Curriculum Learning (CL) Strategies

Curriculum Learning is a training strategy in machine learning where a model is trained on progressively more difficult tasks, much like how humans learn by starting with simpler concepts and gradually moving on to more complex ones [11]. In combinatorial optimization, CL is used to enhance the efficiency and effectiveness of learning-based optimization methods by ensuring that the model develops a strong foundation before tackling more challenging problems.

The curriculum is designed as a sequence of tasks $T_1, T_2, \ldots, T_n$, where each task T_i is more complex than the preceding task T_{i-1}. The model is initially trained on the simpler tasks and is gradually introduced to more difficult ones. The idea is that by mastering simpler problems first, the model builds up the necessary knowledge and skills to handle more complex instances. For example, in the Traveling Salesman Problem, the sequence of tasks might start with small instances involving just a few cities and gradually increase in complexity by adding more cities. Let T_i represent a TSP instance with n_i cities, where $n_1 < n_2 < \cdots < n_n$. The training process might involve optimizing the model's performance on each task until it exceeds a predefined threshold ϵ:

$$\mathbb{E}[L(T_i)] < \epsilon,$$

where $L(T_i)$ is the loss function for task $L(T_i)$. Once the model's performance on task T_i meets this threshold, it moves on to the next task T_{i+1}. The difficulty of a task can be quantified in various ways, depending on the specific combinatorial problem. For instance, in TSP, the difficulty

might be directly proportional to the number of cities n in the problem instance. In another context, like the Job Shop Scheduling Problem (JSP), difficulty could be measured by the number of jobs and machines or the complexity of scheduling constraints.

Example. Consider the Job Shop Scheduling Problem. A curriculum learning strategy might start by training the model on simpler instances with fewer jobs and machines, allowing the model to learn basic scheduling strategies [75]. As the model's performance improves, the curriculum introduces more complex instances with an increased number of jobs and tighter scheduling constraints. For example, the initial task T_1 might involve scheduling 2 jobs across 2 machines, while a more complex task T_n could involve 10 jobs across 5 machines with varying processing times and constraints.

Advantages and Limitations. Curriculum Learning can significantly accelerate the training process by ensuring that the model is not overwhelmed by complexity early on. By gradually increasing the difficulty of tasks, the model is more likely to learn robust features that generalize well to more complex instances [11]. This gradual learning process also helps prevent overfitting to simpler tasks and encourages the model to develop a deeper understanding of the problem domain. However, designing an effective curriculum requires careful consideration of the problem's characteristics and the model's learning dynamics. An improperly designed curriculum may either progress too quickly, overwhelming the model, or too slowly, failing to challenge the model sufficiently. In the worst case, the model might not gain the intended benefits, leading to suboptimal performance on more complex tasks.

1.3.5 Generative Models

Generative models are a class of machine learning models that are capable of generating new data samples from the learned distribution of a dataset. In the context of combinatorial optimization, generative models are employed to generate high-quality solutions to optimization problems by learning the underlying structure of the solution space. By understanding this structure, generative models can produce novel and diverse solutions that can be further refined or used as starting points for more traditional optimization techniques.

Generative Adversarial Networks (GANs). One popular type of generative model is the Generative Adversarial Network, which consists of two neural networks—a generator G and a discriminator D that are trained together in a competitive manner [56]. The generator G tries to produce realistic samples that fool the discriminator, while the discriminator D tries to distinguish between real and generated samples. The objective of a GAN is formulated as a minimax game:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))],$$

where z is a random noise vector sampled from a prior distribution $p_z(z)$, and $p_{\text{data}}(x)$ is the distribution of real data. The generator $G(z)$ aims to transform the noise z into a data sample that

resembles the real data distribution, while the discriminator $D(x)$ attempts to classify whether a given sample x is real or generated. In combinatorial optimization, GANs can be used to generate potential solutions to problems such as the Knapsack Problem [111]. For example, the generator could produce candidate item selections, while the discriminator evaluates whether the selection is likely to be part of an optimal solution. The generated solutions can then be refined using additional optimization techniques or heuristics to ensure feasibility and improve quality.

Variational Autoencoders (VAEs). Another type of generative model is the Variational Autoencoder, which learns a probabilistic mapping from data to a latent space [97]. The VAE consists of two main components: an encoder and a decoder. The encoder maps the input data x to a latent distribution $q(z|x)$, while the decoder reconstructs the data x from a latent representation z. The training objective for a VAE involves maximizing the likelihood of the data while regularizing the latent space to follow a prior distribution, typically a standard normal distribution $p(z)$:

$$\text{Loss} = \mathbb{E}_{q(z|x)}[\log p(x|z)] - D_{\text{KL}}(q(z|x)||p(z)),$$

where D_{KL} is the Kullback-Leibler divergence, a measure of how much the learned latent distribution $q(z|x)$ deviates from the prior $p(z)$. This regularization encourages the VAE to learn a smooth and structured latent space, from which new data samples can be generated by sampling from the latent space and decoding the samples back into the original data space. In the context of the Traveling Salesman Problem, a VAE could be used to learn a latent representation of optimal tours [69]. The encoder compresses a given tour into a latent vector z, which captures the essential features of the tour. New tours can then be generated by sampling from this latent space and decoding the samples into sequences of city visits. This approach allows for the exploration of the solution space in a more structured manner, potentially leading to the discovery of novel and high-quality tours.

However, training generative models can be challenging, particularly in ensuring that the generated solutions are both feasible and optimal. For instance, a GAN trained on combinatorial problems might produce solutions that are realistic but not necessarily optimal or even feasible under the problem's constraints. Additionally, the quality of the generated solutions is highly dependent on the quality and diversity of the training data. If the training data does not adequately represent the full complexity of the problem, the generative model may struggle to generalize, leading to suboptimal solutions.

1.3.6 Large Language Models (LLMs)

Large Language Models such as GPT-3 and GPT-4, originally developed for natural language processing (NLP) tasks, have demonstrated remarkable capabilities in understanding and generating complex patterns in data. Beyond NLP, these models can be adapted to solve combinatorial optimization problems. LLMs can be prompted in specific ways to generate solutions to optimization problems or to assist in the exploration of solution spaces.

LLMs are typically based on transformer architectures, which are well-suited for capturing dependencies in sequential data [194]. The architecture consists of self-attention mechanisms that

allow the model to weigh the importance of different elements in a sequence when generating outputs. The self-attention mechanism is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V,$$

where Q (queries), K (keys), and V (values) are linear projections of the input, and d_k is the dimensionality of the keys. This mechanism enables the model to focus on relevant parts of the input sequence, which is crucial when dealing with combinatorial problems where certain decisions depend on the relationships between various elements.

One approach to use LLMs for CPs is prompting, where the model is provided with a problem description and some context or partial solution as input. The model then generates the remaining part of the solution or suggests possible next steps [128]. This method leverages the model's pre-trained knowledge and ability to generalize across various tasks. The output of the LLM is generated based on the provided prompt:

$$\text{Output} = \text{LLM}(\text{prompt}),$$

where the prompt includes the problem description and any additional context. For example, in the TSP, an LLM could be prompted with the coordinates of the cities and a partial tour, and the model would generate the continuation of the tour that minimizes the total distance.

Advantages and Limitations. The primary advantage of LLMs lies in their flexibility and adaptability to a wide range of combinatorial optimization problems. They are particularly effective when integrated into workflows that require both problem-solving and the generation of structured data. However, the use of LLMs in combinatorial optimization also comes with challenges. These models require large amounts of computational resources. Furthermore, LLMs are not inherently designed for optimization tasks, so their performance may vary depending on how well the problem is formulated as a language task. Ensuring that the solutions generated by LLMs are both feasible and optimal can be challenging, often requiring careful prompt design and possibly additional post-processing steps to refine the outputs.

1.3.7 Representation of Problems as Markov Decision Processes (MDPs)

To apply reinforcement learning (RL) techniques to combinatorial optimization problems, it is common to represent these problems as Markov Decision Processes (MDPs). An MDP provides a mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision-maker.

Representation of TSP as an MDP [147]. In the Traveling Salesman Problem, the objective is to find the shortest possible route that visits each city exactly once and returns to the starting point. To represent TSP as an MDP:

- States S: A state in the TSP can be represented as a sequence of cities that have been visited so far. For example, if the tour has visited cities $\{v_1, v_2, \ldots, v_k\}$, the current state s_t is $(v_1, v_2, \ldots, v_k)$.
- Actions A: The actions correspond to the selection of the next city to visit. From state s_t, the set of possible actions A_t includes all unvisited cities.
- Transition Probability $P(s'|s, a)$: In the deterministic case, the transition probability is 1 if the next state s_{t+1} is the sequence $(v_1, v_2, \ldots, v_k, a)$, where a is the city chosen by the action. The probability is 0 for all other transitions.
- Reward $R(s, a)$: The reward function can be defined as the negative of the distance to the next city: $R(s_t, a) = -d_{v_k,a}$, where $d_{v_k,a}$ is the distance between the last visited city v_k and the city chosen by action a.
- The goal is to find a policy $\pi(s)$ that maximizes the cumulative reward, which corresponds to minimizing the total tour length:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\right],$$

  where γ is the discount factor that balances immediate and future rewards, and T is the total number of steps in the tour.

Representation of VRP as an MDP [76]. In the Vehicle Routing Problem, the objective is to optimize the routes of multiple vehicles delivering goods to a set of customers while minimizing total travel times. To represent VRP as an MDP:

- States S: A state in VRP can be represented as the current location of all vehicles and the set of customers who have been served. For example, if a vehicle has visited customers $\{v_1, v_2, \ldots, v_k\}$ and is currently at v_k, the state includes this information for all vehicles.
- Actions A: The actions correspond to the next customer that each vehicle should visit. The action set depends on the vehicle's current location and remaining capacity.
- Transition Probability $P(s'|s, a)$: The transition probability depends on the selected customer and the vehicle's capacity. The new state reflects the updated location of the vehicle and the reduced demand at the visited customer.
- Reward $R(s, a)$: The reward can be defined as the negative of the travel cost associated with moving from the current location to the next customer:

$$R(s, a) = -c_{v_k,a},$$

  where $c_{v_k,a}$ is the travel cost between the current location v_k and the selected customer a.
- Objective: The goal is to optimize the routes for all vehicles such that the total travel cost is minimized while ensuring that all customer demands are met:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\right].$$

Representation of JSP as an MDP [75]. In the Job Shop Scheduling Problem, the objective is to schedule jobs on machines such that the total time required (makespan) is minimized. To represent JSP as an MDP:

- States S: A state in JSP can be represented by the current status of all jobs, including which operations have been completed and which are in progress on specific machines. For example, if a job j_i is halfway through an operation on machine m_j, this information is part of the current state.
- Actions A: The actions correspond to the scheduling decisions, such as assigning the next operation of a job to a specific machine.
- Transition Probability $P(s'|s, a)$: The transition reflects the completion of an operation and the assignment of the next operation. The next state indicates the updated job and machine statuses.
- Reward $R(s, a)$: The reward function can be defined as the negative increase in makespan, where makespan is the total time taken to complete all jobs:

  $$R(s, a) = -\text{Makespan Increase}(s, a).$$

  Alternatively, the reward could be negative to represent penalties such as delays or machine idle times.
- Objective: The goal is to find a scheduling policy that minimizes the makespan, ensuring that all jobs are completed as efficiently as possible.

  $$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\right].$$

These representations of combinatorial problems as MDPs enable the application of reinforcement learning techniques to solve them effectively. For instance, in the TSP, the MDP framework allows an RL agent to learn a policy that sequences city visits to minimize travel distance. Similarly, in VRP, an RL agent can optimize vehicle routes by learning from the outcomes of different routing decisions, ensuring cost-efficient deliveries. However, there are limitations to this approach. Representing combinatorial problems as MDPs often requires simplifying assumptions, such as deterministic transitions or predefined state spaces, which may not fully capture the complexity of real-world problems. Additionally, the size of the state space in these problems can be enormous, making it challenging to explore all possible states and actions effectively. Finally, designing reward functions that accurately reflect the optimization objective can be difficult, as inappropriate rewards may lead the agent to suboptimal solutions.

1.4 Overcoming Challenges in Combinatorial Optimization Research

Limitations of Traditional Algorithms in Combinatorial Optimization. Traditional algorithms, such as Branch-and-Bound, the Clarke-Wright heuristic, and metaheuristics like Ant-

Colony Optimization (ACO) and Tabu Search, have been widely used to solve combinatorial problems. However, these methods often fall short, particularly in scenarios involving stochastic elements or correlated variables. The performance of these methods can be measured by the optimality gap, which is the difference between the solution provided by an algorithm and the true optimal solution. Mathematically, the optimality gap can be expressed as:

$$\text{Optimality Gap} = \frac{|\text{Model Solution} - \text{Optimal Solution}|}{\text{Optimal Solution}} \times 100\%.$$

For instance, in the Job-shop Scheduling Problem, this gap can be as high as 38.43% on complex instances, indicating that traditional methods may generate solutions that are far from optimal. This suboptimality is particularly problematic in practical applications where near-optimal solutions are crucial. Chapter 2 of this thesis will address this challenge by exploring advanced methodologies aimed at reducing the optimality gap and enhancing solution quality.

Enhancing Generalization in Machine Learning Models for Combinatorial Problems. Generalization refers to the ability of a model to apply what it has learned from specific problem instances to new, unseen instances. In combinatorial optimization, many models, particularly those based on machine learning, are trained on specific instances and may struggle to generalize to larger or more complex instances. The performance of these models on new instances can be assessed by comparing the generalization error:

$$\text{Generalization Error} = \frac{|\text{Performance on New Instance} - \text{Performance on Training Instance}|}{\text{Performance on Training Instance}} \times 100\%.$$

A high generalization error indicates that the model's learned policies do not scale well, leading to inefficiencies and suboptimal solutions when applied to different or larger instances. Chapter 2 of this thesis will focus on developing methodologies to enhance generalization, ensuring that models trained on smaller instances can effectively tackle larger and more complex problems.

Addressing the Research Gap: Stochastic Variables in Combinatorial Optimization. Despite the significant advances in solving combinatorial optimization problems, there remains a notable lack of research focused on combinatorial problems that involve stochastic variables. Many existing studies and methodologies assume deterministic inputs, where all variables are known in advance and do not change. However, real-world problems, such as the Vehicle Routing Problem, often involve stochastic variables—uncertain factors like customer demands, travel times, and other dynamic elements that are revealed only as the solution is being implemented. The complexity introduced by these stochastic variables is significant, as it requires models to make decisions under uncertainty, often leading to solutions that must adapt in real-time. The impact of this lack of research is particularly evident in the field of logistics and transportation, where problems like the Stochastic Vehicle Routing Problem (SVRP) with correlated demands or time windows are common.

Addressing these gaps requires developing new methodologies that can effectively handle the uncertainty and variability inherent in these problems. This thesis will contribute to this area by focusing on the VRP task across Chapters 3, 4, and 5, where we develop and evaluate advanced

approaches to solving combinatorial problems under stochastic conditions. These chapters will cover various aspects, including handling correlated demands, optimizing under time constraints, and enhancing the scalability and adaptability of solutions in stochastic environments.

Modeling and Leveraging Interdependencies in Stochastic Combinatorial Problems. In many real-world combinatorial problems, variables are not independent but are instead correlated, adding another layer of complexity to the optimization process. For instance, in the Stochastic Vehicle Routing Problem, both the demands at each stop and the travel times can be correlated, which complicates the modeling and solution process. The impact of correlated variables can be represented by the covariance matrix Σ, which captures the dependencies between stochastic variables:

$$\Sigma_{ij} = \text{Cov}(X_i, X_j) = \mathbb{E}[(X_i - \mu_i)(X_j - \mu_j)],$$

where X_i and X_j are stochastic variables (such as demand or travel time at different locations), and μ_i and μ_j are their expected values. Accurate modeling of these interdependencies is crucial for obtaining optimal solutions. Chapters 3 through 5 will address these interdependencies, focusing on methods to capture and leverage correlations in stochastic variables to improve optimization outcomes.

Harnessing Large Language Models for Complex Combinatorial Optimization. Large Language Models have demonstrated remarkable capabilities in various reasoning and problem-solving tasks, opening up new possibilities for their application in combinatorial optimization problems. LLMs, with their vast pre-trained knowledge and ability to generate context-aware outputs, have the potential to tackle complex CPs by reasoning through various solution pathways. This potential could revolutionize how we approach problems like the Traveling Salesman Problem, Vehicle Routing Problem, and Job-shop Scheduling Problem.

However, despite their potential, there is currently a significant lack of research focused on leveraging LLMs for solving combinatorial problems. Most existing work on LLMs has concentrated on natural language processing tasks, with relatively few studies exploring their application to CPs. This gap in the literature means that the challenges specific to using LLMs for combinatorial optimization have not been thoroughly investigated or addressed.

One of the primary challenges in applying LLMs to combinatorial problems is the need for complex prompt engineering. To effectively guide an LLM towards generating useful solutions for CPs, carefully crafted prompts are required. These prompts must encapsulate the problem's constraints, objectives, and specific requirements in a way that the LLM can interpret and process effectively. For simple CPs, this may involve straightforward instructions, but as the complexity of the problem increases, so too does the difficulty of designing appropriate prompts.

Creating prompts that can lead to high-quality solutions for complex CPs is not trivial. It requires a deep understanding of both the problem at hand and the inner workings of LLMs. Poorly designed prompts can result in suboptimal or irrelevant solutions, and refining these prompts often involves an iterative process of trial and error. Moreover, the performance of LLMs in generating solutions for CPs can be inconsistent, particularly as problem size and complexity increase, highlighting the need for more research into robust prompt engineering techniques.

Chapter 6 of this thesis will explore the untapped potential of LLMs in solving combinatorial problems, with a focus on understanding the challenges of prompt engineering for complex CP tasks. This chapter will also investigate strategies to overcome these challenges, aiming to harness the full potential of LLMs in this domain. By addressing these challenges, we hope to pave the way for more effective and efficient use of LLMs in solving a wide range of combinatorial optimization problems.

1.5 Scope and Contributions of This Thesis

This thesis explores the integration of machine learning techniques with traditional combinatorial optimization methods to address the challenges inherent in solving complex combinatorial optimization problems. By leveraging advancements in machine learning, particularly in reinforcement learning, neural combinatorial optimization, and hybrid approaches, this research aims to develop innovative solutions that enhance the efficiency, scalability, and adaptability of optimization processes. This section outlines the specific objectives and research questions that guide the thesis, provides an overview of the five key papers that form the foundation of this work, and presents the structure of the thesis.

1.5.1 Objectives and Research Questions

The primary objective of this thesis is to investigate how machine learning can be effectively integrated with combinatorial optimization techniques to solve complex and large-scale combinatorial optimization problems more efficiently and effectively. This overarching goal is supported by several specific objectives, each addressing different aspects of the intersection between machine learning and combinatorial optimization.

- Research Question 1: What methodologies can enhance the generalization ability of machine learning models in combinatorial optimization, allowing them to effectively tackle larger and more complex problem instances, particularly in the context of the Job Shop Scheduling Problem? This research question explores how to improve the generalization of machine learning models trained on specific problem instances, enabling them to perform well on unseen or more complex combinatorial problems.
- Research Question 2: How can we develop effective models and algorithms for combinatorial problems that involve stochastic variables and dynamic conditions, particularly in the context of the Vehicle Routing Problem? This question aims to address the research gap in combinatorial problems involving stochastic variables, focusing on developing approaches that can handle uncertain elements like customer demands or travel times in real-time.
- Research Question 3: How can interdependencies and correlations between stochastic variables in combinatorial problems be accurately modeled and leveraged to improve optimization outcomes? This question focuses on understanding and addressing the complexity introduced by correlated stochastic variables in combinatorial problems and developing methods to incorporate these dependencies into optimization models effectively.

- Research Question 4: What are the most effective techniques for prompt engineering to harness the full potential of Large Language Models in solving complex combinatorial optimization problems? This research question aims to investigate how to design and refine prompt engineering strategies that enable LLMs to generate high-quality solutions for complex combinatorial tasks, overcoming the limitations seen in current approaches.

The answers to these research questions will provide insights into the potential and limitations of machine learning in combinatorial optimization, contributing to both theoretical advancements and practical applications in the field.

1.5.2 Overview of the Five Key Papers and Their Contributions to the Field

This thesis builds upon five research papers, each addressing one of the critical research questions posed in the previous section. These papers collectively provide insights into the challenges of solving combinatorial optimization problems in complex scenarios, particularly when dealing with stochastic variables, dynamic conditions, and generalization challenges. In this chapter, we provide an overview of each of these papers, focusing on their contributions to the field and the methodologies they employed to tackle the key challenges in combinatorial optimization.

- Paper 1: On the Study of Curriculum Learning for Inferring Dispatching Policies on the Job Shop Scheduling Problem [75]
  - Idea: This paper introduces the Reinforced Adaptive Staircase Curriculum Learning (RASCL) method to solve the Job Shop Scheduling Problem using reinforcement learning. The paper develops an autoregressive RL model that is size-agnostic and can generalize across different JSP sizes. This model iteratively constructs scheduling solutions by learning a dispatching policy that selects the next operation to be scheduled based on the current system state. The RASCL strategy dynamically adjusts the difficulty of learning tasks during training by revisiting challenging instances.
  - Contribution: This method allows the RL model to gradually learn from simpler instances to more complex ones, leading to significant improvements in performance. RASCL reduces the average optimality gap from 19.35% to 10.46% on Taillard's benchmark instances and from 38.43% to 18.85% on Demirkol's instances, demonstrating substantial improvements over traditional methods.
- Paper 2: Reinforcement Learning Approach to Stochastic Vehicle Routing Problem with Correlated Demands [76]
  - Idea: The paper addresses the Vehicle Routing Problem with Stochastic Demands, introducing the concept of correlated demands, where customer demands are influenced by external factors. The paper models demand correlations using observable variables, providing more accurate demand predictions than the i.i.d. assumption commonly used in traditional VRP models. An RL framework is proposed to learn routing policies by exploiting demand correlations, allowing dynamic route adjustments as new demand data becomes available.

  - Contribution: The RL-based model improves routing efficiency by 3.26% over baseline approaches, with performance gains increasing for larger problem sizes. The model demonstrates robustness to variations in vehicle capacity, demand types, and levels of correlation, making it adaptable to real-world logistics problems.

- Paper 3: Reinforcement Learning for Solving Stochastic Vehicle Routing Problem [77]
  - Idea: The paper introduces a novel RL framework designed to handle the stochasticity in both demands and travel times. The framework utilizes external variables such as weather conditions to make more accurate predictions and optimize vehicle routes in real-time. The paper explores both a priori approaches, where routes are pre-planned, and reoptimization approaches, where routes are updated dynamically as new information becomes available. The RL model excels in both scenarios, showcasing its flexibility.
  - Contribution: The RL model demonstrates a 3.43% improvement in performance over classical methods such as Ant Colony Optimization and Clarke-Wright heuristics. This performance gain is especially evident in scenarios with stochastic travel times. The RL framework provides a scalable and adaptable solution to SVRP, making it an effective tool for solving real-world logistics problems characterized by demand uncertainty and fluctuating travel times.

- Paper 4: Reinforcement Learning for Solving Stochastic Vehicle Routing Problem with Time Windows [78]
  - Idea: The paper introduces a reinforcement learning model that handles the additional complexity of time windows while also considering stochastic factors such as demands and travel times. This makes the model more applicable to real-world logistics challenges.
  - Contribution: The proposed model shows robustness across varying levels of stochasticity and problem sizes, demonstrating its ability to adapt to different logistical constraints, such as fleet size and customer base. The model achieves a 1.73% reduction in travel times compared to ACO, making it one of the most efficient approaches for solving VRP with time windows.

- Paper 5: Self-Guiding Exploration for Combinatorial Problems [73]
  - Idea: The paper explores the application of Large Language Models to combinatorial problems such as the Traveling Salesman Problem, Vehicle Routing Problem, and Job Scheduling Problem. It introduces the Self-Guiding Exploration (SGE) strategy, designed to enhance the performance of LLMs in solving CPs. SGE autonomously generates multiple thought trajectories for each combinatorial task, decomposes them into manageable subtasks, and refines the results to achieve optimal solutions.
  - Contribution: SGE improves CP optimization performance by 27.84% over existing prompting strategies like Chain-of-Thought, Decomposition, and Self-Refinement. This improvement is particularly evident in hard problems such as the Job Scheduling Problem, where the optimality gap is reduced by 34.85%. In addition to combinatorial

optimization, SGE demonstrates a 2.46% higher accuracy in reasoning tasks such as arithmetic, commonsense, and symbolic reasoning, showcasing its broad applicability.

These five papers collectively make contributions to the fields of machine learning and combinatorial optimization. Each paper introduces novel methodologies in curriculum learning, reinforcement learning, and prompting strategies, to tackle complex problems like JSP, VRP, and their stochastic variants. While the results demonstrate notable improvements over traditional approaches, the methodologies are part of an ongoing effort to enhance the efficiency, scalability, and flexibility of solving NP-hard problems.

1.5.3 Structure of the Thesis

This thesis is organized into five main chapters, each structured to provide a comprehensive understanding of the research conducted in the field of machine learning applied to combinatorial optimization problems. The chapters build upon one another, starting with an introduction to the field, followed by detailed explorations of learning algorithms for different combinatorial problems, and culminating in a discussion of large language models and their application to such problems. The final chapter provides conclusions and suggestions for future research directions.

- Chapter 1: Introduction
  - The first chapter serves as the foundation of the thesis by introducing combinatorial optimization problems and their significance.
  - The chapter also discusses traditional approaches, including exact methods, heuristics, and metaheuristics, and highlights their limitations in solving CP tasks.
  - The role of machine learning in overcoming these limitations is explored, with a focus on reinforcement learning, neural combinatorial optimization, curriculum learning, and the use of large language models.
  - The chapter concludes by outlining the scope and contributions of the thesis, presenting the research questions, and providing an overview of the five key papers that form the basis of this work.
- Chapter 2: Learning Algorithm for Job-Shop Scheduling Problem
  - This chapter presents a detailed investigation into the application of RL and curriculum learning for solving the Job-Shop Scheduling Problem.
  - The chapter discusses the development of an RL model that learns dispatching policies to optimize job scheduling.
  - A novel curriculum learning strategy, Reinforced Adaptive Staircase Curriculum Learning, is introduced to improve the training efficiency of the RL model.
  - The chapter includes detailed descriptions of the architecture, inference strategies, and experimental results that demonstrate significant performance improvements over traditional methods.

- Chapter 3: Learning Algorithm for Stochastic Vehicle Routing Problem
  - This chapter provides a comprehensive exploration of the Stochastic Vehicle Routing Problem (SVRP) with stochastic correlated demands, stochastic travel times, and time windows, integrating these challenges into a unified framework.
  - It begins by addressing the SVRP with stochastic correlated demands, focusing on scenarios where customer demands are correlated with external factors, and introduces a reinforcement learning framework that models these correlations and dynamically adjusts vehicle routes.
  - The discussion extends to SVRP with stochastic travel times, tackling uncertainties in travel times by presenting an RL-based solution that optimizes routes in real-time based on travel conditions, supported by a detailed overview of the problem formulation, model development, and environment settings.
  - The chapter further incorporates time windows into the routing problem, which adds the complexity of delivering goods within specific time constraints while accounting for stochastic demands and travel times.
  - The proposed RL framework is described in detail, including its methodology, problem formulation, architecture, and its ability to optimize routing strategies while respecting time windows.
  - Extensive experiments are conducted for each scenario, comparing the RL-based approach to traditional methods, and the results demonstrate significant improvements in travel cost, time complexity, and adaptability across diverse environments.
  - The chapter concludes with an analysis of multi-vehicle scenarios, training performance, and insights into how correlated demands, stochastic travel times, and time windows collectively influence routing strategies and overall efficiency.
- Chapter 4: Large Language Models for Combinatorial Problems
  - This chapter explores the use of Large Language Models for solving combinatorial optimization problems. A novel prompting strategy called Self-Guiding Exploration is introduced, which leverages the power of LLMs to solve NP-hard problems like the Traveling Salesman Problem, Vehicle Routing Problem, and Job Scheduling Problem.
  - The chapter details the algorithm behind SGE, the baseline prompting techniques used for comparison, and the results of experiments conducted across multiple CPs.
  - The chapter also highlights the broader application of LLMs to reasoning tasks, showing how SGE improves performance not only in combinatorial problems but also in arithmetic and symbolic reasoning tasks.
- Chapter 5: Conclusions and Future Work
  - The final chapter summarizes the key findings and contributions of the thesis. It discusses how the proposed learning algorithms and LLM-based approaches have advanced the state of machine learning in solving combinatorial optimization problems.

– Finally, potential future research directions are outlined, including suggestions for improving the efficiency of the models and extending their applicability to other complex optimization problems.

Chapter 2

Learning Algorithm for Job-Shop Scheduling Problem

This chapter covers the use of Curriculum Learning in combination with Reinforcement Learning to enhance dispatching policy inference for the Job Shop Scheduling Problem. The chapter introduces the concept of JSP, which involves scheduling multiple jobs on heterogeneous machines to minimize the total completion time, or makespan. Traditional heuristic methods, such as Priority Dispatch Rules, while effective, lack adaptability and scalability. The focus of this chapter is to explore how Neural Combinatorial Optimization can automate the creation of domain-specific heuristics using RL, addressing JSP's complexity. To overcome the challenge of RL struggling with large JSP instances, the chapter introduces a size-agnostic deep learning model and proposes a novel Reinforced Adaptive Staircase Curriculum Learning (RASCL) strategy. This strategy dynamically adjusts the learning difficulty by revisiting poorly performing instances, leading to significant improvements in scheduling performance. Experimental results presented in this chapter demonstrate the superiority of the proposed method in reducing the optimality gap on standard datasets such as Taillard's and Demirkol's JSP instances, making it a critical contribution to advancing RL-based solutions for large-scale scheduling problems.

2.1 Introduction

The Job-shop Scheduling Problem (JSP) is a combinatorial problem with vast implications on real-world tasks. It is formulated as a set of jobs, each consisting of a set of operations, to be processed on a set of heterogeneous machines. Furthermore, each operation has a specific machine assigned to it, and the operational time it takes to complete is known in advance. The goal is to define the scheduling order of the operations such that the total completion time or *makespan* is minimized. This problem applies to many tasks concerned with the optimal assignment of capital goods versus means of production, e.g., manufacturing, storage, transportation, etc. However, the JSP is an NP-hard problem, and it is therefore impractical to solve optimally. Optimization methods such as integer programming or constraint programming are computationally expensive methods that are unrealistic to use when fast solutions are required. Therefore, in practice, hand-engineered heuristics are commonly used to find approximate solutions to the problem.

However, designing such heuristic rules is a daunting task that requires specialized knowledge of the problem. In the case of the JSP, Priority Dispatch Rules (PDRs) are a family of constructive heuristics used on scheduling problems that are fast to compute and easy to implement. Although, it is not clear how to extrapolate these rules to different problems. Therefore, recent studies have focused on using Reinforcement Learning (RL) to derive domain-specific heuristics automatically. This technique is known in the literature as Neural Combinatorial Optimization (NCO). Unlike traditional PDRs that use the same dispatching rule for any instance in the problem, NCO particularizes a solution based on the instance information, which makes this technique very compelling. This technique has already demonstrated promising results in the JSP [203, 233].

This work provides a nostrum for improving the learning strategy on the JSP using Curriculum Learning (CL). To this end, we design a problem-specific architecture and novel training approach that stresses this direction. Our main contributions are summarized below:

- This work particularizes a **deep learning model** to address the JSP. Particularly, we present an autoregressive model that iteratively constructs the solution based on the operations that are yet to be scheduled, as well as on the state of the machines during the resolution process. The model is equivariant w.r.t. the job information but also size-agnostic, which enables to seamlessly use it on different problem sizes and train it performing a curriculum strategy.
- We also present **Reinforced Adaptive Staircase Curriculum Learning (RASCL)**, a novel CL strategy that improves the model's performance based on the flexible assignment of difficulty levels during the learning process. In this work, we identify the difficulty for traditional RL methods to learn on large instances of the problem. To mitigate this issue, RASCL dynamically reinforces the model by revisiting the worst-performing instances. Conducted experiments show that this strategy presents an improvement when compared to previous CL approaches.

The aforesaid ideas deliver an improvement in the scheduling solutions when compared to state-of-the-art works on JSP [233]. Notably, we reduce the optimality gap from 19.35% to 10.46% on Taillard's instances and from 38.43% to 18.85% on Demirkol's instances[1].

The remainder of this chapter is organized as follows. In Section 2.2, we present the state-of-the-art of NCO applied on the JSP. In Section 2.3, we provide a formal description of the method. Then, in Section 2.4 the architecture description, and in Section 2.5 the learning strategy are provided. In Section 2.6 the method is experimentally validated. Finally, Section 2.7 concludes this chapter.

2.2 Related Work

The first attempts of using RL to address scheduling problems date back to the 90s [138, 139, 237]. Of specific relevance is Zhang and Dietterich's paper [237] on allocating resources for NASA

[1]Taillard and DMU datasets are publicly available on http://optimizizer.com/TA.php and http://optimizizer.com/DMU.php respectively.

shuttle missions. As it was reflected in the literature, one of the advantages of using RL for such a purpose is that it can be seamlessly used for static, dynamic [5, 43], and stochastic variants of the problem. However, due to technological limitations, this technique could only be applied to small instances of the problem, limiting its applicability significantly.

The recent rise of deep learning has allowed extrapolating this technique to more realistic problem instances. Prior works have particularized deep neural networks, e.g., Pointer Networks [198], to learn in combinatorial spaces. In [7], deep RL was implemented for the first time to learn *end-to-end* solutions to combinatorial problems. The authors used the Pointer Network in an actor-critic architecture to address the Travelling Salesman Problem. Further studies implemented Transformer networks [36, 102, 195]. However, all these works share in common that they are based on sequence-to-sequence models, where the complete solution is output at once. Thereby, heavy sampling and searching techniques were required at inference to improve the solution. Our approach is in line with [147], where the Vehicle Routing Problem is described as a Markov Decision Process, and the solution is iteratively constructed based on sequential decisions.

In the particular case of the JSP, several techniques have been applied to learn on the problem. Imitation learning was used in [79] to learn from optimal solutions on training instances that were labeled using a Mixed-Integer Programming (MIP) solver. RL has also been applied to the problem, e.g., to select pre-defined candidate PDRs according to the scheduling conditions [5, 122]. Other works have addressed the problem from a multi-agent perspective, e.g., in cooperative manufacturing [43, 210] where each agent controls a production line. Moreover, numerous examples of scheduling in many application domains, including manufacturing [122, 203], distributed computing [142, 179, 234] or supply chains. Despite the effort, many of these approaches do not beat traditional heuristics. In addition, a major limitation in many of these works is that the state representation is hard-bounded by some factors (e.g., size of jobs or number of operations to consider), not enabling to scale the solution to arbitrary problem sizes.

Zhang et al. [233] presented the first size-agnostic model on the problem. It formulates the JSP as a disjunctive graph and uses a high discriminative Graph Isomorphism Network (GIN) to embed the states in the resolution procedure. They prove to capture raw features from small problem instances and successfully generalize solutions to much larger instances that the ones experienced during the training. Even though graph neural networks have shown unprecedented success embedding not-euclidean spaces, in this work, we show that a simpler autoregressive neural network is enough to better capture the state information and infer a dispatching rule.

One of the key features of the model presented in this work is the use of CL to stress the learning capabilities of the model to even larger problem instances. This aspect has been pointed out several times in the literature, e.g. [246] employs transfer learning to reconstruct the trained policies on problems of different sizes. However, policy transfer is still relatively costly and inconvenient. Also, [126] uses lifelong learning, where an agent will not only learn to optimize one specific problem instance but reuse what it has learned from previous instances. They also proposed a parallel training method that combines asynchronous updates with a deep deterministic policy gradient to speed up model training. In [125], the authors use a CL strategy with an adaptive staircase mechanism, where, at each iteration, the model changes the difficulty level according to the model performance. In our work, we build on top of this idea, tracking the model's behavior during the learning stressing the worst-performing levels.

2.3 Method

In the JSP, a finite set of m jobs $\{J_i\}_{i=1}^{m}$ are to be processed on a finite set of n machines $\{M_k\}_{k=1}^{n}$. Each job J_i consists of a sequence of n operations $O_{i,1} \rightarrow O_{i,2} \rightarrow ... \rightarrow O_{i,n}$, that have to be processed in a predetermined order. For each operation, $O_{i,j}$, the machine M_k is assigned for a given processing time $D_{i,j}$. The goal is to determine the scheduling order of the operations such that the total execution time or *makespan* is minimized.

There are several constraints that need to be taken into account when obtaining a solution on the problem:

- no-overlap constraints, determine that each machine shall process only one operation at a time;
- non-preemptive constraints, stating that once the process of an operation is initiated, it shall not be interrupted before completion;
- precedence constraints, establishing that the order of operations inside a job J_i, where operation $O_{i,j+1}$ shall not be scheduled until the previous operation $O_{i,j}$ of the job J_i is completed.

2.3.1 Learning the Dispatching Policy

In this work, we formulate the resolution process on the JSP as a Markov Decision Process (MDP), in which the solution is iteratively constructed according to the dispatching decisions inferred. This process resembles how constructive heuristic algorithms operate. We are therefore concerned with learning the dispatching policy that based on the state on the resolution process infers the operation to schedule next. To this end, we propose learning an autoregressive model that dispatches the operations according to the scheduling policy π_θ, which is parameterized using a neural network and optimized using RL.

More formally, we consider that a solution to an instance $x \in \mathcal{X}$ of the JSP consists of $t \in \{1, ..., m \cdot n\}$ decision steps that correspond to the dispatching operations. We denoted as $s_t \in \mathcal{S}$ the state of the machinery, which evolves with the scheduling decisions at every step. The actions denoted as $a_t \in \mathcal{A}$ correspond to the job for which the next pending operation is dispatched at that step. The goal is therefore learning a dispatching policy $\pi : \mathcal{X} \times \mathcal{S} \rightarrow \Delta^{|\mathcal{A}|}$ that for an instance x at the resolution state s_t, outputs the probability distribution over the next pending operations. The transition function $P : \mathcal{X} \times \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ corresponds to the dynamics on the problem, which are deterministic. The model dynamics are therefore expressed as

$$s_{t+1} \sim P(\cdot|x, s_t, a_t) \qquad a_t \sim \pi(\cdot|x, s_t). \tag{2.1}$$

The process repeats until all operations are scheduled. At that point, the solution y is defined as the sequence of dispatch actions $(a_1, a_2, ..., a_{m \cdot n})$.

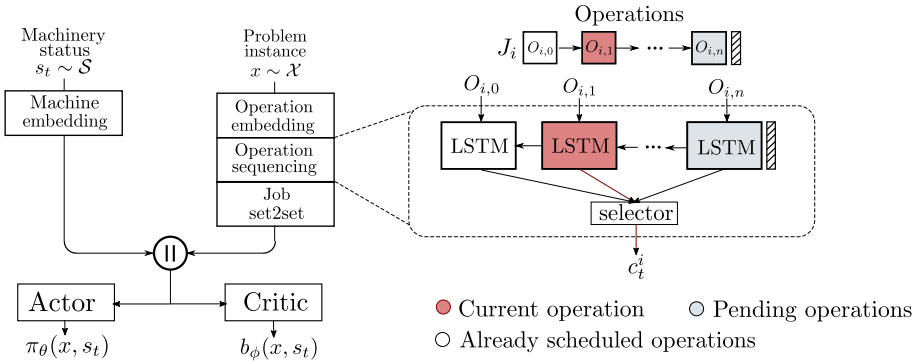


Figure 2.1: **Model Architecture.** The model receives the instance description x as well as the state of the machinery s_t at step t and outputs $\pi(x, s_t)$ a distribution over the jobs indicating the next pending operations to be scheduled. The operations on each job are recurrently encoded using an LSTM network, and a set2set network is used to combine the job information in an equivariant manner. The information regarding the machinery and job status is concatenated and connected to the actor and critic heads respectively.

Regarding the cost function $C : \mathcal{X} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, it is defined as the temporal increment in the completion time τ between two successive states

$$C_t(x, s_t, a_t) = \tau(x, s_{t+1}) - \tau(x, s_t), \tag{2.2}$$

where $\tau(x, s_t)$ refers to the time required to complete all operations from the initial state until s_t, which is computed using the Gantt diagram of a given instance. The cost function can be seen as the negative of the RL reward, and it is formulated to easily apply the *reward-to-go* trick [182]. This uses causality to remove the past sum over rewards so that only rewards collected after the current decision step are considered, which helps in reducing the variance of policy gradient. The makespan $T(x)$ is therefore directly obtained as the sum of collected costs along the resolution process $T(x) = \sum_t C_t(x, s_t, a_t)$.

In order to learn the policy we resort to the Reinforce algorithm [215] and the model is optimized for minimizing the expected makespan,

$$\mathcal{J}^\pi(\theta) = \mathbb{E}_{x\sim\mathcal{X}, a\sim\pi_\theta(\cdot|x,s_t)}\left[T^{\pi_\theta}(x)\right]. \tag{2.3}$$

Following the common variation of *Policy Gradient Theorem* [183], the gradient of the objective function is obtained as

$$\nabla_\theta \hat{\mathcal{J}}^\pi(\theta) \approx$$

$$\frac{1}{B}\sum_{b}\sum_{t=1}^{n\cdot m}\Big((G(x, s_t) - b_\phi(x, s_t)) \cdot \nabla_\theta \log \pi_\theta(a_t|x, s_t)\Big), \tag{2.4}$$

where $b_\phi(s_t, x)$ is the *baseline* - the estimate of the value function at state s_t of the problem instance x, and G, called *return*, is the actual cumulative cost starting at time step t and till all $n \cdot m$ operations are scheduled $G(s_t, x) = \sum_t^{n\cdot m} C_t(s_t, a_t)$. The baseline is computed using a

critic head and it is used to reduce the variance of the gradients, and therefore, to speed up the convergence. The critic is parameterized with parameters ϕ and it is trained to minimize

$$L(\phi) = \frac{1}{B} \sum_{b} \sum_{t=1}^{n \cdot m} ||b_\phi(x, s_t) - G(x, s_t)||^2 \,. \tag{2.5}$$

2.3.2 Inference Strategies

As reflected in the literature [7, 102], the greedy inference of scheduling decisions from the policy function π_θ used to result in poor results in NCO. Therefore, inference strategies, although sometimes quite computationally expensive [7], are commonly used to report competitive solutions. In the following, we describe the inference strategies considered in this work:

Sampling. This technique produces several scheduling solutions by repeating the inference process directly sampling from the policy distribution. Sometimes a temperature hyper-parameter is used to soften the output distribution and increase the diversity of the solutions. We denote as s the width of the sampling strategy.

POMO [105]. Policy Optimization with Multiple Optima mixes both, *greedy* and *sampling* inference strategies. At the initial state, this method rolls out p initial scheduling actions $\{a_0^{(1)}, ..., a_0^{(p)}\}$. Then, the solutions are completed using the greedy strategy. This creates p solutions to the problem $\{y^{(1)}, ..., y^{(p)}\}$ that are evaluated to select the best one. The authors demonstrate that this approach is competitive and computationally efficient compared to the *sampling* strategy.

Beam search [87, 203]. This strategy explores a beam of b dispatches at each decision step. At the initial state, the b most likely scheduling actions $\{a_0^{(1)}, ..., a_0^{(b)}\}$ are selected. Then, at the next decision step for each candidate solution, a new set of b actions is rolled out, thus, the total number of available actions becomes $b \times b$. To prevent the number of candidate solutions from exploding in number, beam search only selects the b candidates with the highest likelihood at each step. At the end of the inference process, b candidate solutions are created.

2.4 Architecture Details

Top view of the architecture. The proposed architecture is depicted in Fig. 2.1. The model receives the instance description x as well as the internal state during the problem resolution s_t and outputs $\pi(x, s_t)$, the distribution over the next pending operations at the current decision step t. The state of the problem consists of the following features: the index of the operations being currently processed, the status of machines and the remaining processing times. Regarding the job information, the sequencing of the operations for each job is obtained using a recurrent LSTM [66] network. Once the operation sequencing is encoded, the job information is passed through a *set2set* module [197], which is particularly useful to aggregate the job descriptions in an

equivariant manner. This module neglects the positioning of the jobs in the instance description, which increases the sample efficiency of the model as equivalent representations of the problem have the same representation. Finally, the embeddings of both descriptors are concatenated and passed into actor-critic networks to calculate the distribution over the dispatching actions, as well as the estimate of the return.

Operation sequencing. Considering that the dispatching actions at any given step only depend on the remaining operations, we use a LSTM network to capture this information. The LSTM operates backward, starting from the last operation in this job till the current operation. The embeding for the operation sequence is computed once for the instance description, and its result is stored for being used during the resolution process. Particularly, the index of the operation currently processed is used to select the output of the LSTM corresponding to the current step t. We denote this vector c_t^i and it encodes the information of the pending operations for the job J_i.

Decoding the solution. The solution is sequentially constructed in an autoregressive process where at each step the model indicates the operation to dispatch next. This is done by sampling from the probability distribution the policy-head computes. The model tracks therefore the operations already scheduled and only operates over the next available operation in each job. Once the scheduling order is determined, we can compute the makespan. The model needs to track also when all the operations in a job are scheduled, as selecting the job is no longer a valid action. To this end, we use a masking scheme that operates on the policy-head and reduces to zero the chances of selecting an invalid dispatch.

Algorithm 1: RASCL algorithm

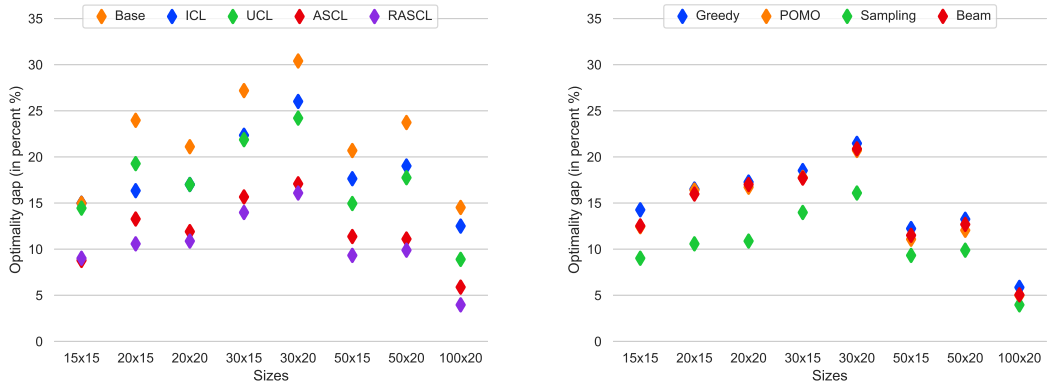
Data: Define the levels of difficulty $\mathcal{L} = \{l_0,...,l_K\}$ and the optimality threshold δ_{th}
Initialize $L = \{l_0\}$, $l = l_0$, and $k = 0$
while $k \leq K$ **do**
  Update $\{g(l) \mid l \in L\}$ according to the optimality gaps on the testing dataset.
  if $\max\{g(l)\} \leq \delta_{th}$ **then**
    Increase the difficulty $k = k + 1$ and $L = L \cup \{l_k\}$
  else
    Sample level $l \sim$ softmax$(\{1 \,/\, g(l) \mid l \in L\})$
  end
  Train policy π_θ on level l
end
Result: Trained policy π_θ

2.5 Learning Using a Curriculum Strategy

Curriculum Learning (CL) is a methodology that uses scenarios of increasing complexity to train the model. CL was adopted from social and educational systems to build a curricula of increasing complexity [206]. Such an approach allows the model to learn policies from the easiest scenario that can be extrapolated to harder levels of difficulty. In the case of the JSP, we identify the

difficulty levels as the different sizes on the problem $m \times n$. The bigger the number of jobs m and operations n, the harder is to learn. In this work, we consider the following most popular CL strategies:

Incremental curriculum learning (ICL). In CL a set of levels of difficulty $\mathcal{L} = \{l_0,...,l_K\}$ has to be predefined for performing the training. Incremental CL represents the simplest approach and it sequentially trains the model on increasing difficulty levels in a sequential manner. The drawback of this strategy is the model's tendency to catastrophic forgetting when trained on consecutive grades of complexity [125], as previous levels of difficulty are not revisited.



(a) Curriculum strategies

(b) Inference strategies

Figure 2.2: Optimatlity gap (in percent) on the (a) curriculum strategies and (b) inference strategies on Taillard's instances. In (a), all strategies incorporate a sampling (s=128). In (b), the measurements are obtained on the RASCL model for POMO (p=3), sampling (s=128) and Beam search (b=3).

Uniform curriculum learning (UCL). In uniform CL, at each iteration the model selects the training complexity with equal chances from the different difficulty levels. In this way, the model is exposed to various training levels and the policy stays inherent to different problem sizes. However, based on the work of [125], learning on small-size tasks is more accessible rather than sampling each training level from a uniform distribution of all sizes of interest, which eventually may not be the most helpful learning strategy.

Adaptive staircase curriculum learning (ASCL). Adaptive staircase is introduced to RL in [125]. This strategy introduces flexibility in selecting the level of complexity during the training. Specifically, the model initializes the learning process on the easiest difficulty level l_0 and adaptively increases the difficulty based on the observed optimality gap. At the level l_k, ASCL revisits previous difficulty levels to not 'forget' its corresponding policies. The levels to revisit are selected randomly following a uniform distribution.

Reinforced adaptive staircase CL (RASCL). In this work we propose a methodology for dynamically adjusting the difficulty between the levels. Similar to ASCL, the agent's is initiated

at the lowest difficulty level l_0 with the aim to make its way to the top level l_K. However, the limitation of ASCL is inherent in the randomness of revisiting the previous levels.

To tackle the problem, RASCL proposes a further improvement of ASCL by revisiting the levels where model shows the worst performance. Specifically, RASCL determines the level of difficulty comparing the estimation of the return to the optimal one. To this end, this method evaluates the optimality gap $g(l)$ for each difficulty level and establishes a policy for revisiting levels inversely proportional to the gap observed. The bigger the optimality gap achieved in a level the higher the chances to revisit the level. The complete algorithm description can be found in Algorithm 1. Particularly, the method tracks a subset of levels L from where the training is performed. The subset of levels available starts as $L = \{l_0\}$ and it increases during the learning. Once the set of levels L are mastered, L is augmented incorporating the next difficulty level. The condition for incorporating the next level is by comparing the maximum optimality gap on L to a predetermined threshold δ_{th}. If the model's return is below the threshold, RASCL adds the new level of difficulty. Otherwise, the method keeps training sampling instances from the current subset of levels L. The proposed algorithm reinforces the learned policy by revisiting and anchoring the most problematic difficulty levels.

2.6 Experiments

Datasets. We train and evaluate our model on scheduling instances of the sizes used on Taillard's [185] and Demirkol's (DMU) [34] datasets: 15x15, 20x15, 20x20, 30x20 and 30x15. The instances are expressed as JSP $m \times n$, where m represents the number of jobs, and n is the number of operations. To test the model's performance, we use substantially larger instances that are not included in the training set: 50x15, 50x20, and 100x20.

Baselines. For the choice of *baselines*, we resort to the analysis on priority dispatch rules (PDRs) in [169] and choose the most popular ones in the research community: Shortest Processing Time (SPT), Minimum Ratio of Flow Due Date to Most Work Remaining (FDD/WKR), Most Work Remaining (MWKR), Most Operations Remaining (MOPNR). Among neural combinatorial solvers, we use the latest state-of-the-art results of the graph neural model on the public JSP benchmarks presented in [233]. For approximating the optimal solution, required when estimating the optimality gap in RASCL, we resort to the constraint programming solver of Google OR-Tools CP-SAT. In the following, we refer to the *Base model* as architecture presented in Fig. 2.1 trained for a particular size of the JSP, without performing a curriculum.

First, we train five instances of the base model on five sizes: 15x15, 20x15, 20x20, 30x15, and 30x20. Each instance of the base model is dedicated to one of five problem sizes. Then, we test all base model instances on an extended set of sizes, adding 50x15, 50x20, and 100x20. The results reflect that the model achieves best performance when trained on the smallest size 15x15, where it shows the smallest optimality gap on all testing sizes, from 14.98% for 15x15 to 9.58% for 100x20. This reflects the difficulty for the model to directly learn on larger instances of the problem.

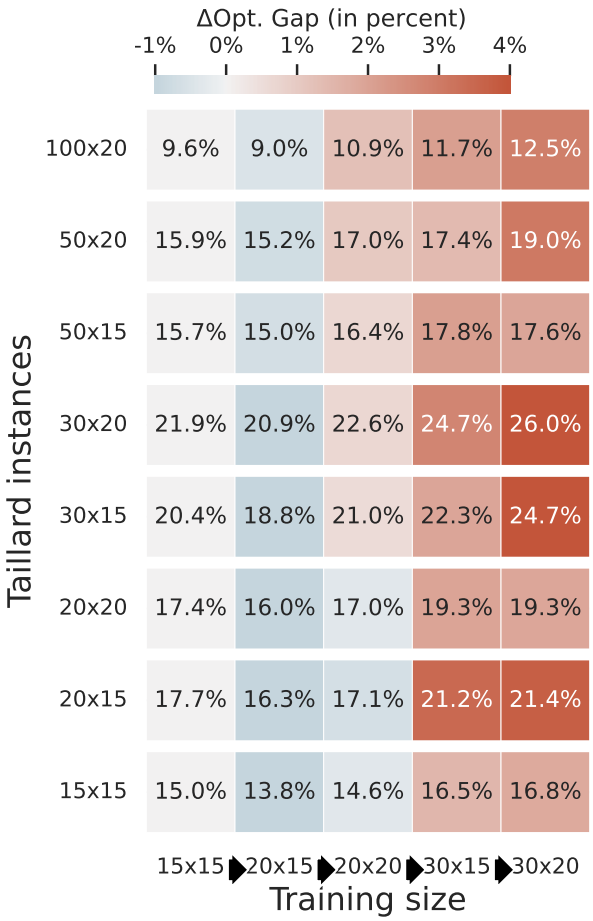


Figure 2.3: Increment in the optimality gap (in percent) on Taillard's instances when trained using (a) Base and (b) ICL learning strategies. From (a), learning directly on large instances is hard for the model; whereas in (b), progressively incrementing the problem size during training reduces the optimality gap.

The model's results can be improved further, as the base model trained on a small size can hardly capture all combinatorial space of action policies inherent in large-size problems. To validate this hypothesis we incorporate different CL methods in the training. We start with ICL, for which we train five instances of the base model in the following manner: the model assigned to the 15x15 size is trained on this size only, the model assigned to 20x15 is trained sequentially on the previous size, which in this case 15x15, and on its assigned 20x15 size. The same logic applies for all sizes up to 30x20, and each learning cycle takes n iterations. Fig. 2.3 shows the results of this approach. Here, the 20x15 model, being trained consecutively on 15x15 and 20x15 sizes, exhibits the best performance reducing the average gap across all sizes tested. However, the ICL approach has an obvious drawback. The best result is still achieved on a small-size problem, 20x15, meaning that learning on larger-size instances is still hard.

To address the problem, we test three additional CL strategies including our proposed RASCL, see Fig. 2.2(a). One can see that UCL generally shows near-ICL behavior, while ASCL and RASCL models demonstrate notably better performance. Particularly, RASCL shows the smallest optimality gap on all test sizes, except for 15x15, where ASCL slightly outperforms RASCL by

Instances		SPT	FDD/WKR	MWKR	MOPNR	[233]	RASCL (s=128)
15×15	Obj.	1546.1	1808.6	1464.3	1481.3	1547.4	**1339.8**
	Gap	(25.89%)	(47.15%)	(19.15%)	(20.53%)	(25.96%)	**(9.02%)**
20×15	Obj.	1813.5	2054.0	1683.6	1686.7	1774.7	**1509.3**
	Gap	(32.82%)	(50.57%)	(23.35%)	(23.55%)	(30.03%)	**(10.58%)**
20×20	Obj.	2067.0	2387.2	1969.8	1968.3	2128.1	**1793.1**
	Gap	(27.75%)	(47.61%)	(21.81%)	(21.71%)	(31.61%)	**(10.87%)**
30×15	Obj.	2419.3	2590.8	2214.8	2195.8	2378.8	**2038.1**
	Gap	(35.27%)	(45.02%)	(23.91%)	(22.83%)	(33.0%)	**(13.98%)**
30×20	Obj.	2619.1	3045.0	2439.0	2433.6	2603.9	**2261.5**
	Gap	(34.44%)	(56.3%)	(25.17%)	(24.94%)	(33.62%)	**(16.09%)**
50×15	Obj.	3441.0	3736.3	3240.0	3254.5	3393.8	**3030.8**
	Gap	(24.11%)	(34.77%)	(16.86%)	(17.37%)	(22.38%)	**(9.32%)**
50×20	Obj.	3570.8	4022.1	3352.8	3346.9	3593.9	**3125.1**
	Gap	(25.54%)	(41.5%)	(17.95%)	(17.68%)	(26.51%)	**(9.89%)**
100×20	Obj.	6139.0	6620.7	5812.2	5856.9	6097.6	**5578.9**
	Gap	(14.41%)	(23.39%)	(8.31%)	(9.15%)	(13.61%)	**(3.96%)**

Table 2.1: Results on (a) Taillard's and (b) DMU instances. **Objective** indicates the average *makespan* for a given problem size; and **Gap**, the average difference (in percent) to the upper bound known for the instances.

0.24%. The reinforced approach displays the smallest optimality gap from 10.58% for 20x15 instances to 3.96% for 100x20 instances.

We also provide in Table 2.3 a comparison for different inference strategies: Greedy, POMO, Sampling and Beam search, all applied to the RASCL model. POMO ($p = 3$) and Beam ($s = 3$), show very similar results, only slightly better than greedy inference. The sampling strategy ($s = 128$) shows the best result across all test sizes, and, consequently, it is the inference strategy used in the following experiments.

At this point, we summarize the results using RASCL at learning and Sampling as the inference strategy. Table 2.1 provides the comparison results on Taillard's dataset. RASCL model displays robust behavior on all test sizes, improving previous best results [233] on average by 45.94% in terms of optimality gap. This model shows a maximum optimality gap of 16.09% on Taillard's dataset, which is a much tighter bound than the offered by traditional PDRs.

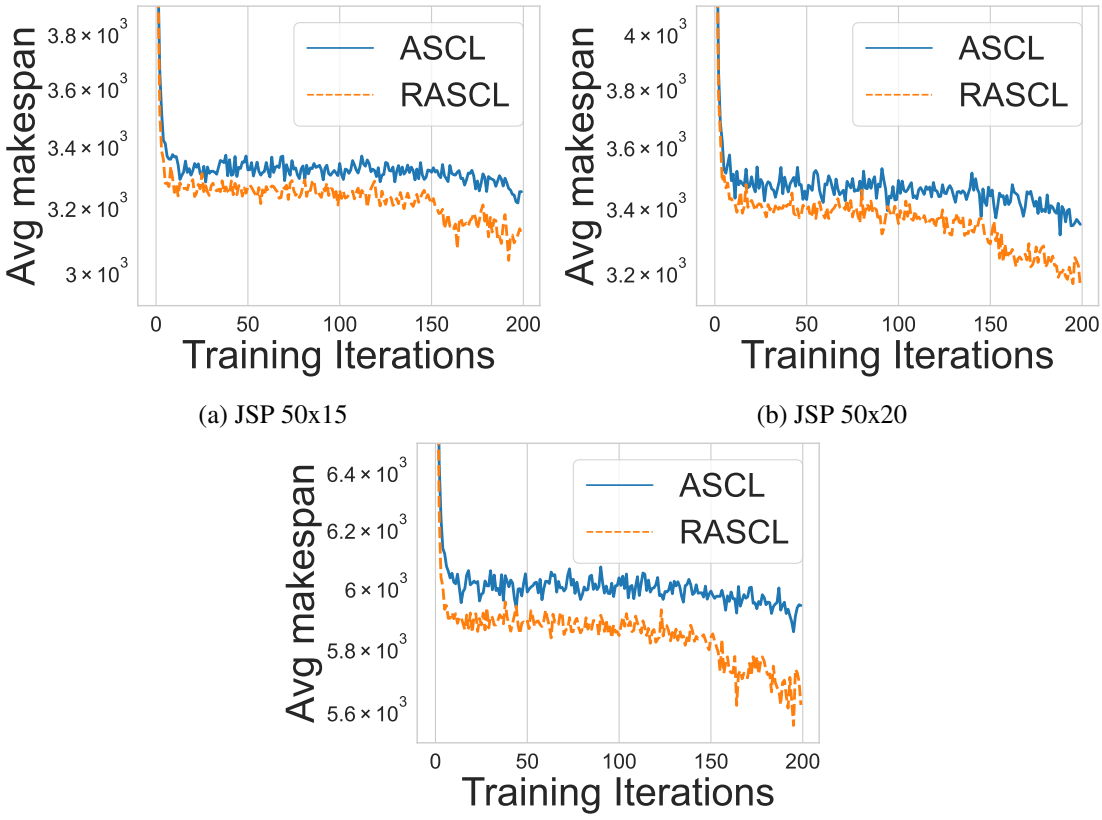
To further test our model, we refer to DMU dataset consisting of 80 instances from 20x15 to 50x20 sizes. Table 2.2 shows outperforming behavior of RASCL model again. On average, it improves previous best results by 50.95%.

Instances		SPT	FDD/WKR	MWKR	MOPNR	[233]	RASCL (s=128)
20×15	Obj.	4951.5	4666.3	4909.9	4513.2	4215.3	**3610.0**
	Gap	(64.13%)	(53.57%)	(62.15%)	(49.16%)	(38.95%)	**(19.36%)**
20×20	Obj.	5690.5	5298.2	5489.0	5052.3	4804.5	**4028.9**
	Gap	(64.57%)	(52.52%)	(58.16%)	(45.17%)	(37.74%)	**(15.98%)**
30×15	Obj.	6306.2	6016.5	6252.9	5742.8	5557.9	**4522.0**
	Gap	(62.57%)	(54.12%)	(60.95%)	(47.14%)	(41.86%)	**(16.35%)**
30×20	Obj.	7036.0	6827.3	6925.0	6491.9	5967.4	**5106.0**
	Gap	(65.91%)	(60.09%)	(63.16%)	(51.97%)	(39.48%)	**(20.0%)**
40×15	Obj.	7601.2	7420.0	7484.2	7105.5	6663.9	**5731.9**
	Gap	(55.88%)	(51.42%)	(52.87%)	(44.72%)	(35.38%)	**(17.49%)**
40×20	Obj.	8538.1	8210.9	8460.9	7870.7	7375.8	**6584.1**
	Gap	(63.0%)	(55.52%)	(61.11%)	(49.22%)	(39.38%)	**(25.42%)**
50×15	Obj.	8975.4	9150.2	8906.0	8436.5	8179.4	**7242.1**
	Gap	(50.37%)	(52.53%)	(48.93%)	(40.79%)	(36.2%)	**(21.54%)**
50×20	Obj.	10132.8	9899.6	9807.0	9408.0	8751.6	**7176.9**
	Gap	(62.2%)	(57.26%)	(56.4%)	(49.61%)	(38.86%)	**(14.66%)**

Table 2.2: Results on (a) Taillard's and (b) DMU instances. **Objective** indicates the average *makespan* for a given problem size; and **Gap**, the average difference (in percent) to the upper bound known for the instances.

	Instances	Greedy	**POMO** [105] (p=3)	**Sampling** (s=128)	**Beam** (b=3)
TAILLARD	15×15	1404.2 (14.26%)	1381.9 (12.43%)	**1339.8 (9.02%)**	1382.9 (12.52%)
	20×15	1590.3 (16.52%)	1588.6 (16.40%)	**1509.3 (10.58%)**	1582.9 (15.97%)
	20×20	1896.5 (17.27%)	1887.5 (16.71%)	**1793.1 (10.87%)**	1892.1 (17.01%)
	30×15	2118.9 (18.52%)	2105.0 (17.78%)	**2038.1 (13.98%)**	2104.8 (17.7%)
	30×20	2365.7 (21.47%)	2350.8 (20.70%)	**2261.5 (16.09%)**	2354.0 (20.87%)
	50×15	3111.6 (12.23%)	3079.1 (11.08%)	**3030.8 (9.32%)**	3090.9(11.5%)
	50×20	3219.6 (13.24%)	3185.9 (12.05%)	**3125.1 (9.89%)**	3203.4 (12.68%)
	100×20	5680.9 (5.86%)	5633.3 (4.97%)	**5578.9 (3.96%)**	5637.5 (5.05%)

Table 2.3: Inference strategies for RASCL on Taillard's instances. Results are reported as average **Objective (Gap)**.



(a) JSP 50x15

(b) JSP 50x20

(c) JSP 100x20

Figure 2.4: Average makespan observed during the training for (a) 50x15 (b) 50x20, and (c) 100x20 sizes. RASCL finds more competitive solutions when compared to ASCL, and stresses the learning process during more iterations.

2.7 Conclusions and Future Work

In this work, we present a deep-RL model to automatically learn a dispatching policy on the JSP. To this end, we formulate the resolution process as MDP, in which a solution is iteratively constructed based on intermediate states on the resolution process. In this work, we present a particularized deep learning model on the JSP that is size-agnostic and also equivariant w.r.t. the job information, which benefits the training as equivalent instances are similarly encoded. In order to improve the behavior of the model, this work relies on Curriculum Learning. In this direction, we present a novel RASCL strategy, which dynamically adjusts the difficulty of the learning according to the model's performance during the learning process. Experiments on Taillard's and Demirkol's instances show that our model improves the optimality gap w.r.t. the current state-of-the-art model by 45.94% and 50.95%, respectively. Our results corroborate that learning using a curricula is key for improving the results on the problem.

There are several future directions for this work. For example, we would like to remark the potential of this technology for addressing stochastic or partially observable problems, domains where traditional approaches have shown limited capabilities.

Hyper-parameter	Value
Instance input dimension	$2 \times m \times n$
Machinery state input dimension	$2 \times n$
Encoder dimension	128
LSTM [layer, dim]	$[1, 128]$
Actor/Critic [depth,width]	$[2, 16]$
Activation function	ReLU
Learning rate	1E-04
Optimizer	Adam
Batch size	128
Training iterations (per level) - n	45000
Level evaluation (iterations)	100_{th}
Threshold opt. gap - δ_{th}	15%

Table 2.4: Hyper-parameter values.

2.8 Models and Configurations

We train the models using the same set of hyperparameters for every size on the problem (see Table 2.4). The validation set consists of 1,000 randomly generated JSP instances. Both the actor and critic utilized fully-connected networks with two layers: the first layer contained 16 neurons, while the second layer contained m neurons for the Actor and a single neuron for the Critic. ReLU activation was used for all layers. The training was conducted using the Adam optimizer with a learning rate of 10^{-4} and a batch size of 128. The Base, ICL and UCL models are trained for 45,000 iterations. Each model including ASCL and RASCL is tested every 100_{th} iteration. During training, for ASCL and RASCL, a maximum gap of δ_{th} = 15% between the optimal test solutions and the model's solutions was used as the criterion for proceeding to the next level. Figure 2.4 shows the average makespan results for 50x15, 50x20, and 100x20 size instances during the training. Regarding the inference strategy, a tree-width of 3 is used for the Beam search and POMO selection strategies, and a sample size of 128 is used for the sampling strategy. Our implementation is available online.[2] Detailed instructions for reproducing the training, evaluation, and plotting the results are included. The model is implemented in PyTorch, and the JSP environment was created using the Gym library. The training was conducted on an NVIDIA A100 SXM 40GB GPU with 2x AMD EPYC 7742 CPUs (8 cores) and 256GB RAM. To train the RASCL model from scratch, it required approximately 8 hours using the described hardware.

[2] `https://github.com/Optimization-and-Machine-Learning-Lab/Job-Shop/tree/main_nips`

2.9 Additional Tables

The additional tables containing the complete results on Taillard's and DMU instances are provided below. Table 2.5 presents the results on Taillard and Table 2.6 presents the results on DMU datasets. The results for the following PDRs are reported: Shortest Processing Time (SPT), Minimum Ratio of Flow Due Date to Most Work Remaining (FDD/WKR), Most Work Remaining (MWKR) and Most Operations Remaining (MOPNR); as well as the neural solver [233]. The "UB" column represents the best upper bound known in the literature. The symbol "*" indicates that the UB is known to be optimal. Our results are reported averaging for 3 random seeds.

Table 2.7 presents a benchmark for the different CL strategies on Taillard's individual instances. Results are reported using a sampling size of 128 on all strategies and likewise, results are averaged for 3 random seeds.

Finally, Table 2.8 provides a comparison for different inference strategies: Greedy, POMO (p = 3), Sampling (s = 128) and Beam search (s = 3), applied to the RASCL model.

Table 2.5: Results on Taillard's instances.

	Instance	SPT	FDD/WKR	MWKR	MOPNR	[233]	RASCL (s=128)	UB
15×15	TA1	1462 (18.77%)	1841 (49.55%)	1491 (21.12%)	1438 (16.82%)	1443 (17.22%)	**1379 (12.02%)**	1231*
	TA2	1446 (16.24%)	1895 (52.33%)	1440 (15.76%)	1452 (16.72%)	1544 (24.12%)	**1325 (6.51%)**	1244*
	TA3	1495 (22.74%)	1914 (57.14%)	1426 (17.08%)	1418 (16.42%)	1440 (18.23%)	**1338 (9.85%)**	1218*
	TA4	1708 (45.36%)	1653 (40.68%)	1387 (18.04%)	1457 (24.0%)	1637 (39.32%)	**1275 (8.51%)**	1175*
	TA5	1618 (32.19%)	1787 (46.0%)	1494 (22.06%)	1448 (18.3%)	1619 (32.27%)	**1303 (6.45%)**	1224*
	TA6	1522 (22.94%)	1748 (41.2%)	1369 (10.58%)	1486 (20.03%)	1601 (29.32%)	**1325 (7.03%)**	1238*
	TA7	1434 (16.87%)	1660 (35.29%)	1470 (19.8%)	1456 (18.66%)	1568 (27.79%)	**1332 (8.56%)**	1227*
	TA8	1457 (19.72%)	1803 (48.15%)	1491 (22.51%)	1482 (21.77%)	1468 (20.62%)	**1325 (8.87%)**	1217*
	TA9	1622 (27.32%)	1848 (45.05%)	1541 (20.96%)	1594 (25.12%)	1627 (27.71%)	**1412 (10.83%)**	1274*
	TA10	1697 (36.74%)	1937 (56.08%)	1534 (23.61%)	1582 (27.48%)	1527 (23.05%)	**1384 (11.52%)**	1241*
20×15	TA11	1865 (37.44%)	2101 (54.83%)	1685 (24.17%)	1665 (22.7%)	1794 (32.2%)	**1506 (10.98%)**	1357*
	TA12	1667 (21.95%)	2034 (48.79%)	1707 (24.87%)	1739 (27.21%)	1805 (32.04%)	**1514 (10.75%)**	1367*
	TA13	1802 (34.18%)	2141 (59.42%)	1690 (25.84%)	1642 (22.26%)	1932 (43.86%)	**1510 (12.43%)**	1343*
	TA14	1635 (21.56%)	1841 (36.88%)	1563 (16.21%)	1662 (23.57%)	1664 (23.72%)	**1455 (8.18%)**	1345*
	TA15	1835 (37.04%)	2187 (63.33%)	1696 (26.66%)	1682 (25.62%)	1730 (29.2%)	**1516 (13.22%)**	1339*
	TA16	1965 (44.49%)	1926 (41.62%)	1584 (16.47%)	1638 (20.44%)	1710 (25.74%)	**1485 (9.19%)**	1360*
	TA17	2059 (40.83%)	2093 (43.16%)	1804 (23.39%)	1856 (26.95%)	1897 (29.75%)	**1614 (10.4%)**	1462*
	TA18	1808 (29.51%)	2064 (47.85%)	1751 (25.43%)	1710 (22.49%)	1794 (28.51%)	**1560 (11.75%)**	1396
	TA19	1789 (34.31%)	1958 (47.0%)	1667 (25.15%)	1651 (23.95%)	1682 (26.28%)	**1451 (8.93%)**	1332*
	TA20	1710 (26.85%)	2195 (62.83%)	1689 (25.3%)	1622 (20.33%)	1739 (29.01%)	**1482 (9.94%)**	1348*
20×20	TA21	2175 (32.46%)	2455 (49.51%)	2044 (24.48%)	1964 (19.61%)	2252 (37.15%)	**1838 (11.94%)**	1642*
	TA22	1965 (22.81%)	2177 (36.06%)	1914 (19.62%)	1905 (19.06%)	2102 (31.37%)	**1752 (9.5%)**	1600
	TA23	1933 (24.15%)	2514 (61.46%)	1983 (27.36%)	1922 (23.44%)	2085 (33.91%)	**1748 (12.27%)**	1557
	TA24	2230 (35.64%)	2391 (45.44%)	1982 (20.56%)	1943 (18.19%)	2200 (33.82%)	**1807 (9.91%)**	1644*
	TA25	1950 (22.26%)	2267 (42.13%)	1941 (21.69%)	1957 (22.7%)	2201 (37.99%)	**1772 (11.1%)**	1595
	TA26	2188 (33.17%)	2644 (60.93%)	1951 (18.75%)	1964 (19.54%)	2176 (32.44%)	**1802 (9.68%)**	1643
	TA27	2096 (24.76%)	2514 (49.64%)	2091 (24.46%)	2160 (28.57%)	2132 (26.9%)	**1895 (12.8%)**	1680
	TA28	1968 (22.77%)	2330 (45.35%)	1997 (24.58%)	1952 (21.77%)	2146 (33.87%)	**1788 (11.54%)**	1603*
	TA29	2166 (33.29%)	2232 (37.35%)	1860 (14.46%)	1899 (16.86%)	1952 (20.12%)	**1753 (7.88%)**	1625
	TA30	1999 (26.2%)	2348 (48.23%)	1935 (22.16%)	2017 (27.34%)	2035 (28.47%)	**1776 (12.12%)**	1584
30×15	TA31	2335 (32.37%)	2459 (39.4%)	2134 (20.98%)	2143 (21.49%)	2565 (45.41%)	**2028 (14.97%)**	1764*
	TA32	2432 (36.32%)	2672 (49.78%)	2223 (24.61%)	2188 (22.65%)	2388 (33.86%)	**2092 (17.26%)**	1784
	TA33	2453 (36.96%)	2766 (54.44%)	2349 (31.16%)	2308 (28.87%)	2324 (29.76%)	**2114 (18.03%)**	1791
	TA34	2434 (33.15%)	2669 (46.01%)	2245 (22.81%)	2193 (19.97%)	2332 (27.57%)	**2077 (13.62%)**	1828*
	TA35	2497 (24.41%)	2525 (25.81%)	2226 (10.91%)	2255 (12.36%)	2505 (24.81%)	**2103 (4.78%)**	2007*
	TA36	2445 (34.41%)	2690 (47.88%)	2365 (30.02%)	2307 (26.83%)	2497 (37.27%)	**2046 (12.48%)**	1819*
	TA37	2664 (50.42%)	2492 (40.71%)	2130 (20.27%)	2190 (23.66%)	2325 (31.28%)	**2080 (17.45%)**	1771*
	TA38	2155 (28.81%)	2425 (44.95%)	2050 (22.53%)	2179 (30.25%)	2302 (37.6%)	**1882 (12.49%)**	1673*
	TA39	2477 (37.99%)	2596 (44.62%)	2221 (23.73%)	2167 (20.72%)	2410 (34.26%)	**2029 (13.04%)**	1795*
	TA40	2301 (37.87%)	2614 (56.62%)	2205 (32.12%)	2028 (21.51%)	2140 (28.22%)	**1930 (15.64%)**	1669
30×20	TA41	2499 (24.64%)	2991 (49.18%)	2620 (30.67%)	2538 (26.58%)	2667 (33.02%)	**2356 (17.51%)**	2005
	TA42	2710 (39.91%)	3027 (56.27%)	2416 (24.73%)	2440 (25.97%)	2664 (37.53%)	**2260 (16.68%)**	1937
	TA43	2434 (31.85%)	2926 (58.5%)	2345 (27.03%)	2432 (31.74%)	2431 (31.69%)	**2163 (17.17%)**	1846
	TA44	2906 (46.84%)	3462 (74.94%)	2544 (28.55%)	2426 (22.59%)	2714 (37.14%)	**2293 (15.87%)**	1979
	TA45	2640 (32.0%)	3245 (62.25%)	2524 (26.2%)	2487 (24.35%)	2637 (31.85%)	**2250 (12.5%)**	2000
	TA46	2667 (32.95%)	3008 (49.95%)	2447 (21.98%)	2490 (24.13%)	2776 (38.38%)	**2314 (15.35%)**	2006
	TA47	2620 (38.7%)	2940 (55.64%)	2263 (19.8%)	2286 (21.02%)	2476 (31.07%)	**2141 (13.34%)**	1889
	TA48	2620 (35.26%)	2991 (54.41%)	2356 (21.63%)	2371 (22.41%)	2490 (28.55%)	**2289 (18.17%)**	1937
	TA49	2666 (35.95%)	2865 (46.1%)	2382 (21.47%)	2397 (22.23%)	2556 (30.34%)	**2270 (15.76%)**	1961
	TA50	2429 (26.31%)	2995 (55.75%)	2493 (29.64%)	2469 (28.39%)	2628 (36.66%)	**2279 (18.51%)**	1923
50×15	TA51	3856 (39.71%)	3851 (39.53%)	3435 (24.46%)	3567 (29.24%)	3599 (30.4%)	**3181 (15.25%)**	2760*
	TA52	3266 (18.51%)	3734 (35.49%)	3394 (23.15%)	3303 (19.85%)	3341 (21.23%)	**3029 (9.91%)**	2756*
	TA53	3507 (29.08%)	3394 (24.92%)	3098 (14.02%)	3115 (14.65%)	3186 (17.26%)	**2921 (7.51%)**	2717*
	TA54	3142 (10.67%)	3603 (26.91%)	3272 (15.25%)	3265 (15.01%)	3266 (15.04%)	**2914 (2.64%)**	2839*
	TA55	3225 (20.38%)	3664 (36.77%)	3188 (19.0%)	3279 (22.4%)	3232 (20.64%)	**3050 (13.85%)**	2679*
	TA56	3530 (26.93%)	4016 (44.41%)	3134 (12.69%)	3100 (11.47%)	3378 (21.47%)	**3012 (8.31%)**	2781*
	TA57	3725 (26.57%)	3720 (26.4%)	3261 (10.81%)	3335 (13.32%)	3471 (17.94%)	**3180 (8.05%)**	2943*
	TA58	3365 (16.64%)	3926 (36.08%)	3365 (16.64%)	3420 (18.54%)	3732 (29.36%)	**3103 (7.56%)**	2885*
	TA59	3294 (24.07%)	3672 (38.31%)	3131 (17.93%)	3117 (17.4%)	3381 (27.34%)	**3009 (13.33%)**	2655*
	TA60	3500 (28.53%)	3783 (38.93%)	3122 (14.65%)	3044 (11.79%)	3352 (23.1%)	**2909 (6.83%)**	2723*
50×20	TA61	3606 (25.73%)	4142 (44.42%)	3343 (16.56%)	3376 (17.71%)	3654 (27.41%)	**3184 (11.02%)**	2868*
	TA62	3639 (26.84%)	3897 (35.83%)	3462 (20.67%)	3417 (19.1%)	3722 (29.73%)	**3229 (12.55%)**	2869*
	TA63	3521 (27.8%)	3852 (39.82%)	3233 (17.35%)	3276 (18.91%)	3536 (28.35%)	**3004 (9.04%)**	2755*
	TA64	3447 (27.57%)	4001 (48.08%)	3188 (17.99%)	3057 (13.14%)	3631 (34.38%)	**2916 (7.92%)**	2702*
	TA65	3332 (22.28%)	4062 (49.06%)	3429 (25.83%)	3249 (19.23%)	3359 (23.27%)	**3068 (12.59%)**	2725*
	TA66	3677 (29.24%)	3940 (38.49%)	3287 (15.54%)	3335 (17.22%)	3555 (24.96%)	**3144 (10.51%)**	2845*
	TA67	3487 (23.43%)	3974 (40.67%)	3351 (18.62%)	3392 (20.07%)	3567 (26.27%)	**3120 (10.44%)**	2825*
	TA68	3336 (19.83%)	3857 (38.54%)	3203 (15.05%)	3251 (16.77%)	3680 (32.18%)	**2955 (6.14%)**	2784*
	TA69	3862 (25.76%)	4349 (41.62%)	3550 (15.6%)	3526 (14.82%)	3592 (16.97%)	**3295 (7.29%)**	3071*
	TA70	3801 (26.91%)	4147 (38.46%)	3482 (16.26%)	3590 (19.87%)	3643 (21.64%)	**3336 (11.39%)**	2995*
100×20	TA71	6232 (14.06%)	6818 (24.78%)	6036 (10.47%)	5938 (8.67%)	6452 (18.08%)	**5653 (3.46%)**	5464*
	TA72	5973 (15.29%)	6358 (22.72%)	5583 (7.76%)	5639 (8.84%)	5695 (9.92%)	**5365 (3.55%)**	5181*
	TA73	6482 (16.42%)	6967 (25.13%)	6050 (8.66%)	6128 (10.06%)	6462 (16.06%)	**5885 (5.69%)**	5568*
	TA74	6062 (13.54%)	6381 (19.52%)	5678 (6.35%)	5642 (5.68%)	5885 (10.23%)	**5427 (1.65%)**	5339*
	TA75	6217 (15.3%)	6757 (25.32%)	6029 (11.81%)	6212 (15.21%)	6355 (17.86%)	**5790 (7.38%)**	5392*
	TA76	6370 (19.24%)	6641 (24.32%)	5887 (10.2%)	5936 (11.12%)	6135 (14.84%)	**5748 (7.6%)**	5342*
	TA77	6045 (11.2%)	6540 (20.31%)	5905 (8.63%)	5829 (7.23%)	6056 (11.41%)	**5588 (2.8%)**	5436*
	TA78	6143 (13.89%)	6750 (25.14%)	5700 (5.67%)	5886 (9.12%)	6101 (13.11%)	**5570 (3.26%)**	5394*
	TA79	6018 (12.32%)	6461 (20.59%)	5749 (7.3%)	5652 (5.49%)	5943 (10.92%)	**5424 (1.23%)**	5358*
	TA80	5848 (12.83%)	6534 (26.07%)	5505 (6.21%)	5707 (10.11%)	5892 (13.68%)	**5339 (3.01%)**	5183*

Table 2.6: Results on DMU instances.

	Instance	SPT	FDD/WKR	MWKR	MOPNR	[233]	RASCL (s=128)	UB
20×15	DMU1	4516 (76.2%)	3535 (37.92%)	3988 (55.6%)	3882 (51.46%)	3323 (29.65%)	**3107 (21.23%)**	2563
	DMU2	4593 (69.73%)	3847 (42.17%)	4555 (68.33%)	3884 (43.53%)	3630 (34.15%)	**3272 (20.92%)**	2706
	DMU3	4438 (62.5%)	4063 (48.77%)	4117 (50.75%)	3979 (45.7%)	3660 (34.02%)	**3270 (19.74%)**	2731*
	DMU4	4533 (69.84%)	4160 (55.86%)	3995 (49.68%)	4079 (52.83%)	3816 (42.97%)	**3138 (17.57%)**	2669
	DMU5	4420 (60.79%)	4238 (54.17%)	4977 (81.05%)	4116 (49.73%)	3897 (41.76%)	**3290 (19.68%)**	2749*
	DMU41	5283 (62.65%)	5187 (59.7%)	5377 (65.55%)	5070 (56.1%)	4316 (32.88%)	**3807 (17.21%)**	3248
	DMU42	5354 (57.94%)	5583 (64.69%)	6076 (79.23%)	4976 (46.78%)	4858 (43.3%)	**4027 (18.79%)**	3390
	DMU43	5328 (54.84%)	5086 (47.81%)	4938 (43.5%)	5012 (45.66%)	4887 (42.02%)	**4159 (20.87%)**	3441
	DMU44	5745 (64.71%)	5550 (59.12%)	5630 (61.41%)	5213 (49.46%)	5151 (47.68%)	**4109 (17.8%)**	3488
	DMU45	5305 (62.13%)	5414 (65.46%)	5446 (66.44%)	4921 (50.4%)	4615 (41.05%)	**3921 (19.83%)**	3272
20×20	DMU6	6230 (92.05%)	5258 (62.08%)	5556 (71.27%)	4747 (46.33%)	4358 (34.34%)	**3830 (18.06%)**	3244
	DMU7	5619 (84.47%)	4789 (57.22%)	4636 (52.2%)	4367 (43.37%)	3671 (20.52%)	**3488 (14.51%)**	3046
	DMU8	5239 (64.34%)	4817 (51.1%)	5078 (59.28%)	4480 (40.53%)	4048 (26.98%)	**3716 (16.56%)**	3188
	DMU9	4874 (57.63%)	4675 (51.2%)	4519 (46.15%)	4519 (46.15%)	4482 (44.95%)	**3542 (14.55%)**	3092
	DMU10	4808 (61.13%)	4149 (39.04%)	4963 (66.32%)	4133 (38.51%)	4021 (34.75%)	**3420 (14.61%)**	2984
	DMU46	6403 (58.69%)	5778 (43.2%)	6168 (52.86%)	6136 (52.07%)	5876 (45.63%)	**4611 (14.28%)**	4035
	DMU47	6015 (52.7%)	6058 (53.8%)	6130 (55.62%)	5908 (49.99%)	5771 (46.51%)	**4521 (14.78%)**	3939
	DMU48	5345 (42.04%)	5887 (56.44%)	5701 (51.5%)	5384 (43.08%)	5034 (33.78%)	**4366 (16.02%)**	3763
	DMU49	6072 (63.67%)	5807 (56.52%)	6089 (64.12%)	5469 (47.41%)	5470 (47.44%)	**4368 (17.74%)**	3710
	DMU50	6300 (68.95%)	5764 (54.57%)	6050 (62.24%)	5380 (44.27%)	5314 (42.5%)	**4427 (18.72%)**	3729
30×15	DMU11	5864 (70.96%)	4798 (39.88%)	4961 (44.64%)	4891 (42.59%)	4435 (29.3%)	**4031 (17.52%)**	3430
	DMU12	5966 (70.7%)	5595 (60.09%)	5994 (71.5%)	4947 (41.55%)	4864 (39.17%)	**4033 (15.39%)**	3495
	DMU13	5744 (56.04%)	5324 (44.63%)	6190 (68.16%)	4979 (35.26%)	4918 (33.6%)	**4273 (16.08%)**	3681*
	DMU14	5469 (61.14%)	4830 (42.31%)	5567 (64.02%)	4839 (42.58%)	4130 (21.69%)	**3862 (13.79%)**	3394*
	DMU15	5518 (65.06%)	4928 (47.41%)	5299 (58.51%)	4653 (39.19%)	4392 (31.38%)	**3913 (17.05%)**	3343*
	DMU51	6538 (56.9%)	7002 (68.03%)	6841 (64.17%)	6691 (60.57%)	6241 (49.77%)	**4745 (13.87%)**	4167
	DMU52	7341 (70.29%)	6650 (54.26%)	6942 (61.03%)	6591 (52.89%)	6714 (55.74%)	**5083 (17.91%)**	4311
	DMU53	7232 (64.59%)	7170 (63.18%)	7430 (69.09%)	6851 (55.92%)	6724 (53.03%)	**5147 (17.14%)**	4394
	DMU54	7178 (64.56%)	6767 (55.14%)	6461 (48.12%)	6540 (49.93%)	6522 (49.52%)	**5133 (17.68%)**	4362
	DMU55	6212 (45.45%)	7101 (66.26%)	6844 (60.24%)	6446 (50.92%)	6639 (55.44%)	**5000 (17.07%)**	4271
30×20	DMU16	6241 (66.38%)	5948 (58.57%)	5837 (55.61%)	5743 (53.11%)	4953 (32.04%)	**4537 (20.95%)**	3751
	DMU17	6487 (70.08%)	6035 (58.23%)	6610 (73.31%)	5540 (45.25%)	5379 (41.03%)	**4657 (22.1%)**	3814
	DMU18	6978 (81.53%)	5863 (52.52%)	6363 (65.53%)	5714 (48.65%)	5100 (32.67%)	**4685 (21.88%)**	3844*
	DMU19	5767 (53.05%)	5424 (43.95%)	6385 (69.45%)	5223 (38.61%)	4889 (29.75%)	**4444 (17.94%)**	3768
	DMU20	6910 (86.25%)	6444 (73.69%)	6472 (74.45%)	5530 (49.06%)	4859 (30.97%)	**4443 (19.76%)**	3710
	DMU56	7698 (55.8%)	8248 (66.93%)	7930 (60.49%)	7620 (54.22%)	7328 (48.31%)	**5926 (19.94%)**	4941
	DMU57	7746 (66.4%)	7694 (65.28%)	7063 (51.73%)	7345 (57.79%)	6704 (44.02%)	**5493 (18.0%)**	4655
	DMU58	7269 (54.4%)	7601 (61.45%)	7708 (63.72%)	7216 (53.27%)	6721 (42.76%)	**5566 (18.22%)**	4708
	DMU59	7114 (53.85%)	7490 (61.98%)	7335 (58.63%)	7589 (64.12%)	7109 (53.74%)	**5659 (22.38%)**	4624
	DMU60	8150 (71.4%)	7526 (58.28%)	7547 (58.72%)	7399 (55.6%)	6632 (39.47%)	**5650 (18.82%)**	4755
40×15	DMU21	7400 (68.95%)	6416 (46.48%)	6314 (44.16%)	6048 (38.08%)	5317 (21.39%)	**5254 (19.95%)**	4380*
	DMU22	7353 (55.62%)	6645 (40.63%)	6980 (47.72%)	6351 (34.41%)	5534 (17.12%)	**5495 (16.3%)**	4725*
	DMU23	7262 (55.57%)	6781 (45.27%)	6472 (38.65%)	6004 (28.62%)	5620 (20.39%)	**5421 (16.13%)**	4668*
	DMU24	6799 (46.28%)	6582 (41.61%)	7079 (52.3%)	6155 (32.42%)	5753 (23.77%)	**5441 (17.06%)**	4648*
	DMU25	6428 (54.37%)	5756 (38.23%)	6042 (45.1%)	5365 (28.84%)	**4775 (14.67%)**	4872 (17.0%)	4164*
	DMU61	7817 (51.14%)	8757 (69.32%)	8734 (68.87%)	8076 (56.15%)	8203 (58.6%)	**6081 (17.58%)**	5172
	DMU62	7759 (47.37%)	8082 (53.5%)	8262 (56.92%)	8253 (56.75%)	8091 (53.68%)	**6167 (17.13%)**	5265
	DMU63	8296 (55.76%)	8384 (57.42%)	8364 (57.04%)	8417 (58.04%)	8031 (50.79%)	**6301 (18.31%)**	5326
	DMU64	8444 (60.84%)	8490 (61.71%)	8406 (60.11%)	8161 (55.45%)	7738 (47.39%)	**6087 (15.94%)**	5250
	DMU65	8454 (62.89%)	8307 (60.06%)	8189 (57.78%)	8225 (58.48%)	7577 (45.99%)	**6200 (19.46%)**	5190
40×20	DMU26	7766 (67.12%)	7240 (55.8%)	7107 (52.94%)	6236 (34.19%)	5946 (27.95%)	**5771 (24.19%)**	4647*
	DMU27	7501 (54.72%)	6965 (43.67%)	7313 (50.85%)	6936 (43.07%)	6418 (32.38%)	**6177 (27.41%)**	4848*
	DMU28	8621 (83.74%)	6516 (38.87%)	8194 (74.64%)	6714 (43.09%)	5986 (27.58%)	**5976 (27.37%)**	4692*
	DMU29	8052 (71.65%)	6971 (48.6%)	7448 (58.77%)	6990 (49.01%)	6051 (28.99%)	**5767 (22.94%)**	4691*
	DMU30	7372 (55.79%)	6910 (46.03%)	7890 (66.74%)	6869 (45.16%)	**5988 (26.54%)**	6015 (27.11%)	4732*
	DMU66	8971 (56.92%)	9606 (68.03%)	8966 (56.83%)	8726 (52.63%)	8475 (48.24%)	**7217 (26.24%)**	5717
	DMU67	9096 (56.48%)	9103 (56.6%)	9306 (60.09%)	9372 (61.22%)	8832 (51.94%)	**7152 (23.03%)**	5813
	DMU68	9265 (60.49%)	9431 (63.36%)	9445 (63.61%)	8722 (51.08%)	8693 (50.58%)	**7181 (24.39%)**	5773
	DMU69	9215 (61.41%)	9951 (74.3%)	9450 (65.53%)	8697 (52.34%)	8634 (51.23%)	**7175 (25.68%)**	5709
	DMU70	9522 (61.69%)	9416 (59.89%)	9490 (61.15%)	9445 (60.38%)	8735 (48.33%)	**7410 (25.83%)**	5889
50×15	DMU31	8869 (57.25%)	7899 (40.05%)	8147 (44.45%)	7192 (27.52%)	7156 (26.88%)	**6895 (22.25%)**	5640*
	DMU32	7814 (31.84%)	7316 (23.44%)	8004 (35.04%)	7267 (22.61%)	**6506 (9.77%)**	7231 (22.0%)	5927*
	DMU33	8114 (41.66%)	7262 (26.78%)	7710 (34.6%)	7069 (23.41%)	**6192 (8.1%)**	7076 (23.53%)	5728*
	DMU34	7625 (41.6%)	7725 (43.45%)	7709 (43.16%)	6919 (28.49%)	**6257 (16.19%)**	6598 (22.53%)	5385*
	DMU35	8626 (53.08%)	7099 (25.98%)	7617 (35.17%)	7033 (24.81%)	**6302 (11.84%)**	6906 (22.56%)	5635*
	DMU71	9594 (53.92%)	10889 (74.7%)	9978 (60.08%)	9514 (52.64%)	9797 (57.18%)	**7441 (19.38%)**	6233
	DMU72	9882 (52.43%)	11602 (78.96%)	10135 (56.33%)	10063 (55.22%)	9926 (53.11%)	**7864 (21.3%)**	6483
	DMU73	9953 (61.5%)	10212 (65.7%)	9721 (57.73%)	9615 (56.01%)	9933 (61.17%)	**7591 (23.17%)**	6163
	DMU74	9866 (58.62%)	10659 (71.37%)	10086 (62.15%)	9536 (53.31%)	9833 (58.09%)	**7398 (18.94%)**	6220
	DMU75	9411 (51.86%)	10839 (74.91%)	9953 (60.61%)	10157 (63.9%)	9892 (59.63%)	**7421 (19.75%)**	6197
50×20	DMU36	9911 (76.32%)	8084 (43.82%)	8090 (43.92%)	7703 (37.04%)	7470 (32.89%)	**6339 (12.77%)**	5621*
	DMU37	8917 (52.4%)	9433 (61.22%)	9685 (65.53%)	7844 (34.06%)	7296 (24.7%)	**6815 (16.48%)**	5851*
	DMU38	9384 (64.26%)	8428 (47.52%)	8414 (47.28%)	8398 (47.0%)	7410 (29.7%)	**6424 (12.45%)**	5713*
	DMU39	9221 (60.45%)	8177 (42.28%)	9266 (61.23%)	7969 (38.66%)	6827 (18.79%)	**6592 (14.7%)**	5747*
	DMU40	9406 (68.66%)	7773 (39.38%)	8261 (48.13%)	8173 (46.55%)	7325 (31.34%)	**6398 (14.72%)**	5577*
	DMU76	11677 (71.39%)	11576 (69.91%)	10571 (55.16%)	11019 (61.73%)	9698 (42.35%)	**7905 (16.03%)**	6813
	DMU77	10401 (52.46%)	11910 (74.58%)	11148 (63.41%)	10577 (55.04%)	10693 (56.74%)	**7773 (13.94%)**	6822
	DMU78	10585 (56.35%)	11464 (69.34%)	10540 (55.69%)	10989 (62.32%)	9986 (47.5%)	**7912 (16.87%)**	6770
	DMU79	11115 (59.47%)	11035 (58.32%)	11201 (60.7%)	10729 (53.93%)	10936 (56.9%)	**7925 (13.7%)**	6970
	DMU80	10711 (60.2%)	11116 (66.26%)	10894 (62.94%)	10679 (59.72%)	9875 (47.7%)	**7686 (14.96%)**	6686

Table 2.7: Curriculum strategies on Taillard's instances.

	Instance	[233]	**Base** (s=128)	**ICL** (s=128)	**UCL** (s=128)	**ASCL** (s=128)	**RASCL** (s=128)	UB
15×15	TA1	1443 (17.22%)	1475 (19.82%)	1475 (19.82%)	1393 (13.16%)	1387 (12.67%)	**1379 (12.02%)**	1231*
	TA2	1544 (24.12%)	1401 (12.62%)	1401 (12.62%)	1391 (11.82%)	1336 (7.4%)	**1325 (6.51%)**	1244*
	TA3	1440 (18.23%)	1393 (14.37%)	1393 (14.37%)	1408 (15.6%)	1341 (10.1%)	**1338 (9.85%)**	1218*
	TA4	1637 (39.32%)	1340 (14.04%)	1340 (14.04%)	1404 (19.49%)	1305 (11.06%)	**1275 (8.51%)**	1175*
	TA5	1619 (32.27%)	1390 (13.56%)	1390 (13.56%)	1407 (14.95%)	1329 (8.58%)	**1303 (6.45%)**	1224*
	TA6	1601 (29.32%)	1388 (12.12%)	1388 (12.12%)	1375 (11.07%)	**1309 (5.74%)**	1325 (7.03%)	1238*
	TA7	1568 (27.79%)	1425 (16.14%)	1425 (16.14%)	1398 (13.94%)	**1321 (7.66%)**	1332 (8.56%)	1227*
	TA8	1468 (20.62%)	1407 (15.61%)	1407 (15.61%)	1409 (15.78%)	**1316 (8.13%)**	1325 (8.87%)	1217*
	TA9	1627 (27.71%)	1472 (15.54%)	1472 (15.54%)	1478 (16.01%)	**1390 (9.11%)**	1412 (10.83%)	1274*
	TA10	1527 (23.05%)	1439 (15.95%)	1439 (15.95%)	1398 (12.65%)	**1332 (7.33%)**	1384 (11.52%)	1241*
20×15	TA11	1794 (32.2%)	1671 (23.14%)	1576 (16.14%)	1652 (21.74%)	1538 (13.34%)	**1506 (10.98%)**	1357*
	TA12	1805 (32.04%)	1678 (22.75%)	1596 (16.75%)	1589 (16.24%)	**1498 (9.58%)**	1514 (10.75%)	1367*
	TA13	1932 (43.86%)	1691 (25.91%)	1569 (16.83%)	1634 (21.67%)	1543 (14.89%)	**1510 (12.43%)**	1343*
	TA14	1664 (23.72%)	1685 (25.28%)	1535 (14.13%)	1564 (16.28%)	1488 (10.63%)	**1455 (8.18%)**	1345*
	TA15	1730 (29.2%)	1675 (25.09%)	1554 (16.06%)	1621 (21.06%)	**1492 (11.43%)**	1516 (13.22%)	1339*
	TA16	1710 (25.74%)	1689 (24.19%)	1582 (16.32%)	1657 (21.84%)	1558 (14.56%)	**1485 (9.19%)**	1360*
	TA17	1897 (29.75%)	1800 (23.12%)	1705 (16.62%)	1732 (18.47%)	1666 (13.95%)	**1614 (10.4%)**	1462*
	TA18	1794 (28.51%)	1785 (27.87%)	1662 (19.05%)	1645 (17.84%)	1634 (17.05%)	**1560 (11.75%)**	1396
	TA19	1682 (26.28%)	1615 (21.25%)	1544 (15.92%)	1609 (20.8%)	1526 (14.56%)	**1451 (8.93%)**	1332*
	TA20	1739 (29.01%)	1632 (21.07%)	1558 (15.58%)	1574 (16.77%)	1519 (12.69%)	**1482 (9.94%)**	1348*
20×20	TA21	2252 (37.15%)	1944 (18.39%)	1967 (19.79%)	1937 (17.97%)	1861 (13.34%)	**1838 (11.94%)**	1642*
	TA22	2102 (31.37%)	1937 (21.06%)	1865 (16.56%)	1875 (17.19%)	1769 (10.56%)	**1752 (9.5%)**	1600
	TA23	2085 (33.91%)	1903 (22.22%)	1870 (20.1%)	1853 (19.01%)	**1745 (12.07%)**	1748 (12.27%)	1557
	TA24	2200 (33.82%)	1958 (19.1%)	1898 (15.45%)	1916 (16.55%)	1818 (10.58%)	**1807 (9.91%)**	1644*
	TA25	2201 (37.99%)	2006 (25.77%)	1866 (16.99%)	1900 (19.12%)	1807 (13.29%)	**1772 (11.1%)**	1595
	TA26	2176 (32.44%)	1971 (19.96%)	1928 (17.35%)	1943 (18.26%)	1810 (10.16%)	**1802 (9.68%)**	1643
	TA27	2132 (26.9%)	2066 (22.98%)	1980 (17.86%)	1939 (15.42%)	1900 (13.1%)	**1895 (12.8%)**	1680
	TA28	2146 (33.87%)	1924 (20.02%)	1841 (14.85%)	1814 (13.16%)	1798 (12.16%)	**1788 (11.54%)**	1603*
	TA29	1952 (20.12%)	1926 (18.52%)	1856 (14.22%)	1864 (14.71%)	1796 (10.52%)	**1753 (7.88%)**	1625
	TA30	2035 (28.47%)	1949 (23.04%)	1852 (16.92%)	1880 (18.69%)	1794 (13.26%)	**1776 (12.12%)**	1584
30×15	TA31	2565 (45.41%)	2263 (28.29%)	2149 (21.83%)	2172 (23.13%)	2072 (17.46%)	**2028 (14.97%)**	1764*
	TA32	2388 (33.86%)	2301 (28.98%)	2216 (24.22%)	2260 (26.68%)	**2072 (16.14%)**	2092 (17.26%)	1784
	TA33	2324 (29.76%)	2410 (34.56%)	2350 (31.21%)	2295 (28.14%)	2185 (22.0%)	**2114 (18.03%)**	1791
	TA34	2332 (27.57%)	2317 (26.75%)	2166 (18.49%)	2180 (19.26%)	2086 (14.11%)	**2077 (13.62%)**	1828*
	TA35	2505 (24.81%)	2389 (19.03%)	2235 (11.36%)	2257 (12.46%)	2141 (6.68%)	**2103 (4.78%)**	2007*
	TA36	2497 (37.27%)	2326 (27.87%)	2248 (23.58%)	2235 (22.87%)	2072 (13.91%)	**2046 (12.48%)**	1819*
	TA37	2325 (31.28%)	2329 (31.51%)	2251 (27.1%)	2197 (24.05%)	2098 (18.46%)	**2080 (17.45%)**	1771*
	TA38	2302 (37.6%)	2108 (26.0%)	2036 (21.7%)	2071 (23.79%)	1926 (15.12%)	**1882 (12.49%)**	1673*
	TA39	2410 (34.26%)	2226 (24.01%)	2194 (22.23%)	2119 (18.05%)	2056 (14.54%)	**2029 (13.04%)**	1795*
	TA40	2140 (28.22%)	2084 (24.87%)	2032 (21.75%)	2007 (20.25%)	1973 (18.21%)	**1930 (15.64%)**	1669
30×20	TA41	2667 (33.02%)	2604 (29.88%)	2543 (26.83%)	2482 (23.79%)	2397 (19.55%)	**2356 (17.51%)**	2005
	TA42	2664 (37.53%)	2534 (30.82%)	2399 (23.85%)	2416 (24.73%)	**2235 (15.38%)**	2260 (16.68%)	1937
	TA43	2431 (31.69%)	2445 (32.45%)	2332 (26.33%)	2354 (27.52%)	2194 (18.85%)	**2163 (17.17%)**	1846
	TA44	2714 (37.14%)	2671 (34.97%)	2506 (26.63%)	2518 (27.24%)	2323 (17.38%)	**2293 (15.87%)**	1979
	TA45	2637 (31.85%)	2552 (27.6%)	2470 (23.5%)	2443 (22.15%)	2289 (14.45%)	**2250 (12.5%)**	2000
	TA46	2776 (38.38%)	2615 (30.36%)	2569 (28.07%)	2515 (25.37%)	2341 (16.7%)	**2314 (15.35%)**	2006
	TA47	2476 (31.07%)	2468 (30.65%)	2398 (26.95%)	2323 (22.98%)	2200 (16.46%)	**2141 (13.34%)**	1889
	TA48	2490 (28.55%)	2495 (28.81%)	2471 (27.57%)	2396 (23.7%)	**2285 (17.97%)**	2289 (18.17%)	1937
	TA49	2556 (30.34%)	2496 (27.28%)	2428 (23.81%)	2329 (18.77%)	**2266 (15.55%)**	2270 (15.76%)	1961
	TA50	2628 (36.66%)	2522 (31.15%)	2434 (26.57%)	2419 (25.79%)	2280 (18.56%)	**2279 (18.51%)**	1923
50×15	TA51	3599 (30.4%)	3551 (28.66%)	3394 (22.97%)	3357 (21.63%)	3303 (19.67%)	**3181 (15.25%)**	2760*
	TA52	3341 (21.23%)	3428 (24.38%)	3337 (21.08%)	3254 (18.07%)	3106 (12.7%)	**3029 (9.91%)**	2756*
	TA53	3186 (17.26%)	3164 (16.45%)	3122 (14.91%)	3035 (11.7%)	2942 (8.28%)	**2921 (7.51%)**	2717*
	TA54	3266 (15.04%)	3310 (16.59%)	3163 (11.41%)	3134 (10.39%)	3012 (6.09%)	**2914 (2.64%)**	2839*
	TA55	3232 (20.64%)	3259 (21.65%)	3247 (21.2%)	3109 (16.05%)	3065 (14.41%)	**3050 (13.85%)**	2679*
	TA56	3378 (21.47%)	3314 (19.17%)	3290 (18.3%)	3150 (13.27%)	3081 (10.79%)	**3012 (8.31%)**	2781*
	TA57	3471 (17.94%)	3504 (19.06%)	3437 (16.79%)	3326 (13.01%)	3249 (10.4%)	**3180 (8.05%)**	2943*
	TA58	3732 (29.36%)	3433 (18.99%)	3334 (15.56%)	3321 (15.11%)	3166 (9.74%)	**3103 (7.56%)**	2885*
	TA59	3381 (27.34%)	3303 (24.41%)	3196 (20.38%)	3152 (18.72%)	3025 (13.94%)	**3009 (13.33%)**	2655*
	TA60	3352 (23.1%)	3201 (17.55%)	3098 (13.77%)	3037 (11.53%)	2932 (7.68%)	**2909 (6.83%)**	2723*
50×20	TA61	3654 (27.41%)	3622 (26.29%)	3495 (21.86%)	3443 (20.05%)	3221 (12.31%)	**3184 (11.02%)**	2868*
	TA62	3722 (29.73%)	3615 (26.0%)	3496 (21.85%)	3428 (19.48%)	3274 (14.12%)	**3229 (12.55%)**	2869*
	TA63	3536 (28.35%)	3345 (21.42%)	3232 (17.31%)	3163 (14.81%)	3040 (10.34%)	**3004 (9.04%)**	2755*
	TA64	3631 (34.38%)	3332 (23.32%)	3164 (17.1%)	3180 (17.69%)	2933 (8.55%)	**2916 (7.92%)**	2702*
	TA65	3359 (23.27%)	3384 (24.18%)	3314 (21.61%)	3250 (19.27%)	3140 (15.23%)	**3068 (12.59%)**	2725*
	TA66	3555 (24.96%)	3505 (23.2%)	3415 (20.04%)	3334 (17.19%)	**3144 (10.51%)**	3144 (10.51%)	2845*
	TA67	3567 (26.27%)	3640 (28.85%)	3442 (21.84%)	3449 (22.09%)	3148 (11.43%)	**3120 (10.44%)**	2825*
	TA68	3680 (32.18%)	3374 (21.19%)	3174 (14.01%)	3193 (14.69%)	2999 (7.72%)	**2955 (6.14%)**	2784*
	TA69	3592 (16.97%)	3645 (18.69%)	3559 (15.89%)	3501 (14.0%)	3338 (8.69%)	**3295 (7.29%)**	3071*
	TA70	3643 (21.64%)	3718 (24.14%)	3553 (18.63%)	3540 (18.2%)	3357 (12.09%)	**3336 (11.39%)**	2995*
100×20	TA71	6452 (18.08%)	6384 (16.84%)	6238 (14.17%)	5985 (9.54%)	5798 (6.11%)	**5653 (3.46%)**	5464*
	TA72	5695 (9.92%)	5863 (13.16%)	5771 (11.39%)	5556 (7.24%)	5439 (4.98%)	**5365 (3.55%)**	5181*
	TA73	6462 (16.06%)	6417 (15.25%)	6294 (13.04%)	6054 (8.73%)	5997 (7.7%)	**5885 (5.69%)**	5568*
	TA74	5885 (10.23%)	6018 (12.72%)	5859 (9.74%)	5727 (7.27%)	5518 (3.35%)	**5427 (1.65%)**	5339*
	TA75	6355 (17.86%)	6466 (19.92%)	6294 (16.73%)	6154 (14.13%)	5918 (9.76%)	**5790 (7.38%)**	5392*
	TA76	6135 (14.84%)	6112 (14.41%)	6060 (13.44%)	5886 (10.18%)	**5714 (6.96%)**	5748 (7.6%)	5342*
	TA77	6056 (11.41%)	5999 (10.36%)	5935 (9.18%)	5716 (5.15%)	5655 (4.03%)	**5588 (2.8%)**	5436*
	TA78	6101 (13.11%)	6294 (16.69%)	6153 (14.07%)	5941 (10.14%)	5735 (6.32%)	**5570 (3.26%)**	5394*
	TA79	5943 (10.92%)	5995 (11.89%)	5945 (10.96%)	5773 (7.75%)	5567 (3.9%)	**5424 (1.23%)**	5358*
	TA80	5892 (13.68%)	5907 (13.97%)	5819 (12.27%)	5639 (8.8%)	5480 (5.73%)	**5339 (3.01%)**	5183*

Table 2.8: Inference strategies for RASCL on Taillard's instances.

	Instance	Greedy	POMO (p=3)	Sampling (s=128)	Beam (b=3)	UB
15×15	TA1	1445 (17.38%)	1445 (17.38%)	**1379 (12.02%)**	1421 (15.43%)	1231*
	TA2	1405 (12.94%)	1405 (12.94%)	**1325 (6.51%)**	1390 (11.74%)	1244*
	TA3	1469 (20.61%)	1344 (10.34%)	**1338 (9.85%)**	1405 (15.35%)	1218*
	TA4	1304 (10.98%)	1304 (10.98%)	**1275 (8.51%)**	1304 (10.98%)	1175*
	TA5	1345 (9.89%)	1345 (9.89%)	**1303 (6.45%)**	1326 (8.33%)	1224*
	TA6	1360 (9.85%)	1329 (7.35%)	**1325 (7.03%)**	1326 (7.11%)	1238*
	TA7	1426 (16.22%)	1426 (16.22%)	**1332 (8.56%)**	1400 (14.1%)	1227*
	TA8	1422 (16.84%)	1355 (11.34%)	**1325 (8.87%)**	1391 (14.3%)	1217*
	TA9	1460 (14.6%)	1460 (14.6%)	**1412 (10.83%)**	1460 (14.6%)	1274*
	TA10	1406 (13.3%)	1406 (13.3%)	**1384 (11.52%)**	1406 (13.3%)	1241*
20×15	TA11	1623 (19.6%)	1623 (19.6%)	**1506 (10.98%)**	1627 (19.9%)	1357*
	TA12	1587 (16.09%)	1587 (16.09%)	**1514 (10.75%)**	1541 (12.73%)	1367*
	TA13	1617 (20.4%)	1617 (20.4%)	**1510 (12.43%)**	1614 (20.18%)	1343*
	TA14	1548 (15.09%)	1548 (15.09%)	**1455 (8.18%)**	1548 (15.09%)	1345*
	TA15	1546 (15.46%)	1546 (15.46%)	**1516 (13.22%)**	1517 (13.29%)	1339*
	TA16	1584 (16.47%)	1584 (16.47%)	**1485 (9.19%)**	1584 (16.47%)	1360*
	TA17	1698 (16.14%)	1698 (16.14%)	**1614 (10.4%)**	1698 (16.14%)	1462*
	TA18	1614 (15.62%)	1597 (14.4%)	**1560 (11.75%)**	1614 (15.62%)	1396
	TA19	1561 (17.19%)	1561 (17.19%)	**1451 (8.93%)**	1561 (17.19%)	1332*
	TA20	1525 (13.13%)	1525 (13.13%)	**1482 (9.94%)**	1525 (13.13%)	1348*
20×20	TA21	1913 (16.5%)	1913 (16.5%)	**1838 (11.94%)**	1913 (16.5%)	1642*
	TA22	1853 (15.81%)	1853 (15.81%)	**1752 (9.5%)**	1811 (13.19%)	1600
	TA23	1854 (19.08%)	1837 (17.98%)	**1748 (12.27%)**	1898 (21.9%)	1557
	TA24	1907 (16.0%)	1904 (15.82%)	**1807 (9.91%)**	1905 (15.88%)	1644*
	TA25	1903 (19.31%)	1903 (19.31%)	**1772 (11.1%)**	1922 (20.5%)	1595
	TA26	1944 (18.32%)	1898 (15.52%)	**1802 (9.68%)**	1944 (18.32%)	1643
	TA27	2001 (19.11%)	2001 (19.11%)	**1895 (12.8%)**	1998 (18.93%)	1680
	TA28	1826 (13.91%)	1826 (13.91%)	**1788 (11.54%)**	1810 (12.91%)	1603*
	TA29	1839 (13.17%)	1839 (13.17%)	**1753 (7.88%)**	1802 (10.89%)	1625
	TA30	1925 (21.53%)	1901 (20.01%)	**1776 (12.12%)**	1918 (21.09%)	1584
30×15	TA31	2105 (19.33%)	2105 (19.33%)	**2028 (14.97%)**	2092 (18.59%)	1764*
	TA32	2163 (21.24%)	2163 (21.24%)	**2092 (17.26%)**	2183 (22.37%)	1784
	TA33	2225 (24.23%)	2175 (21.44%)	**2114 (18.03%)**	2203 (23.0%)	1791
	TA34	2108 (15.32%)	2108 (15.32%)	**2077 (13.62%)**	2108 (15.32%)	1828*
	TA35	2172 (8.22%)	2115 (5.38%)	**2103 (4.78%)**	2172 (8.22%)	2007*
	TA36	2124 (16.77%)	2092 (15.01%)	**2046 (12.48%)**	2124 (16.77%)	1819*
	TA37	2208 (24.68%)	2208 (24.68%)	**2080 (17.45%)**	2138 (20.72%)	1771*
	TA38	1943 (16.14%)	1943 (16.14%)	**1882 (12.49%)**	1928 (15.24%)	1673*
	TA39	2106 (17.33%)	2106 (17.33%)	**2029 (13.04%)**	2104 (17.21%)	1795*
	TA40	2035 (21.93%)	2035 (21.93%)	**1930 (15.64%)**	1996 (19.59%)	1669
30×20	TA41	2440 (21.7%)	2440 (21.7%)	**2356 (17.51%)**	2428 (21.1%)	2005
	TA42	2375 (22.61%)	2375 (22.61%)	**2260 (16.68%)**	2363 (21.99%)	1937
	TA43	2352 (27.41%)	2312 (25.24%)	**2163 (17.17%)**	2366 (28.17%)	1846
	TA44	2464 (24.51%)	2393 (20.92%)	**2293 (15.87%)**	2440 (23.29%)	1979
	TA45	**2248 (12.4%)**	2248 (12.4%)	2250 (12.5%)	2248 (12.4%)	2000
	TA46	2410 (20.14%)	2410 (20.14%)	**2314 (15.35%)**	2391 (19.19%)	2006
	TA47	2211 (17.05%)	2211 (17.05%)	**2141 (13.34%)**	2211 (17.05%)	1889
	TA48	2350 (21.32%)	2350 (21.32%)	**2289 (18.17%)**	2327 (20.13%)	1937
	TA49	2415 (23.15%)	2377 (21.21%)	**2270 (15.76%)**	2415 (23.15%)	1961
	TA50	2392 (24.39%)	2392 (24.39%)	**2279 (18.51%)**	2351 (22.26%)	1923
50×15	TA51	3295 (19.38%)	3214 (16.45%)	**3181 (15.25%)**	3279 (18.8%)	2760*
	TA52	3082 (11.83%)	3082 (11.83%)	**3029 (9.91%)**	3029 (9.91%)	2756*
	TA53	2944 (8.35%)	2935 (8.02%)	**2921 (7.51%)**	2945 (8.39%)	2717*
	TA54	2924 (2.99%)	2924 (2.99%)	2914 (2.64%)	**2904 (2.29%)**	2839*
	TA55	3108 (16.01%)	3108 (16.01%)	**3050 (13.85%)**	3184 (18.85%)	2679*
	TA56	3201 (15.1%)	3104 (11.61%)	**3012 (8.31%)**	3176 (14.2%)	2781*
	TA57	3315 (12.64%)	3224 (9.55%)	**3180 (8.05%)**	3247 (10.33%)	2943*
	TA58	3143 (8.94%)	3143 (8.94%)	**3103 (7.56%)**	3141 (8.87%)	2885*
	TA59	3100 (16.76%)	3091 (16.42%)	**3009 (13.33%)**	3049 (14.84%)	2655*
	TA60	3004 (10.32%)	2966 (8.92%)	**2909 (6.83%)**	2955 (8.52%)	2723*
50×20	TA61	3242 (13.04%)	**3179 (10.84%)**	3184 (11.02%)	3240 (12.97%)	2868*
	TA62	3309 (15.34%)	3292 (14.74%)	**3229 (12.55%)**	3286 (14.53%)	2869*
	TA63	3069 (11.4%)	3059 (11.03%)	**3004 (9.04%)**	3117 (13.14%)	2755*
	TA64	3044 (12.66%)	3010 (11.4%)	**2916 (7.92%)**	3020 (11.77%)	2702*
	TA65	3256 (19.49%)	3141 (15.27%)	**3068 (12.59%)**	3236 (18.75%)	2725*
	TA66	3249 (14.2%)	3224 (13.32%)	**3144 (10.51%)**	3243 (13.99%)	2845*
	TA67	3194 (13.06%)	3194 (13.06%)	**3120 (10.44%)**	3191 (12.96%)	2825*
	TA68	3040 (9.2%)	3030 (8.84%)	**2955 (6.14%)**	2992 (7.47%)	2784*
	TA69	3383 (10.16%)	3320 (8.11%)	**3295 (7.29%)**	3344 (8.89%)	3071*
	TA70	3410 (13.86%)	3410 (13.86%)	**3336 (11.39%)**	3365 (12.35%)	2995*
100×20	TA71	5769 (5.58%)	5694 (4.21%)	**5653 (3.46%)**	5709 (4.48%)	5464*
	TA72	5462 (5.42%)	5366 (3.57%)	**5365 (3.55%)**	5455 (5.29%)	5181*
	TA73	6052 (8.69%)	5982 (7.44%)	**5885 (5.69%)**	6040 (8.48%)	5568*
	TA74	5510 (3.2%)	5472 (2.49%)	5427 (1.65%)	**5352 (0.24%)**	5339*
	TA75	5850 (8.49%)	5833 (8.18%)	**5790 (7.38%)**	5837 (8.25%)	5392*
	TA76	5859 (9.68%)	5858 (9.66%)	**5748 (7.6%)**	5935 (11.1%)	5342*
	TA77	5677 (4.43%)	5657 (4.07%)	**5588 (2.8%)**	5663 (4.18%)	5436*
	TA78	5699 (5.65%)	5604 (3.89%)	5570 (3.26%)	**5541 (2.73%)**	5394*
	TA79	5548 (3.55%)	5484 (2.35%)	**5424 (1.23%)**	5488 (2.43%)	5358*
	TA80	5383 (3.86%)	5383 (3.86%)	**5339 (3.01%)**	5355 (3.32%)	5183*

Chapter 3

Learning Algorithm for Stochastic Vehicle Routing Problem

This chapter explores the application of Reinforcement Learning (RL) to address three significant variants of the classical Vehicle Routing Problem (VRP): the Stochastic Vehicle Routing Problem with Correlated Demands (SVRPCD), the Stochastic Vehicle Routing Problem with Travel Time (SVRPTT), and the Stochastic Vehicle Routing Problem with Time Windows (SVRPTW). Each variant introduces unique complexities that are prevalent in real-world logistics and transportation systems, including stochastic and correlated demands, uncertain travel times, and time window constraints for deliveries. For the SVRPCD, the chapter presents an end-to-end RL-based framework capable of learning demand correlations based on observable variables such as weather, enabling the generation of robust routing strategies that adapt to vehicle capacity and demand changes without human intervention. For the SVRPTT, the proposed RL framework incorporates dynamic routing policies that leverage external variables, such as temperature, pressure, and humidity, to estimate stochastic travel times, leading to more cost-effective and efficient routing. For the SVRPTW, the chapter develops an RL model that utilizes attention-based neural networks to optimize delivery routes while adhering to time window constraints and handling uncertainties in travel times and customer demands. Across all three variants, the RL frameworks outperform traditional metaheuristic and heuristic methods, including Ant Colony Optimization and Tabu Search, achieving significant reductions in travel times and improving transportation efficiency. Extensive experiments highlight the models' adaptability and scalability, demonstrating their potential to revolutionize how stochastic routing problems are approached in academia and industry. By introducing innovative use of external variables and enabling real-time reoptimization, these RL-based solutions provide a flexible and effective approach to tackling complex routing challenges, contributing to both operational efficiency and environmental sustainability.

3.1 Learning Algorithm for Stochastic Vehicle Routing Problem with Correlated Demands

3.1.1 Introduction

Reinforcement Learning (RL) is a subdiscipline of machine learning that aims to teach algorithms to make decisions by maximizing a predefined reward signal through interactions with their environment. RL has demonstrated remarkable success across a range of applications, including gaming [171] and robotics [2]. One of the recent areas of research on RL is its application to combinatorial optimization problems. These problems have traditionally been solved using either fast but locally optimal heuristics or globally optimal but slow solvers (e.g., Google OR tools). The application of RL to combinatorial optimization offers a middle ground, providing a potential balance between computational efficiency and optimality, while also offering the ability to learn black-box heuristics without human intervention. Despite these benefits, the effective use of RL for combinatorial optimization requires careful design of the environment and appropriate training settings.

The **Vehicle Routing Problem** (VRP) is a well-known example of a combinatorial optimization problem. It is a generalization of the classic Traveling Salesman Problem (TSP) and involves finding the optimal route for a fleet of vehicles to deliver goods to customers with specified demands and locations. The objective was to minimize the total travel cost. The first RL model capable of solving the VRP was introduced in 2018 [147]. The authors employed a pointer-network-based architecture to tackle complex combinatorial challenges.

The **Stochastic Vehicle Routing Problem** (SVRP) is a challenging optimization problem with many potential applications in industry ([28,51]). Unlike the deterministic VRP, some parameters in the SVRP are uncertain and can only be realized during the completion of the vehicle route. This formulation is more representative of real-world problems encountered in industry ([28,51]). Although classical solutions have been proposed for the SVRP ([52,107,161]), the potential of Deep Reinforcement Learning to address this task has not yet been explored.

The SVRP with **correlated stochastic demands** is widely acknowledged as a highly challenging task within the VRP research community ([165]). The solution to this problem has the potential to yield significant benefits in terms of cost and carbon emission reduction for companies involved in logistics and transportation. In this study, we aim to address the SVRP with correlated stochastic demands by developing a close-to-real-world formulation and solving it using an RL model.

Contributions. Our specific contributions to this field are summarized as follows:

- We propose a **novel formulation** for the Stochastic Vehicle Routing Problem with Correlated Demands (SVRPCD) that incorporates correlations among demands through observable stochastic variables. Our study focuses on optimizing a reinforcement learning model for this variant of VRP. Our proposed method exhibits an average performance **improvement of 3.26%** compared to the state-of-the-art model described in [52]. This advantage became more pronounced as the size of the problem increased.

- Our experimental results provide empirical evidence of the **superiority of non-i.i.d. environments**, a concept previously theorized in classical research. The state-of-the-art baseline model exhibited a 1% decrease in performance on instances with correlated demands, and this disparity increased with larger problem sizes. Conversely, our model demonstrated an **improvement of 2% in instances with correlated demands**, exhibiting consistent results across all problem sizes.

- Our model can learn the correlation between demands **without human intervention**, making it readily applicable to other SVRPCD settings. Through sensitivity experiments, we demonstrate the **robustness of the model** in response to environmental changes. This suggests the **potential for significant impacts on transportation costs and environmental sustainability** through additional research in this area. Further research could expand the scope of this study to incorporate additional sources of stochasticity, such as stochastic travel time and stochastic customers.

The subsequent sections of this work are organised as follows. In Section 3.1.2, we present an overview of the classical research avenues focused on addressing the SVRPCD, along with the state-of-the-art metaheuristic approach and previous endeavors to employ ML techniques for SVRPCD. Section 3.1.3 encompasses the classical problem formulation and provides definitions of the baseline models used in this study. Section 3.1.4 elaborates on our problem formulation, encompassing the routing policy employed to train the RL agent, as well as the sensitive factors that influence the proposed model. Subsequently, in Section 3.1.5, we provide a detailed description of the architecture employed. The experimental evaluation and assessment of robustness against sensitive factors are conducted in Section 3.1.6. Lastly, Section 3.1.7 offers the concluding remarks for this study.

3.1.2 Related Work

Several studies have applied RL techniques to address combinatorial optimization problems [8, 12, 31, 35, 39, 62, 63, 82, 92, 96, 147, 167, 181, 191, 217, 218, 225, 228, 236, 241, 243]. For example, [9] were pioneers in utilizing RL to solve Traveling Salesman (TSP) and Knapsack problems. They reported better performance compared to heuristics, achieving results that were close to globally optimal solutions. [151] applied a Q-learning approach to solve a Beer Game problem. Their model can also be extended to address decentralized multi-agent combinatorial problems. Furthermore, [193] demonstrated that RL techniques can find optimal solutions for simple combinatorial problems such as Knapsack, Secretary, and AdWords problems. These studies highlight the potential of RL for combinatorial optimization.

Recent advancements in deterministic VRP have leveraged RL techniques [19,29,33,42,60,81, 91,117–119,124,127,132,135,141,178,200,208,235,250]. The first RL-based method for solving the VRP was introduced in [147]. [131] proposed an RL agent as a metaheuristic algorithm that guides a set of simple rules in the search for VRP solutions. This model demonstrated the lowest travel cost for VRP instances with 50-100 customers. However, the method requires domain expertise from a researcher to construct a set of feasible heuristics. To address this limitation, [204] developed an end-to-end RL-based solver, which achieved state-of-the-art results for large-scale

VRP instances with 500, 1000, and 2000 customers. [89] introduced a novel hyperheuristic scheme called Bandit VNS. This scheme leverages the General Variable Neighborhood Search (VNS) metaheuristic to address deterministic VRP. Through experimental evaluations, the authors demonstrated that the Bandit VNS approach outperforms previous heuristics and RL methods in terms of solution quality. Specifically, for instances involving 32 to 120 customers and 5 to 25 vehicles, the proposed approach achieved a remarkable improvement of over 25%.

Despite extensive research on deterministic VRP utilizing RL algorithms [25, 45, 48, 59, 61, 72, 88, 109, 110, 113, 114, 120, 133, 140, 157, 175, 177, 187, 189, 196, 199, 202, 209, 221, 251, 252], the application of Machine Learning (ML) algorithms to SVRPCD remains limited. As discussed in [192], a significant body of literature exists only on classical algorithms for Vehicle Routing Problem with Stochastic Demands (SVRPCD), which are typically categorized into three main types:

1. **Branch-and-Bound** algorithms: Branch-and-Cut [49], Branch-and-Price [27], and Integer L-shaped [107].
2. **Heuristics**: LKH3 [64] and Clarke-Wright [161]
3. **Metaheuristics**: Tabu Search [112] and Ant-Colony Optimization [52]

Among Branch-and-Bound algorithms, Integer L-shaped [107] has demonstrated superior performance in solving SVRPCD. The algorithm replaces the cost with variable θ, which is used to compute a lower bound through the application of the branch-and-bound algorithm. However, the computational time of the Integer L-shaped algorithm exhibits an exponential dependence on the number of vehicles, limiting its practical application in industry contexts. The literature on VRP has seen extensive research on the use of heuristics algorithms, with LKH3 showing superior performance for deterministic problems ([64]), but giving unstable results for SVRP. By contrast, the Clarke-Wright algorithm [161] is a simple heuristic that has shown stable performance for both deterministic and stochastic problems. Its longevity and consistency in the field have made the Clarke-Wright algorithm a frequently used reference point for comparative purposes in SVRP literature. The application of metaheuristic algorithms has been prevalent in industry for the solution of VRP with stochastic demands with Ant-Colony Optimization (ACO), demonstrating state-of-the-art performance [52].

A separate subfield of literature on SVRPCD involves instances with correlated demands. To ensure the efficacy of classical methods for SVRPCD instances with non-i.i.d. demands, it is crucial to consider demand correlation in the design of the algorithm. For instance, the pheromone update formula in ACO should be adjusted to reflect the correlation between demands ([165]). Similarly, Branch-and-Bound algorithms must be modified to address this correlation [15]. In general, classical algorithms have been shown to exhibit inferior performance on this type of problem because of the assumption of i.i.d. demands in their design.

Theoretically, instances of the Vehicle Routing Problem with correlated stochastic demands have the potential to reduce travel times compared to uncorrelated instances. This is because the model can exploit the correlation to improve demand estimation, and as a result, implement a more effective routing policy. This theoretical advantage has been previously established in [15],

and its practical application was demonstrated in [165]. However, their approach required manual intervention to provide information about the demand probability density functions to the model, which limits its generalizability.

In recent years, a limited number of studies have applied ML algorithms to SVRP. In [168], a policy-based model was proposed to address the SVRP with stochastic demands, where the authors trained a simple linear model and achieved improved performance compared to heuristic methods. In [148], the authors utilized decision trees to solve the SVRP, whereas [149] employed an evolutionary algorithm based on a radial basis network. Furthermore, [84] was the first study to use deep reinforcement learning to address SVRP with stochastic customers.

SVRP can find applications in various research domains, including the Internet of Vehicles (IOV), which integrates vehicles with the Internet of Things network and Smart City solutions. [3] presents a new IOV architecture that utilizes RL techniques based on vehicle2vec embeddings. The experimental results demonstrate that this architecture significantly reduces delivery delay by more than 20% when compared to state-of-the-art baselines. [4] proposes a T-Coin based IOV system that incentivizes vehicles to follow routes that alleviate traffic congestion. The method is evaluated on a simulated traffic scenario using a real map of Beijing city, and the results reveal a reduction in average travel time compared to the baselines. These research works illustrate the potential of employing VRP (especially SVRP) solutions in modern IOV and Smart City systems, offering promising avenues for addressing transportation challenges and improving overall efficiency.

3.1.3 Background

The Vehicle Routing Problem (VRP) is an NP-hard combinatorial optimization problem. This involves optimizing the routes of vehicles to deliver goods from a depot node to customer nodes with known locations and demands. The objective of the VRP is to minimize the total travel cost of all vehicles used to meet customer demands. Over the years, VRP has been extensively studied and various methods and heuristics have been proposed to solve it [106]. The original VRP formulation was first introduced in [30].

The Stochastic Vehicle Routing Problem (SVRP) is a challenging optimization problem that is of increasing interest in the research community owing to its greater applicability to real-world industrial problems. Unlike its deterministic counterpart, the SVRP parameters are uncertain in advance and can only be realized during the completion of the route. This makes the SVRP a more representative model of real-world routing scenarios and has led to a growing body of research dedicated to solving it [28, 51]. According to [192], there are three main sources of stochasticity in the SVRP:

1. stochastic demands (SVRPCD) - the demand of customers is uncertain,
2. stochastic customers (SVRPC) - the presence or absence of certain customers can be uncertain,
3. stochastic travel times (SVRPTT) - the costs associated with traversing routes are uncertain.

[130] demonstrated that deterministic formulation cannot provide an optimal solution to the SVRP. [107] presented the formulation of the SVRPCD, providing a mathematical representation that considers the inherent uncertainties in the problem.

Classical Problem Formulation

Notations:
N : set of nodes (customers and depot)
C : set of customers
c_{ij} : cost of travel from node i to node j
ξ_i : stochastic demand variable of customer i
μ_{is} : s^{th} realization of the demand of customer i
K : total number of vehicles
Q : maximum capacity of each vehicle
x_{ij} : 1 if arc (i, j) is used in the route, 0 otherwise.

SVRPCD is formulated on a complete, undirected graph $G = (N, X)$. The set of nodes $N = 0, 1, \ldots, n$ includes a depot node (denoted by node 0) and customer nodes. The depot node serves as the base for a fleet of K vehicles with an initial capacity of Q. Each customer node $i \in C = N \backslash 0$ has a stochastic demand variable ξ_i, with its s^{th} realization denoted as μ_{is}. Set of arcs $X = (i, j) : i, j \in N, i < j$ represents the connections between nodes, with the cost of traversing arc $(i, j) \in X$ in a route denoted by c_{ij}. The variable x_{ij} represents the action of an agent, where x_{ij} is 1 if arc (i, j) is used in the route, and 0 otherwise.

The solution to the problem comprises a set of routes assigned to each of the K vehicles, where each route is a sequence of nodes that start and end at the depot node (node 0). Upon visiting a customer node, the corresponding goods are supplied, thereby reducing the customer's demand to zero and decreasing the vehicle's capacity by the magnitude of the customer's demand.

The objective of the problem is to

$$\begin{aligned}
\text{minimize} \quad & \textstyle\sum_{i,j \in C} c_{ij} x_{ij} + \mathscr{R}(x), && (3.1)\\
\text{subject to} \quad & \textstyle\sum_{j=1}^{n} x_{0j} = 2|K|, && (3.2)\\
& \textstyle\sum_{i<k} x_{ik} + \sum_{j>k} x_{kj} = 2 \quad \forall k \in N, && (3.3)\\
& \textstyle\sum_{i,j \in C} x_{ij} \leq |C| - [\sum_{i \in C} \mathbb{E}[\xi_i]/Q], && (3.4)\\
& 0 \leq x_{ij} \leq 1 \quad \forall i, j \in C, && (3.5)\\
& 0 \leq x_{0j} \leq 2 \quad \forall j \in N, && (3.6)\\
& x = (x_{ij}) \text{ integer} \quad \forall i, j \in N. && (3.7)
\end{aligned}$$

The objective is to minimize the total travel cost (3.1), which is represented as the sum of the costs of each arc in the route plus an additional penalty term $\mathscr{R}(x)$. The constraints ensure that each vehicle's path begins and ends at the depot ((3.2), where $x_{0j} = 0$ or 1 does not directly imply vehicles must return from nodes other than the depot—it only governs connections involving the depot), each customer is visited exactly once ((3.3), where the equality ensures that one arc enters

and one arc leaves each customer node, maintaining the continuity of the route and ensuring that no customer is skipped or visited more than once), the vehicle's maximum capacity is sufficient to deliver the expected total demand (3.4), the depot can connect to each node up to twice to account for both departures and returns in multi-vehicle routing (3.6), and the decision variable x_{ij} satisfies the bounds and integrality conditions specified in (3.5), (3.6), and (3.7). These constraints ensure that the solution is feasible and adheres to the requirements of the routing problem.

Objective function (3.1) includes a term referred to as the recourse cost $\mathscr{R}(x)$. This term accounts for the cost associated with failure situations, where a vehicle may arrive at a customer node with insufficient capacity to fulfill demand. In such a scenario, the vehicle must return to the depot to refill its capacity and then return to the failed customer, incurring an additional cost that can be mathematically expressed as:

$$\begin{aligned}\mathscr{R}(x) &= \textstyle\sum_{k=1}^{K} \mathscr{R}^k(x), \\ \mathscr{R}^k(x) &= 2\textstyle\sum_{j=2}^{t}\sum_{l=1}^{j-1} P(\sum_{s=2}^{j-1}\xi_s \leq lQ < \sum_{s=2}^{j}\xi_s) c_{0j}.\end{aligned}$$

The total recourse cost is equal to the sum of the recourse costs for each vehicle, capturing the additional costs incurred due to capacity failures across the entire fleet. The mathematical representation of the recourse cost for vehicle k is given as $\mathscr{R}^k(x)$. This term quantifies the expected additional cost associated with failure scenarios along the route of vehicle k, where a failure occurs if the vehicle's remaining capacity is insufficient to meet the demand of a customer. Specifically, $\mathscr{R}^k(x)$ calculates the likelihood of the l^{th} failure scenario occurring at the j^{th} customer node along the route, weighted by the round-trip travel cost required for the vehicle to return to the depot, refill, and revisit the failed customer. Here, t represents the total number of customer nodes on the route of vehicle k, defining the range of customer indices considered in the recourse calculation. The term lQ denotes the capacity threshold, where l is an integer index (e.g., $l = 1, 2, \ldots$) representing multiples of the vehicle's capacity Q. The inequality $\sum_{s=2}^{j-1}\xi_s \leq lQ < \sum_{s=2}^{j}\xi_s$ identifies the failure scenario where the cumulative demand from customers 2 through $j-1$ is within the capacity lQ, but the total demand up to customer j exceeds it, causing the failure. This probabilistic framework ensures that the recourse cost reflects the expected impact of failures rather than a single deterministic scenario.

Baselines

The **Clarke-Wright** (CW) heuristic [161] is a well-known algorithm in the literature for SVRPCD. This method is based on the concept of *savings*, which is a measure of the reduction in the total travel cost that results from merging two customer nodes into a single route. The saving value between customers i and j is calculated as

$$saving_{ij} = c_{0i} + c_{0j} - c_{ij},$$

where c_{0i} and c_{0j} are the traveling costs between the depot and customers i and j, respectively, and c_{ij} is the traveling cost between customer i and customer j. Here, $c_{0i} + c_{0j}$ represents the cost of serving customers i and j on separate routes (before merging), while c_{ij} represents the cost of serving i and j consecutively on the same route (after merging). The savings values between all

customer pairs are collected in a savings list and sorted in decreasing order. The route merging procedure begins by selecting the largest savings value from the top of the savings list. If the expected demand $\mathbb{E}[\xi_i] + \mathbb{E}[\xi_j]$ of combining customers i and j into the same route does not exceed vehicle capacity Q, then both customers are combined into the same route. The route merging procedure is repeated until no feasible mergings are possible in the savings list.

Tabu Search (TS) is a metaheuristic optimization algorithm that has been widely utilized for solving SVRPCD. The algorithm begins with a randomly generated feasible solution and evaluates the total travel cost. Subsequently, over a defined number of iterations $k_{Tabu,max}$ a set of candidate solutions is generated by applying a set of pre-defined neighborhood heuristic operations to the current solution. Finally, the algorithm returns the best solution among all visited feasible solutions.

[112] shows that a specific set of four heuristics (specific neighborhood, 2-opt, swap-operation, reallocate-operation) demonstrates superior performance on SVRPCD compared to other sets of heuristics that have been previously considered in the literature.

Ant Colony Optimization (ACO) is a well-established metaheuristic. It operates by simulating the behavior of ants in the search for food sources. The algorithm models the ants as solution agents that traverse the solution space by exploring and exploiting feasible routes. When a route is feasible, the corresponding solution agent lays a pheromone on the arcs that form the route. The concentration of pheromones in an arc is proportional to the quality of the arc as a candidate solution, with higher pheromone concentrations leading to an increased likelihood of being followed by subsequent agents. Over time, the pheromone evaporates, simulating the natural behavior of ants. The search for new solutions continues as ants probabilistically select the arcs to follow.

The evaporation of pheromones helps avoid trapping the solution in the local optima. In a study by [52], the superiority of the ACO algorithm for SVRPCD was demonstrated in comparison to other methods. The transition from the current location (i) to the next location (j) for an ant is governed by the following state transition formula,

$$j = \text{argmax}_{j \in \varphi_i} \left\{ \tau_{ij}^a \eta_{ij}^b \right\}.$$

ACO algorithm is sensitive to its parameters: the pheromone trail τ_{ij} on the edge (i, j), the handcrafted heuristic η_{ij} that provides additional evaluation of the edge, the parameters a and b that control the relative importance of pheromone and heuristic, and the number of ants k utilized in the optimization process.

3.1.4 Method

Problem Formulation

Our formulation minimizes total travel cost over the set of demand realizations as follows

$$\text{minimize} \quad \tfrac{1}{|S|} \textstyle\sum_{s \in S} (\sum_{i,j \in C} c_{ij} x_{ij} + 2 \sum_{i \in C} r_{is} c_{0i}),$$

s.t. $\forall i, j \in N$ the r_i, x_{ij} are integers. Here r_{is} represents the recourse action of going to the depot, refilling, and returning to the i^{th} customer, where refill and return actions add the factor of 2 in

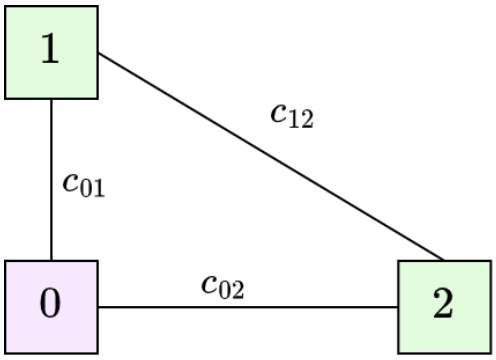


Figure 3.1: Correlated demands example. The example consists of a depot (node 0) and two customers (nodes 1 and 2). Let's assume that the travel cost from the depot to each customer is $c_{01} = c_{02} = 10$, and the travel cost between customers is $c_{12} = 10\sqrt{2}$. Single vehicle with maximum capacity $Q = 50$ is utilized. The demand of each customer can either be 10 or 50. In the case of uncorrelated demands, the optimal strategy is $0 \to 1 \to 0 \to 2 \to 0$, resulting in a travel cost of 40. However, if the demands are positively correlated with a 90% chance of having the same value $(10, 10)$ or $(50, 50)$, the vehicle can directly go to the second customer, if the demand of the first customer is 10, following the strategy $0 \to 1 \to 2 \to 0$. This results in an expected travel cost of 38. If the demands are 100% correlated, the expected travel cost is 37. Utilizing correlations among the demands can lead to improved performance of the agent.

the second term. The constraint implies that the recourse action must be an integer value. This formulation allows for the representation of complex SVRPCD in a deterministic manner as a combination of scenarios, where each scenario s is characterized by the realization of the demand of each customer, μ_{is}. The set of scenarios S with demand realizations μ_{is} can be used for training purposes by any RL algorithm. This approach explicitly optimizes the recourse action, making the formulation suitable for solutions using both the classical and RL models.

Correlated demands. We incorporate new observable stochastic variables into the problem (weather variables), which are assumed to affect the stochastic demand of each customer, creating correlations between them. The agent could observe weather variables. Knowing their values, the agent can adjust its demand estimates and optimize its routing plan to minimize the objective. This formulation closely resembles industry practice, as companies can estimate the unknown demand for a given day by considering various parameters that influence it (e.g., weather). Each weather variable w_j is modeled to have a uniform distribution $w_j \sim U(-1; 1)$ to affect the demand as follows

$$\xi_i = \bar{\xi}_i + \sum_j \sum_k \alpha_{ijk} w_j w_k + \epsilon_i.$$

We use w_j to model the stochastic demand ξ_i. The predicted value of the stochastic variable will be equal to a constant value $\bar{\xi}_i$, augmented by the weighted cross effect of weather variables w_j, w_k, and perturbed by random noise ϵ_i correlated among customers. Weather variables are shared across customers, leading to *correlated demand*. Figure 3.1 illustrates a simple example of the impact of using correlated demands in the policy on agent performance.

Learning the Routing Policy

State. We define the RL setting as a SVRPCD optimization problem with one vehicle and a given set of inputs at time t

$$I^t \doteq \{(w, p_i, d_i^t), i \in N\},$$

where each state is represented by a set of tuples. The tuple (w, p_i, d_i^t) encodes information about the weather, location of customer i, and demand of customer i at time t. One can view w as a vector of features that can affect the demand of customers, p_i as a coordinate representation or one-hot encoding of the position of customer i, and d_i^t as the current demand estimate of customer i.

Within the context of Reinforcement Learning (RL), the action corresponds to the movement of a vehicle towards a specific node. Given the current state at time t, the agent selects the next node from a set N, and the vehicle proceeds to that node for the subsequent iteration at time $t+1$. Here, the term **action** is represented by a pointer that chooses node $i \in N$ (which can be a customer or a depot) at each time step t using a policy π (i.e., $a^t \sim \pi(\cdot|I^t, h^t)$). The policy π takes into consideration the current state I^t and the memory state h^t, which encodes the information of all previous actions $a^1, ..., a^{t-1}$. By employing this policy π, a probability distribution over all nodes N is generated, enabling the agent to determine the node towards which the vehicle should move. When a specific action is executed, the vehicle incurs a corresponding cost denoted as C_t (in real-world scenarios, this cost can represent factors such as time, distance, or monetary expenses associated with this movement). In our RL framework, this cost is considered as the negative reward, implying that a lower cost corresponds to a higher reward for the agent.

Transition. We update the representation of the state after each action a^t is taken $I^{t+1} \sim f(\cdot|I^t, a^t)$, where the weather variables and customer locations are assumed to remain unchanged. Consequently, the formula for updating the customer demand d_i^t and vehicle capacity q^t provides a definition of the state transition function:

$$\begin{aligned} d_i^{t+1} &= \max\left\{0, d_i^t - q^t\right\}, \\ d_k^{t+1} &= d_k^t, \text{ for } k \neq i, \text{and } q^{t+1} = \max\left\{0, q^t - d_i^t\right\}. \end{aligned} \tag{3.8}$$

Objective. We quantify the **cost** at a given time step t after taking action a^t, which entails moving from node i to node j, as $C_t(I^t, a^t) = c_{ij}$. In the event of capacity exhaustion, the **naive recourse strategy** is employed, in which the vehicle returns directly to the depot to refill its capacity, incurring an additional cost $C_t(I^t, a^t) = c_{ij} + 2c_{0j}$. The policy is learned by utilizing the Reinforce algorithm [215] with the aim of minimizing the cumulative expected cost

$$\mathcal{J}^\pi(\theta) = \mathbb{E}[\textstyle\sum_{t=1}^T C_t^{\pi_\theta}(I^t, a^t)],$$

where θ represents the parameters of policy π and the cost of the policy is accumulated over the course of a finite horizon T, where T is defined as the point in time when all customer nodes have been visited. According to the Policy Gradient Theorem [183], the gradient of the objective function can be approximated as

$$\begin{aligned} \nabla_\theta \hat{\mathcal{J}}^\pi(\theta) \approx \tfrac{1}{B} \textstyle\sum_{b=1}^B \sum_{t=1}^T ((TC(I^t, h^t) - b_\phi(I^t, h^t)) \\ \cdot \nabla_\theta \log \pi_\theta(a^t|I^t, h^t)), \end{aligned}$$

where b is an instance from a batch of size B; $b_\phi(s_t, x)$ is the baseline, the estimate of the value function, and TC, is the actual cumulative cost $TC(I^t, h^t) = \sum_{t=1}^{T} C_t(I^t, a^t)$. The baseline is computed using a critic network to speed up convergence. The critic is trained by updating the parameters ϕ to minimize

$$L(\phi) = \frac{1}{B}\sum_{b=1}^{B}\sum_{t=1}^{T}||b_\phi(I^t, h^t) - TC(I^t, h^t)||^2.$$

Sensitive Factors

The proposed method comprises several critical components that require careful consideration when assessing its robustness. First, the method's performance may be contingent on the choice of inference strategies used to determine the action a^t at each state. Second, the stochastic demand in the proposed method comprises three terms: a constant term, weather-related term, and noise. The absolute magnitude of each term can affect the performance of the method. Third, the maximum capacity of the vehicle may have a significant impact on the occurrence of failed situations. Fourthly, the stochastic demand can be estimated using various methods. Finally, in industry applications of the SVRPCD, two additional factors can also be of critical importance: the agent may have access to the customer demand once he reaches its node, and the positions of the customers can be fixed.

Inference Strategies. Utilization of the *greedy* inference approach, where actions are selected based on the maximum probability $a^t = \arg\max[\pi(\cdot|I^t, h^t)]$, has been shown to produce suboptimal results [7, 102]. Therefore, this study evaluates two alternative inference strategies to address this issue.

The *Sampling* approach entails repeating the inference process several times by sampling from the policy distribution, with the aim of generating multiple routing solutions. The number of samples drawn is represented by the variable n_s. The ***Beam Search*** strategy [87, 203] involves exploring a set of n_b candidate actions at each time step of the inference process. Subsequently, each candidate solution is expanded with a new set of actions, from which the beam search selects only the n_b candidates with the highest likelihood.

Demand Terms Effect. To conduct a comprehensive analysis of the impact of each component of the stochastic demand formulation, we defined three term-effect variables:

$$A_i = \frac{\bar{\xi_i^2}}{T_i} \qquad B_i = \frac{\mathbb{E}\big[(\sum_j \sum_k \alpha_{ijk} w_j w_k)^2\big]}{T_i}, \qquad \Gamma_i = \frac{\mathbb{E}\big[\epsilon_i^2\big]}{T_i},$$

$$T_i = \bar{\xi_i^2} + \mathbb{E}[(\textstyle\sum_j \sum_k \alpha_{ijk} w_j w_k)^2] + \mathbb{E}\left[\epsilon_i^2\right],$$

where A_i, B_i, Γ_i represent the effects of the constant, weather-related, and noise terms on the observed demand realization. These values are expressed on a normalized scale $[0, 1]$ and are required to sum up to 1. Utilizing these variables can facilitate a more in-depth examination of the effect of the stochastic components of demand on the performance of the model.

Filling Rate. We define the filling rate $\Phi = \frac{Q}{\sum_i \mathbb{E}[\xi_i]}$, as the ratio of the maximum vehicle capacity to the sum of the expected demands of all customers. A higher value of Φ indicates a lower likelihood of failure demands, and thus reduces the need for recourse actions.

A Priori vs Reoptimization Routing. The term *a priori* route refers to a route solution that is generated prior to revealing the actual customer demands. This solution is based on assumed demand estimates $\hat{d}_i$, which model uses as an input demand, $d_i = \hat{d}_i, \forall i \in N$. The current vehicle capacity q^t is calculated with the given demand estimates. After a priori route is fully constructed the travel cost is calculated using the actual demand realizations.

The a priori approach is differentiated from the *reoptimization* approach, which involves updating the solution after the current demand realization of each customer has been revealed, $\hat{d}_i^t = \mu_{is}$. In the reoptimization approach, an initial demand estimate $\hat{d}_i$, is utilized for each customer i prior to the knowledge of its actual demand realization. Once the agent reaches customer i for the first time, the initial demand estimate is updated with the actual demand realization and utilized in subsequent optimization steps.

Demand Estimates. We introduce two methods for estimating the customer demand. The first method assumes that the unknown demand realization is equal to its constant component, $\hat{d}_i = \bar{\xi}_i, \forall i \in N$. The second approach for estimating demand involves utilizing information related to weather variables. This necessitates the existence of a *weather history*, which serves as a repository for tuples of the form (w^k, p_i, μ_{ik}), representing the k-th realization of weather and demand variables for customer i with position p_i. Given the actual weather value w, the Euclidean distance is calculated to identify a set of K nearest neighbors, $N(w)$, in the weather history. The demand estimate is then derived from the normalized demands of the K nearest neighbors as follows:

$$\hat{d}_i = (\sum_{w_k \in N(w)}^{K} \frac{1}{||w_k - w||} \mu_{ik}) / (\sum_{w_k \in N(w)}^{K} \frac{1}{||w_k - w||}). \tag{3.9}$$

Fixed vs Flexible Customer Position. We consider two approaches to customer positioning. The first approach is a *flexible position*, in which customers may have different positions on the map for each new problem instance. In this case, the (z, l) coordinates on the map of each customer are used as static inputs to the model, $s_i = (z_i, l_i)$. Another approach is to maintain *fixed customer positions*. This scenario is common in real-world applications, where the locations of customers often remain constant. In this case, because the (z_i, l_i) coordinates are the same during training and inference, the customer location can be encoded using a one-hot vector $s_i = (0, 0, ..., 1, ..., 0)$.

Supply Type. Another factor that may affect the performance of the model is supply type. One possible approach is *partial supply*, where the demand of customer i can be met with any quantity of the current vehicle capacity q^t. In this scenario, the demand-state transition function remains

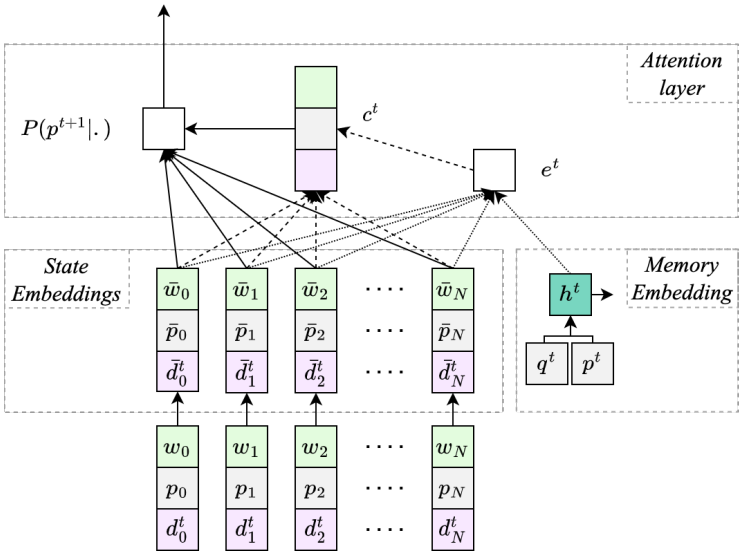


Figure 3.2: **Network architecture.** The bottom part depicts the input of the model that consists of three components: weather information w, the location p_i, and the dynamic demand d_i^t of each customer i. The model generates State Embeddings for each customer and encodes the vehicle's current position (p^t) and capacity (q^t) as the Memory Embedding (h^t). The embeddings are then combined through an Attention Layer to obtain probabilities over the nodes, representing the likelihood of each node being the next position of the vehicle.

the same (3.8). The second approach is *full supply only*, where the demand of customer i can be fulfilled only completely if there is a sufficient quantity of current vehicle capacity q^t. In this case, the state transition function of the demand of customer i upon the vehicle's visit to the customer's node will be defined as $d_i^{t+1} = 0$, if $q^t \geq d_i^t$ and $d_i^{t+1} = d_i^t$ otherwise.

3.1.5 Architecture Details

Embeddings. The architecture of the model is illustrated in Figure 3.2. It consists of three main components: finding *state embeddings* given the input data, encoding the sequence of visited nodes into a *memory embedding*, and decoding concatenated embeddings into action probabilities using *the attention layer*. The input to the model is a set of tuples (w, p_i, d_i^t) for each node i. The state embeddings are obtained by applying a convolution layer, resulting in a D-dimensional representation $\rho_i^t = (\bar{w}, \bar{p}_i, \bar{d}_i^t)$.

Encoding. The encoding of the sequence of visited nodes is accomplished using LSTM cell. This cell produces a memory embedding $h^t \in \mathbb{R}^D$, given the vehicle's current position p^t and capacity q^t. To combine the information from the different embeddings, we utilize an attention mechanism that computes an attention score $e^t = e^t(\rho_i^t, h^t)$ and a context vector $c^t = \sum_{i=1}^{M} e_i^t \rho_i^t$.

Decoding. The conditional probabilities are computed by combining context vector c^t and state embeddings as

$$u_i^t = W_a^T \tanh\left(W_b[\rho_i^t; c^t]\right), P(p^{t+1}|\cdot) = \text{softmax}(u_i^t),$$

where W_a and W_b are the trainable parameters. The softmax function is applied to the values u_i^t to obtain a vector of probabilities $P(p^{t+1}|\cdot)$ representing the likelihood of each node being the next in the route. These probabilities can then be utilized in the inference strategy to produce the routing solution.

3.1.6 Experiments

Data Generation

Positions. In this study, we evaluated the performance of model for five different problem sizes, with 10, 20, 50, and 100 customers. The problem instances were generated by randomly selecting the coordinates (z_i, l_i) of each customer within the unit square $[0, 1] \times [0, 1]$. The depot was located at the center of this square with coordinates $(z_0, l_0) = (0.5, 0.5)$.

Demand. Three weather variables are considered in this study: temperature, humidity, and wind speed. These variables were generated using a uniform distribution in the interval $[-1, 1]$. The constant part of the demand $\bar{\xi}_i$ is generated using a uniform distribution over the interval $[1, 10]$, while the noise part ϵ follows a multivariate normal distribution $\mathcal{N}(0, \Sigma)$, where variance $\sigma^2_{\epsilon_i}$ is chosen based on a given noise term effect value Γ_i and a randomly created non-zero covariance matrix Σ. By default, the values of A_i, B_i, and Γ_i are set to 0.6, 0.2, and 0.2 respectively. Then, the values α_{ijk} are randomly generated to satisfy the given $B_i, \bar{\xi}_i$, and w_j, w_k values. Finally, the capacity Q was determined based on the filling rate Φ.

Datasets. To obtain the *weather dataset*, a set of 10,000 realizations of (w, p_i, μ_{is}) was generated using the above (positions and demand) generation process. During the model training phase, random values (w, p_i, μ_{is}) is created for every training iteration. In addition, a *validation dataset* consisting of 1,000 instances was created for each problem size and saved for ACO parameter search. A similar process was used to create a *test dataset* for model comparison.

Masking

A masking procedure was implemented to generate feasible solutions. This procedure involves setting the log-probabilities of infeasible solutions to $-\infty$. The masking we used included the following:

- preventing already visited nodes to be visited again
- preventing all customer nodes to be visited if the vehicle's estimated capacity is 0
- preventing customers whose estimated demand exceed the estimated vehicle capacity

This masking procedure ensures that when the vehicle visits a customer, it must satisfy all customer demands.

Table 3.1: Results of ACO algorithm with varying parameter values on SVRPCD validation data. The rows represent values for the parameters controlling the pheromone importance (a), heuristic importance (b), and number of ants (k). The columns indicate the number of customers. Results are reported as the average travel cost over the validation data, with lower values indicating better performance.

ACO PARAMETERS	10	20	50	100
3 / 4 / 20	3.24	7.21	14.11	31.26
6 / 8 / 24	3.21	7.05	14.02	31.13
8 / 2 / 5	3.23	7.11	14.19	31.25
4 / 9 / 15	3.18	6.79	14.03	31.14
3 / 10/ 12	**3.17**	**6.54**	**13.92**	**30.89**

Implementation Details

Parameters. To obtain the state embedding, we employed a 1-dimensional convolutional layer with D filters and $|\rho_i^t|$ channels. To obtain the memory embedding, we utilized single-layer LSTM with a state size D. The outputs of the layer computing $P(p^{t+1}|\cdot)$ are followed by a critic network, consisting of two fully-connected layers with D and one neurons respectively. The activation function utilized in all the layers is the Rectified Linear Unit (ReLU). The default dimensionality of the embeddings was set to $D = 128$.

Training. The variables in the neural network architecture were initialized using the Xavier initialization method. During training, the Adam optimizer was used with a learning rate of 10^{-4} and a batch size of $B = 128$. To reduce overfitting, a dropout with a probability of 0.1 was employed during training. The model was trained on a GPU system consisting of an NVIDIA A100 SXM 40GB GPU and 2x AMD EPYC 7742 CPUs (8 cores) with 256GB RAM for 10,000 iterations on each problem size.

Results

Baselines. For the CW heuristic, implementation proposed in [161] was employed. For the Tabu search, we used [112] algorithm, with a set of four heuristics mentioned in the paper and a maximum of 2000 iterations. The ACO algorithm implementation [52] is sensitive to its parameter values, and thus, a random grid search was performed over discrete parameter space, with the pheromone importance (a) in the range [1, 10], heuristic importance (b) in the range [1, 10], and the number of ants (c) in the range [5, 30]. The performances of 15 randomly generated parameter tuples were evaluated using **a validation dataset**. Table 3.1 presents the results for five parameter tuples, with the best result achieved by a,b, and c equal to 3,10, and 12 respectively. For the heuristic evaluation, the reverse distance measure, $\eta_{ij} = \frac{1}{d_{ij}}$, was employed.

Table 3.2 presents a comparison of the average travel cost results on the test data for the baselines and RL models. The baselines consist of Tabu search, CW, and the best-performing model of ACO, as listed in Table 3.1. According to classical research findings ([52]), the ACO model consistently exhibits the lowest travel cost across all problem sizes among the baselines.

Table 3.2: Results of Baseline and the RL models on SVRPCD test data. The rows depict three SVRPCD heuristic/metaheuristics and RL model that employs a beam search inference strategy.

BASELINES/INSTANCES	10	20	50	100
CLARKE-WRIGHT	3.31	7.31	14.54	32.59
TABU SEARCH	3.25	7.25	14.23	31.64
ANT-COLONY OPIMIZATION	3.17	6.54	13.92	30.89
RL-BEAM SEARCH	**3.13**	**6.39**	**13.33**	**29.27**

Table 3.3: Results of different Inference strategies on SVRPCD test data. The rows depict three different inference strategies. The columns indicate number of customers. Results are reported for a priori (Apr.) and reoptimization (Reopt.) settings.

	10		20		50		100	
INFERENCE STRATEGIES	APR.	REOPT.	APR.	REOPT.	APR.	REOPT.	APR.	REOPT.
RL-GREEDY	3.60	3.49	7.35	7.13	15.33	14.87	33.66	32.65
RL-SAMPLING	3.32	3.22	6.78	6.57	14.13	13.71	31.03	30.10
RL-BEAM-SEARCH	**3.13**	**3.04**	**6.39**	**6.20**	**13.33**	**12.93**	**29.27**	**28.39**

For the RL model, an a priori routing approach was employed with the demand values estimated using the k-NN algorithm ($k = 5$). This configuration provides a fair comparison with the baselines, as their routing solutions are determined in an a priori manner and they use the same demand estimates. During inference, a beam search strategy with a beam width ($n_b = 3$) was utilized. The RL model exhibits improved performance compared to ACO, resulting in a decrease in the travel cost by 1.26%, 2.29%, 4.24%, and 5.24% for 10, 20, 50, and 100 customers respectively. On an average, the observed improvement was 3.26%. It is worth mentioning that a larger size of the problem constitutes a larger $\%$ gap between the RL and ACO models.

Sensitive Factors. The following experiments were conducted to assess the sensitivity of the proposed model to various factors. The default model utilized RL with a beam-search inference strategy ($n_b = 3$), k-NN demand estimates, fixed customer positions, partial supply, demand term proportions set to $A, B, \Gamma = 0.6, 0.2, 0.2$, and filling rate $\Phi = 0.5$. We started experiments with a single-vehicle scenario to evaluate the model's performance and its dependence on sensitive factors. In each experiment, a single factor was changed and evaluated across the four problem sizes, and the results for both the a priori and reoptimization approaches are reported. Although the reoptimization approach consistently produced superior results, the a priori approach is still relevant in industrial applications and may be effective in certain real-world scenarios; thus, both approaches are important to consider.

The performance of greedy, sampling, and beam-search **inference strategies** were evaluated.

Table 3.4: Results of two types of demand estimates on SVRPCD test data. The rows depict demand estimate types.

	10		20		50		100	
Demand Estimate	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.
Constant demand estimate	3.38	3.28	6.90	6.70	14.40	13.96	31.61	30.67
kNN demand estimate	**3.13**	**3.04**	**6.39**	**6.20**	**13.33**	**12.93**	**29.27**	**28.39**

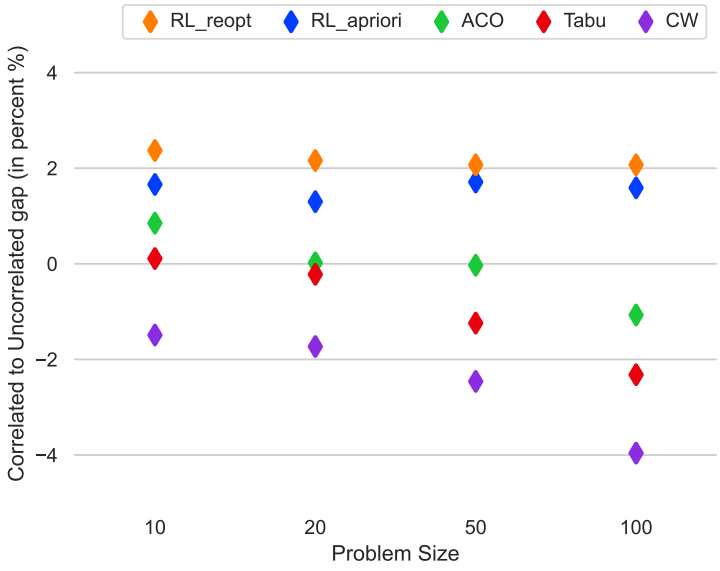


Figure 3.3: Avg. travel cost gap (in percent) of setting with correlated demands ($\mathrm{A}/\mathrm{B}/\Gamma = 0.8/0.2/0.0$) to setting with uncorrelated demands ($\mathrm{A}/\mathrm{B}/\Gamma = 0.8/0.0/0.2$) for Baseline and RL models with higher values indicating better performance.

The results (Table 3.3) indicate that beam-search exhibits superior performance across all four problem sizes using both a priori and reoptimization approaches. Sampling strategy with $n_s = 16$ was found to be superior to greedy search.

Experiments evaluating the **demand estimates** were performed, and the results indicate superior performance of the k-NN estimate, as shown in Table 3.4. The results of the experiments investigating the impact of **customer positions** on performance are presented in Table 3.5. The results indicate the superior performance of fixed customer positions in the test data.

Four sets of values were utilized in the experiments on different **demand term effects** (Table 3.6). The results indicate that constant and weather terms have a positive impact on performance, whereas noise has a negative effect.

The Figure 3.3 illustrates the percentage gap in travel times between the correlated and uncorrelated demands. For the purpose of these experiments, correlated demands were generated using the values $A, B, \Gamma = 0.8, 0.2, 0.0$, while uncorrelated demands were generated using $A, B, \Gamma = 0.8, 0.0, 0.2$. The results reveal that the RL model with the reoptimization approach yields a 2.2% improvement in performance when dealing with correlated demands, and this difference is consistent across all problem sizes. The RL with an a priori approach provides a stable, albeit lower, improvement on average. On the other hand, the baseline models display a negative gap, indicating that they do not capture the correlations between demands, and this negative gap becomes more pronounced with larger problem sizes.

The experiments on **signal ratio** employed the values of Φ equal to $0.1, 0.5,$ and 0.9. The results presented in Table 3.7, indicate that Φ has a positive impact on performance of the model, where higher Φ values result in increased vehicle capacity and fewer recourse actions. Importantly, even with a relatively small value of $\Phi = 0.1$, the model is still able to produce comparably good

Table 3.5: Results of two types of Customer positions on SVRPCD test data. The rows depict customer position types.

	10		20		50		100	
Customers Position	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.
Flexible customer positions	3.51	3.40	7.16	6.95	14.93	14.48	32.78	31.80
Fixed customer positions	**3.13**	**3.04**	**6.39**	**6.20**	**13.33**	**12.93**	**29.27**	**28.39**

Table 3.6: Results of different proportions of demand terms on SVRPCD test data. The rows depict proportional effect of demand terms.

	10		20		50		100	
$A/B/\Gamma$	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.
0.8 / 0.0 / 0.2	3.02	2.95	6.15	6.02	12.88	12.54	28.26	27.54
0.8 / 0.2 / 0.0	**2.97**	**2.88**	**6.07**	**5.89**	**12.66**	**12.28**	**27.81**	**26.97**
0.6 / 0.2 / 0.2	3.13	3.04	6.39	6.20	13.33	12.93	29.27	28.39
0.4 / 0.3 / 0.3	3.38	3.28	6.90	6.70	14.40	13.96	31.61	30.67

Table 3.7: Results of different Filling rates on SVRPCD test data. The rows depict values of the filling rate.

	10		20		50		100	
Filling Rate	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.
0.1	3.51	3.40	7.16	6.95	14.93	14.48	32.78	31.80
0.5	3.13	3.04	6.39	6.20	13.33	12.93	29.27	28.39
0.9	**2.91**	**2.82**	**5.95**	**5.77**	**12.40**	**12.02**	**27.22**	**26.41**

Table 3.8: Results of two Supply types on SVRPCD test data. The rows depict supply types.

	10		20		50		100	
Supply Type	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.
Full supply only	3.47	3.37	7.10	6.88	14.80	14.35	32.49	31.52
Partial supply	**3.13**	**3.04**	**6.39**	**6.20**	**13.33**	**12.93**	**29.27**	**28.39**

Table 3.9: Results of a varying number of vehicles on SVRPCD test data. The rows depict number of vehicles.

Number of vehicles	10	20	50	100
1	3.04	6.20	12.93	28.39
2	2.98	6.13	12.74	28.01
3	2.95	6.06	12.59	27.73
5	**2.95**	**6.01**	**12.42**	**27.64**

Table 3.10: Average inference time (in seconds) using different baselines over a test set of size 1000.

Baseline	10	20	50	100
Clarke-Wright	0.003	0.013	0.054	0.171
Tabu Search	0.831	2.723	8.123	56.35
Ant-Colony Optimization	1.028	4.095	27.39	120.3
RL-greedy	0.052	0.108	0.159	0.328
RL-sampling	0.059	0.129	0.204	0.385
RL-beam search	0.064	0.141	0.217	0.406

performance, despite the need for an average of ten recourse actions.

Experiments evaluating the influence of **supply type** on performance were conducted, and the results are presented in Table 3.8. The data indicate that partial supply conditions result in the superior performance on the test data.

Multi-Vehicle Scenario

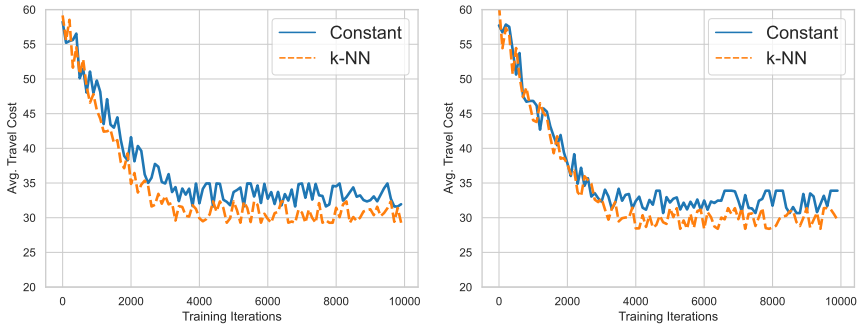
To scale our approach to a multi-vehicle scenario, we augmented the input layer of the actor network with q_k^t and p_k^t for all vehicles $k \in K$ and predicted p_k^{t+1} for each vehicle. This enabled us to find the next position of each vehicle based on its current position and capacity with minimal modifications to the architecture.

We tested the model on varying numbers of vehicles using the RL model with a reoptimization setting. The results (Table 3.9) suggest that increasing the number of vehicles can lead to a decrease in total travel cost, but its marginal effect diminishes as the number of vehicles increases. This demonstrates the ability of the model to obtain better routing strategies when having more vehicles to operate with.

Time Complexity

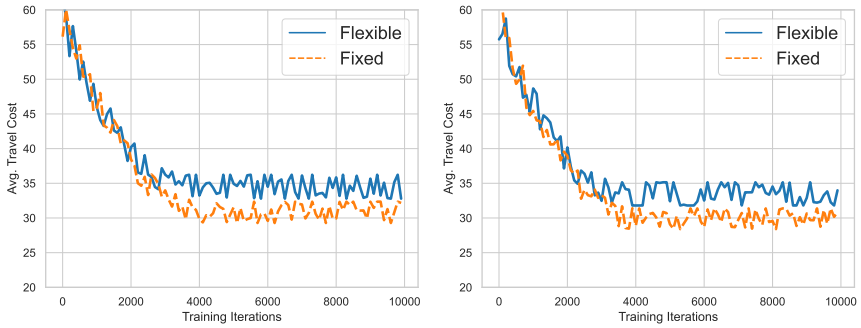
Table 3.10 displays a comparison of the average run-times of the algorithms considered. The RL models adopted an a priori routing approach, with the k-NN algorithm ($k = 5$) used for demand estimation. This setup ensures a fair comparison with the baselines (CW, Tabu, and ACO), as their routing solutions are determined in an a priori manner, and they use the same demand estimates.

Among the considered algorithms, the CW heuristic exhibited the shortest average time required to obtain a solution for a given instance. Although RL requires more time, all the solutions were found in less than one second. It is noteworthy that both RL and CW exhibit linear time complexity, whereas Tabu and ACO display quadratic complexity.



(a) A priori (b) Reoptimization

Figure 3.4: Average travel cost observed during training using (a) a priori and (b) reoptimization approaches of constant and k-NN demand estimates. The k-NN estimate outperforms the constant in both settings.



(a) A priori (b) Reoptimization

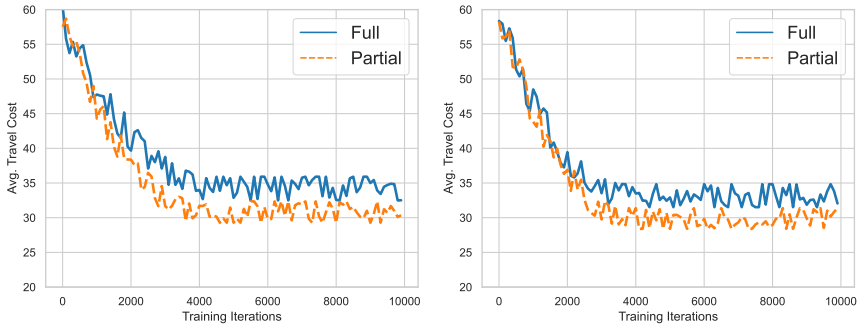
Figure 3.5: Average travel cost observed during training using (a) a priori and (b) reoptimization approaches for Flexible and Fixed customer positions. Fixed positions outperform Flexible positions in both settings.

Training Curves

The training curves of different demand estimates can be observed in Figure 3.4, where both approaches initially exhibit similar performance, but after 2000 iterations, the k-NN estimate consistently demonstrates improved results. Figure 3.5 shows the training curves of the different customer positioning approaches, which depict the consistently improved performance of fixed customer positions during the training process. Figure 3.6 shows the training curves of different supply types.

3.1.7 Discussion and Conclusion

In this study, we propose a **novel formulation** for the Stochastic Vehicle Routing Problem with Correlated Demands (SVRPCD) that incorporates the impact of weather variables on customer demands. This formulation is relevant for practical applications because it enables companies to consider observed variables, such as weather, in the planning of vehicle routes. Our proposed approach utilizes Reinforcement Learning (RL) to learn the correlation between demands, making it highly applicable to various SVRPCD scenarios.



(a) A priori (b) Reoptimization

Figure 3.6: Average travel cost observed during training using (a) a priori and (b) reoptimization approaches of Full and Partial supply types. Partial supply outperforms Full supply in both settings.

We conducted experiments to evaluate the performance of the RL model compared with the state-of-the-art model in [52]. Our results demonstrate that the RL model **outperforms the prior approach** and confirm the **superiority of non-i.i.d. environments** for SVRPCD problems. Furthermore, the model can learn the correlation between demands **without human intervention**, which enhances its practicality and versatility. Through sensitivity experiments, we demonstrate the **robustness of the model** in response to environmental changes.

The findings of this study suggest potential for significant **reductions in transportation costs** and improvements in **environmental sustainability**. Future research could extend the scope of this study to include additional sources of stochasticity, such as stochastic travel time.

3.2 Learning Algorithm for Vehicle Routing Problem with Stochastic Travel Times

3.2.1 Introduction

In this research endeavor, we aim to enhance the applicability of our framework to real-world supply-chain scenarios by incorporating two important sources of stochasticity, namely stochastic demand and stochastic travel times. Travel cost is often modeled as a function of travel time, with longer travel times typically incurring higher costs due to fuel consumption, labor expenses, or vehicle maintenance. We conduct experiments to assess the dependence of our proposed model on different settings of the SVRPTT environment, which are frequently explored in classical research. It is important to note that the proposed architecture is designed to be flexible and capable of addressing SVRPTT instances with an arbitrary number of vehicles. Our specific **contributions** to this field are summarized as follows:

- In this study, we address the Stochastic Vehicle Routing Problem and propose a **novel RL-based approach** to solve it. Our model incorporates stochastic customer demands and travel times, which are crucial factors in real-world logistics scenarios. We compared

our RL model to state-of-the-art approaches and demonstrated its superiority, achieving a **3.43% reduction in travel cost** compared to the best-performing classical model [52].

- We conducted comprehensive experiments to evaluate the performance of our model across various VRP settings. We examined different inference strategies, including greedy search, sampling, and beam search. The results consistently demonstrated the superiority of beam search in terms of travel cost reduction. We investigated the impact of different factors on model performance. Our findings revealed that fixed customer positions, higher fill rates, and partial delivery settings led to improved routing strategies, at the same time being **robust in other environment setting**. We also observed that incorporating weather information, which introduced correlations among customers, enabled our RL agent to achieve up to a **2% improvement in routing strategies**.
- Overall, our research showcases the effectiveness of RL in addressing the SVRPTT and its potential to outperform traditional approaches used in industry. The results highlight the **significant cost reductions and potential environmental benefits** that can be achieved by adopting RL-based routing solutions.

Formulation

The objective of the problem is to

$$\text{minimize} \qquad \textstyle\sum_{i,j \in C} c_{ij} x_{ij} + \mathscr{R}(x), \tag{3.10}$$

where N - set of customers and depot, C - customers set, c_{ij} - realization of $\{ij\}$ travel cost, ξ_i - stochastic demand of customer i, μ_i - realization of the demand of customer i, K - set of vehicles, Q - maximum capacity of vehicle, x_{ij} - binary variable that shows whether (i, j) is used in the route.

In this formulation, the SVRPTT is modeled on a complete, undirected graph $G = (N, X)$. A depot node (i.e. node 0) and customer nodes are represented as the set of nodes $N = \{0, 1, \ldots, n\}$. The depot functions as the base for the set of K vehicles with an initial loading value of Q. A stochastic demand variable ξ_i, including its s^{th} realization characterized as μ_{is}, exists per customer node $i \in C = N \backslash 0$. Arcs, given as $X = (i, j) : i, j \in N, i < j$, denote the associations between nodes; the travel cost of an arc $(i, j) \in X$ in a route is c_{ij}. The representation x_{ij} is an action of an agent with x_{ij}=1 when an arc (i, j) is traveled in the route and 0 otherwise. In case of situations when a vehicle fails to fulfill the demand of a customer due to a shortage of load, the recourse cost $\mathscr{R}(x)$ accounts for it. To be precise, the term is responsible for the cost incurred for traversing to the depot and back to refill in order to meet the customer demand.

The final solution using this formulation is a set of routes for each vehicle, with each route starting and ending at the zeroth node. With each customer visit, the demanded goods are supplied, which in turn means the customer demands are met (set to 0) and the load of the vehicle is reduced by the amount of the customer demand.

3.2.2 Method

Objective. We propose a problem formulation where the objective is to

$$\text{minimize} \quad \frac{1}{|S|}\sum_{s\in S}(\sum_{i,j\in C} c_{ijs}x_{ij} + 2\sum_{i\in C} r_{is}c_{0is}),$$

where the agent minimizes the travel cost in the context of a set of scenarios, denoted as S. The agent aims to solve this problem by making decisions on action variables, specifically x_{ij} and r_is. Here, r_is represents a binary recourse action that determines whether the vehicle needs to return to the depot after visiting customer i in order to replenish its capacity and prevent potential issues with future customers along the route. Compared to the VRP with stochastic demands, this formulation extends the model by incorporating the realizations of stochastic travel times c_{ijs}

Stochastic Variables. The agent lacks prior knowledge of the distribution of stochastic variables and does not have access to the specific realization s of the demand μ_{is} or the travel cost c_{ijs}. The agent can only observe the value of μ_{is} after visiting customer i and the value of c_{ijs} after traversing the arc $\{ij\}$. However, the agent has the ability to estimate the values of stochastic variables by leveraging the knowledge of the observable variables. We define a set of random variables W that influence the realization of demand and travel times in the following manner:

$$\theta_i = \bar{\theta}_i + \sum_m \sum_n \alpha_{imn} w_m w_n + \epsilon_i, \ \theta_i \in \{\xi_i, c_{i0}, ..., c_{iN}\},$$

where we define θ_i as a stochastic variable representing the demand of customer i or the travel cost from customer i to other nodes. The value of θ_i is determined by a fixed term $\bar{\theta}_i$, which is augmented by the weighted interaction between variables $w_m \in W$ and $w_n \in W$, as well as random noise ϵ_i. It is important to note that the agent does not possess knowledge of the distribution of the variables in W or their specific influence on θ_i. However, the agent does have access to the realizations of W variables, allowing him to implicitly learn the mapping from W variables to stochastic variables to enhance the routing strategy.

Weather Variables. This study focuses on a set of three W variables referred to as weather variables, namely temperature, pressure, and humidity. This selection is motivated by real-world scenarios where the agent lacks prior knowledge of demand and travel times but can estimate them by observing the prevailing weather conditions, which exert a direct influence on these factors. For instance, the demand for ice-cream and the travel time between ice-cream shops may vary based on the temperature. By incorporating these weather variables, we aim to create a realistic environment where the agent utilizes observable weather information to estimate the associated stochastic variables effectively. It is important to acknowledge that the presented formulation provides an abstract representation of the problem, and in practice, there may exist additional stochastic variables beyond demand and travel time that impact the agent's costs. Moreover, there is the possibility of incorporating additional W variables that the agent assumes to have an influence on the environment.

Routing Policy

The architecture used for this model is similar to the one previously employed for the VRP with stochastic demands; however, the inputs in this case represent estimated travel times rather than the actual travel times.

State. The proposed SVRPTT formulation defines the environment as a state (I_s^t, h_k^t) associated with a particular realization s and vehicle k,

$$I_s^t \doteq \{(W, d_i^t, c_{i0s}, ..., c_{iNs}), i \in N\},\ h_k^t \doteq (q_k^t, p_k^t), k \in K.$$

The state is dynamic and evolves over time $t \in \{1, ..., T\}$, where T represents the time step when all vehicles have returned to the depot and all customer demands have been fulfilled. I_s^t is characterized by a set of weather variables W, the current demand of customer i at time t, denoted as d_i^t, and the travel cost realizations c_{ijs} between all nodes $\{ij\}$. The weather variables and travel times remain constant throughout the time horizon, while the demand d_i^t takes on the value of the demand realization μ_{is} at $t = 1$, and 0 at $t = T$ for all customers $i \in N$. h_k^t is characterized by the current load q_k^t and the position p_k^t of each vehicle $k \in K$ at time t.

Action. At each time step t, the agent selects an action a_k^t for each vehicle, where the action represents the choice of the next node $i \in N$ to visit. The decision-making process is dynamic and governed by a policy π, such that $a_k^t \sim \pi(\cdot|I_s^t, h_k^t)$. In other words, the policy determines the probability distribution of selecting the next node for each vehicle based on the current state (I_s^t, h_k^t).

Transitions. Upon executing the action a_k^t, the positions of the vehicles are updated to p_k^{t+1} based on the nodes they move to. The representation of vehicle positions can be encoded using various techniques, such as coordinates on a 2D map or one-hot vectors. The demands and loads for vehicle k that visits the customer i are updated according to the following formulas:

$$d_i^{t+1} = \max\left\{0, d_i^t - q_k^t\right\},\ d_j^{t+1} = d_j^t, \text{ for } j \neq i, \text{ and } q_k^{t+1} = \max\left\{0, q_k^t - d_i^t\right\}. \tag{3.11}$$

Objective. The objective remains identical to that of the SVRPCD scenario. The agent aims to derive an optimal policy π with parameters Θ by employing the Reinforce algorithm, which seeks to minimize the cumulative expected cost

$$\mathcal{J}^\pi(\Theta) = \mathbb{E}[\textstyle\sum_t^T C_t(I_s^t, a_k^t)].$$

The cost function $C_t(I_s^t, a_k^t)$ is defined as the sum of traversal costs for each vehicle k moving from node i_k to node j_k after executing action a_k^t. It is calculated as $\sum_{k=1}^K c_{i_k j_k s}$. In cases where vehicle k requires recourse action due to a failure, an additional recourse cost of $2c_{0j_k s}$ is incurred. The objective function is accumulated over a set of scenarios S, each consisting of T_s time steps, and the gradient of the objective function is computed as follows:

$$\nabla_\theta \hat{\mathcal{J}}^\pi(\Theta) \approx \tfrac{1}{S}\textstyle\sum_{s=1}^S \sum_{t=1}^{T_s}((C(I_s^t, h_k^t) - b_\phi(I_s^t, h_k^t)) \cdot \nabla_\Theta \log \pi_\Theta(a_k^t|I_s^t, h_k^t)),$$

where $b_\phi(I_s^t, h_k^t)$ denotes the baseline function with parameters ϕ that are trained to minimize $L(\phi) = \frac{1}{S}\sum_s^S \sum_{t=1}^{T_s} ||b_\phi(I_s^t, h_k^t) - C(I_s^t, h_k^t)||^2$.

3.2.3 Environment Settings

The model's robustness and performance improvement over time need to be evaluated across various environmental settings that closely resemble real-life scenarios. This experimentation is

crucial to assess its ability to adapt and excel in diverse conditions. Therefore, we aim to examine key environmental factors that hold significance in classical SVRPTT research.

Inference. The performance evaluation of the model can be conducted by investigating its reliance on different inference strategies, encompassing widely employed methods like greedy sampling, random sampling, and beam search. In the case of greedy sampling, the action with the highest probability is chosen at each time step t to construct the routing solution. Random sampling generates multiple solutions by randomly selecting n_s actions in accordance with their respective probability distributions at each time step. In beam search, the top n_b cumulative most probable actions are selected. The utilization of these distinct inference strategies may yield disparate routing solutions, thereby leading to diverse performance outcomes.

Signal Ratio. The three components of θ_i, namely the constant term, weather-related factors, and random noise, are likely to exert varying influences on the distribution of stochastic variables. In order to quantify this influence, we introduce signal ratio variables A_i, B_i, Γ_i and investigate the impact of different signal ratio values on the model's performance.

$$A_i = \frac{\bar{\theta_i^2}}{T_i} \qquad B_i = \frac{\mathbb{E}\left[(\sum_m \sum_n \alpha_{imn} w_m w_n)^2\right]}{T_i}, \qquad \Gamma_i = \frac{\mathbb{E}\left[\epsilon_i^2\right]}{T_i},$$

$$T_i = \bar{\theta_i^2} + \mathbb{E}[(\sum\nolimits_m \sum\nolimits_n \alpha_{imn} w_m w_n)^2] + \mathbb{E}\left[\epsilon_i^2\right].$$

Fill rate. The performance of the model may be influenced by the maximum capacity of each vehicle. To quantify its impact, we introduce a fill rate variable, denoted as $\Phi = \frac{Q}{\mathbb{E}[\xi_i]}$, which represents the relative value of the maximum capacity in relation to the expected value of the total demand.

A Priori vs Reoptimization. In SVRPTT, there are two distinct approaches that are both significant in industry and warrant consideration. The first approach, known as a priori, involves the agent constructing the entire route without prior knowledge of the realizations of stochastic variables. On the other hand, the reoptimization approach allows the agent to unveil the realizations of variables as it visits the corresponding customers and arcs. This enables the agent to dynamically update the routes based on the acquired information, thus addressing the problem in an online fashion.

Variable Estimates. Prior to constructing the route, the agent can utilize estimates of the stochastic variables as inputs, denoted as $\hat{\theta}_i$. In this study, we propose two types of estimates: the constant estimate and the k-NN estimate. The constant estimate simply represents the expected value of the stochastic variable, $\hat{\theta}_i = \bar{\theta}_i$. The k-NN estimate, on the other hand, involves the utilization of a historical weather dataset, which comprises tuples of $(w^g, \mu_{ig}, c_{i0g}, ..., c_{iNg})$. By employing the k-NN algorithm, the agent can leverage the current w vector to identify the expected value from the w_g closest tuples in the weather dataset,

$$\hat{d}_i = \left(\sum\nolimits_{w_g \in N(w)} \frac{1}{||w_g - w||} \mu_{ig}\right) / \left(\sum\nolimits_{w_g \in N(w)} \frac{1}{||w_g - w||}\right), \tag{3.12}$$

$$\hat{c}_{ij} = \left(\sum\nolimits_{w_g \in N(w)} \frac{1}{||w_g - w||} c_{ijg}\right) / \left(\sum\nolimits_{w_g \in N(w)} \frac{1}{||w_g - w||}\right). \tag{3.13}$$

Customer Positions. In this study, we introduce two distinct settings for customer positioning: fixed and flexible. In the fixed positioning setting, the positions of customers are predetermined and

remain unchanged during the training and inference phases. As a result, the constant component of travel times, denoted as $\bar{c}_{ij}$, remains constant throughout. This setting may yield improved performance due to the relatively less challenging nature of the problem. It is particularly relevant for businesses operating with fixed customers (e.g. B2B delivery). In contrast, the flexible customer positioning setting allows for varying customer positions in each problem instance. This results in different $\bar{c}_{ij}$ values for each instance (relevant to B2C delivery).

Delivery Types. There are two distinct delivery modes of goods: full delivery and partial delivery. In the full delivery mode, the vehicle is only capable of delivering goods to a customer in their entirety, meaning that the goods are indivisible. If a customer has a demand of μ_{is}, the vehicle must deliver the entire quantity at once. If the vehicle's capacity is insufficient, it must perform a recourse action. On the other hand, in the partial delivery mode, the goods can be divided, allowing the vehicle to deliver them in partial quantities.

3.2.4 Architecture Details

Customer Information. We present the architectural framework for the models illustrated in Figure 3.7. To obtain continuous representations of customers, referred to as state embeddings, we utilize a 1D convolutional layer with D filters. This layer plays a crucial role in extracting meaningful and continuous representations of the customer-related information $(w, d_i^t, c_{i0}^t, ..., c_{iN}^t)$.

Vehicle Information. The vehicle-related information, including the current position p_k^t and load q_k^t, is processed using an LSTM (Long Short-Term Memory) module with a state size of D. This LSTM module captures the temporal dependencies and generates the memory embedding h_k^t. The memory layer allows the model to retain and utilize information from previous time steps $[1, ..., t-1]$ regarding each vehicle k.

Next Node Probabilities. To combine the state and memory embeddings effectively, we employ an attention layer. This attention mechanism enables the model to assign importance weights to all node embeddings based on their relevance. The attention layer produces the output $P(p_k^{t+1}|\cdot)$, which represents the log-probability of each node i (including the depot) being the next node in the route of vehicle k.

Masking. To ensure the generation of feasible solutions, we employ a masking procedure. Specifically, we assign a value of $-\infty$ to the probability of infeasible position $P(p_k^{t+1}|\cdot)$ corresponding to customer nodes whose demand has already been fulfilled. By applying this masking technique, we effectively exclude these already serviced customer nodes from the set of potential next nodes in the route for each vehicle.

3.2.5 Experiments

Nodes. The model was evaluated on four distinct problem sizes involving 10, 20, 50, and 100 customers. To set up the problem instances, we randomly generated the positions of these customers on a 2D map defined by the interval $[0, 1] \times [0, 1]$. The depot was located at coordinates $(0.5, 0.5)$ as a fixed reference point.

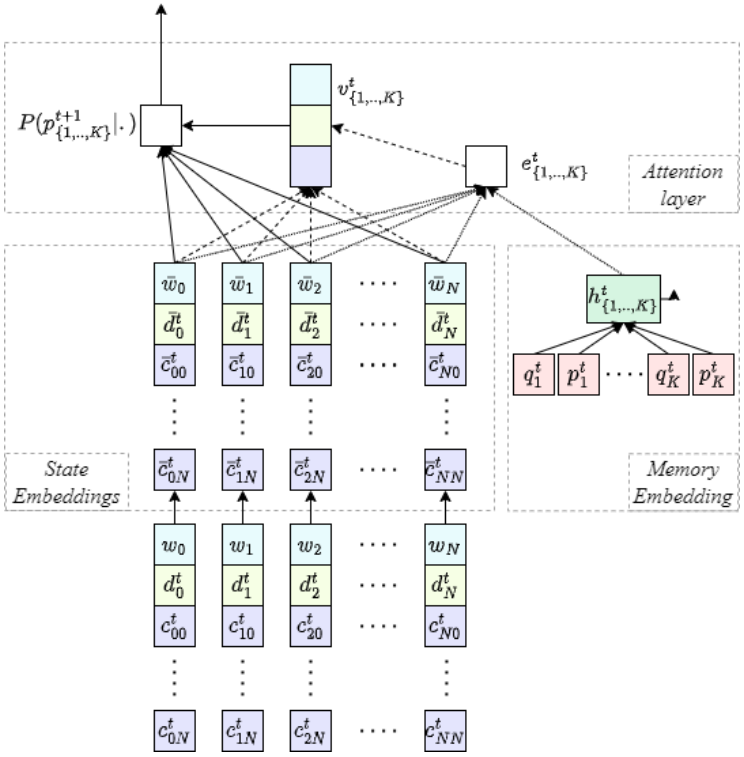


Figure 3.7: **Network architecture.** The lower section illustrates the model's input, which is composed of three elements: weather data w, the varying demand d_i^t, and travel times c_{ij}^t. The model generates State Embeddings for each customer and encodes the vehicle's load (q_k^t) and current position (p_k^t) as the Memory Embedding (h_k^t). The embeddings are subsequently combined using an Attention Layer, resulting in probabilities assigned to the nodes. These probabilities represent the likelihood of each node being the next position for vehicle k.

Variables. The weather variables W consist of three variables, each following a random uniform distribution within the range of $[-1, 1]$. The noise term ϵ is generated from a multivariate normal distribution $\mathcal{N}(0, \Sigma)$, where Σ is a non-zero covariance matrix. In the experiments, we vary the values of A_i, B_i, Γ_i, and Φ. To match these values, we generate the constant and noise components of the stochastic variables accordingly. By employing this generation process, we construct a weather dataset comprising 10,000 scenarios, represented by tuples $(w, \mu_{is}, c_{i0s}, ..., c_{iNs})$ for all $i \in N$. Additionally, we create a separate test dataset consisting of 1,000 scenarios for evaluation purposes.

Results

Baselines. To establish baseline models, we implemented stochastic approaches based on the works by [161], [112], [52]. Since ACO is state-of-the-art algorithm for the SVRPTT, we conducted a hyperparameter search on validation dataset to identify its optimal configuration for our specific environment. Table 3.11 displays the results obtained from the search, which involved exploring the pheromone importance, heuristic importance, and the number of ants. Table 3.12 illustrates results on test data achieved by baselines and RL with beam search inference ($n_b = 3$).

Table 3.11: The performance of the ACO algorithm on the SVRPTT validation dataset. The number of customers are represented in the columns. The results are presented as the average travel cost over the validation dataset, with lower values indicating better performance.

ACO PARAMETERS	10	20	50	100
3 / 4 / 20	3.73	7.61	15.88	34.86
6 / 8 / 24	3.70	7.56	15.77	34.63
4 / 9 / 15	3.67	7.49	15.62	34.31
3 / 10/ 12	**3.63**	**7.42**	**15.46**	**33.96**

Table 3.12: The results of the Baseline and RL models on the SVRPTT test dataset.

BASELINES	10	20	50	100
CLARKE-WRIGHT	3.88	7.93	16.53	36.30
TABU SEARCH	3.76	7.67	16.00	35.13
ANT-COLONY OPIMIZATION	3.63	7.42	15.46	33.96
RL-BEAM SEARCH	**3.51**	**7.16**	**14.93**	**32.78**

The lower values of travel cost show better performance of the model. For fair comparison, the baselines and RL are tested in the same environment configuration (a priori, fixed customers, partial delivery, same distributions and fill rate). RL shows better performance on all sizes with 3.31%, 3.50%, 3.43% and 3.47% (3.43% on average) less travel cost compared to ACO results.

Environment Settings. To evaluate the sensitivity of the proposed model, we conducted experiments using the default configuration. This configuration included the use of beam search with a beam width of $n_b = 3$ for inference, k-NN variable estimates, fixed customer positions, partial delivery, signal ratios $A, B, \Gamma = 0.6, 0.2, 0.2$ respectively, a fill rate of $\Phi = 0.5$ and a single vehicle $K = 1$. The experimental results presented in Table 3.13 provide insights into the performance of different **inference strategies** with parameter settings of $n_s = 16$ and $n_b = 3$. The findings indicate that beam search exhibits superior performance compared to the other strategies, highlighting its effectiveness in optimizing the model's output. The comparison presented in Table 3.14 demonstrates that the k-NN estimation method surpasses the performance of constant demands.

Table 3.15 presents the results obtained from experiments conducted to evaluate the impact of different types of stochasticity. The first experiment involved a fully deterministic model, which naturally demonstrated the best performance. In the second experiment, stochastic demand was introduced, with a ratio of $A, B, \Gamma = 0.6, 0.2, 0.2$. The third experiment introduced stochastic travel cost using the same ratio. The results indicate that stochastic travel cost has a greater impact on performance compared to stochastic demand. This can be attributed to the fact that travel cost is influenced by a larger number of variables compared to demand. In the fourth experiment, both sources of stochasticity were combined, resulting in a more challenging problem setting.

Table 3.16 presents the impact of varying signal ratio settings on the performance of the model. The findings indicate that increasing the signal from the weather variable leads to a reduction in travel times. This suggests that the RL model is capable of capturing and leveraging the implicit influence of weather on demands and travel times. It is worth noting that this impact may vary across different customers, yet the model can effectively utilize this information to enhance routing strategies. The performance of the model was evaluated through experiments examining

Table 3.13: The performance of inference strategies on the SVRPTT test dataset. The reported results pertain to both the a priori (Apr.) and reoptimization (Reopt.) settings.

	10		20		50		100	
INFERENCE STRATEGIES	APR.	REOPT.	APR.	REOPT.	APR.	REOPT.	APR.	REOPT.
RL-GREEDY	3.98	3.86	8.12	7.88	16.93	16.42	37.18	36.06
RL-SAMPLING	3.69	3.58	7.54	7.32	15.73	15.26	34.54	33.50
RL-BEAM-SEARCH	**3.51**	**3.40**	**7.16**	**6.95**	**14.93**	**14.48**	**32.78**	**31.80**

Table 3.14: The results obtained from two different types of variable estimates on the SVRPTT test dataset.

	10		20		50		100	
Variable Estimate	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.
Constant variable estimate	3.76	3.64	7.67	7.44	16.0	15.52	35.13	34.07
kNN variable estimate	**3.51**	**3.40**	**7.16**	**6.95**	**14.93**	**14.48**	**32.78**	**31.80**

Table 3.15: The results of various stochasticity sources on the SVRPTT test dataset.

	10		20		50		100	
Stochasticity Sources	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.
No Stochasticity	2.88	2.79	5.88	5.71	12.26	11.90	26.93	26.12
Stochastic Demand	3.13	3.04	6.39	6.20	13.33	12.93	29.27	28.39
Stochastic Travel Cost	3.38	3.28	6.90	6.70	14.40	13.96	31.61	30.67
Stochastic Demand / Travel Cost	3.51	3.40	7.16	6.95	14.93	14.48	32.78	31.80

the influence of **customer positions** on the outcomes, as depicted in Table 3.17. The findings suggest that fixed customer positions yield superior performance on the test data.

We tested the performance variation between baseline models and RL models under correlated and uncorrelated (demand, travel cost) settings (Figure 3.8). It is evident from the results that the baseline models lack the ability to capture and utilize the correlations between customers, leading to suboptimal routing decisions. In contrast, the RL models demonstrate the capability to exploit these correlations, resulting in more cost-effective routes.

Table 3.18 presents the findings from experiments conducted with different values of the signal ratio. The results indicate that increasing the fill rate, which corresponds to a larger capacity of the vehicle, facilitates the routing task. This is reflected in the improved performance of the model, as it is able to identify more optimal routes under such conditions. The experiments were conducted with partial delivery, which allows the agent to deliver goods in a more convenient manner, leading to a comparatively easier problem setting. As reflected in the results, the performance of the model is higher under the partial delivery setting, indicating its ability to leverage this condition to identify more efficient routes (Table 3.21).

Multi-Vehicle Scenarios. Table 3.19 presents the results of experiments conducted to assess the impact of the number of vehicles on the performance of the model. Theoretically, increasing the number of vehicles should simplify the problem by enabling the division of a large route into multiple smaller routes, resulting in fewer recourse actions and reduced total length. Consistent with this notion, our findings demonstrate that as the number of vehicles increases, the model's

Table 3.16: The results obtained from various signal ratios on the SVRPTT test dataset.

	10		20		50		100	
A/B/Γ	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.
0.8 / 0.0 / 0.2	3.41	3.31	6.97	6.76	14.53	14.09	31.91	30.95
0.8 / 0.2 / 0.0	**3.35**	**3.25**	**6.84**	**6.64**	**14.26**	**13.84**	**31.32**	**30.38**
0.6 / 0.2 / 0.2	3.51	3.40	7.16	6.95	14.93	14.48	32.78	31.80
0.4 / 0.3 / 0.3	3.76	3.64	7.67	7.44	16.00	15.52	35.13	34.07

Table 3.17: The results obtained from two types of positions on the SVRPTT test dataset.

	10		20		50		100	
Customers Position	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.
Flexible customer positions	3.88	3.77	7.93	7.69	16.53	16.03	36.30	35.21
Fixed customer positions	**3.51**	**3.40**	**7.16**	**6.95**	**14.93**	**14.48**	**32.78**	**31.80**

Table 3.18: The results obtained from various fill rates on the SVRPTT test dataset.

	10		20		50		100	
Fill Rate	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.
0.1	3.88	3.77	7.93	7.69	16.53	16.03	36.30	35.21
0.5	3.51	3.40	7.16	6.95	14.93	14.48	32.78	31.80
0.9	**3.29**	**3.19**	**6.71**	**6.51**	**14.00**	**13.58**	**30.74**	**29.81**

performance improves.

Time Complexity. Table 3.20 presents the inference times of RL models and baselines for a single problem instance. It is observed that RL and CW models exhibit inference times that are suitable for real-time solving, indicating their efficiency in finding solutions for industry (hundreds of customers). On the other hand, ACO and Tabu search models demonstrate inference times that have an exponential dependence on the problem size, suggesting that their computational requirements increase significantly as the problem becomes larger.

3.2.6 Conclusion

In conclusion, the Vehicle Routing Problem with Stochastic Travel Times (SVRPTT) is a challenging yet significant task in the industry. It is characterized by two crucial sources of stochasticity: stochastic customer demands and stochastic travel times. In this study, we have presented a formulation of the problem that incorporates both sources of uncertainty. Furthermore, we have proposed an RL agent architecture along with a training methodology to effectively address the formulated problem. In real-world industrial applications, companies often leverage available information that can potentially influence stochastic demand and travel times. Weather information is one such example that can introduce correlation among customers. In our study, we integrated this type of information into the environment and architecture of our model. The results revealed that the RL agent (Table 3.16), when provided with access to weather information, was able to achieve **routing strategies that outperformed those without such information by up to 2%**.

This finding highlights the significance of incorporating relevant external factors, such as

Table 3.19: The results obtained from various number of vehicles on the SVRP test dataset.

	10		20		50		100	
Vehicle Number	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.
1	3.51	3.40	7.16	6.95	14.93	14.48	32.78	31.80
2	3.43	3.33	7.01	6.80	14.61	14.17	32.08	31.12
3	3.41	3.31	6.96	6.75	14.52	14.08	31.88	30.92
5	**3.40**	**3.29**	**6.94**	**6.73**	**14.46**	**14.03**	**31.76**	**30.81**

Table 3.20: Inference time results in seconds on the SVRPTT test dataset.

Baseline	10	20	50	100
Clarke-Wright	0.005	0.016	0.058	0.185
Tabu Search	0.964	2.936	8.672	57.12
Ant-Colony Optimization	1.230	4.291	27.64	123.7
RL-greedy	0.061	0.121	0.191	0.394
RL-sampling	0.064	0.125	0.232	0.405
RL-beam search	0.072	0.178	0.293	0.416

weather, into the modeling and decision-making processes of stochastic routing problems. By utilizing this additional information, the RL agent was able to effectively capture and exploit the correlations among customers, resulting in improved routing strategies and ultimately leading to enhanced operational efficiency in real-world scenarios. We conducted a set of evaluations of the proposed model in various settings relevant to the VRP in industrial applications. The results demonstrate the robustness of the model in learning effective routing strategies across all tested scenarios. Furthermore, we compared the model to a state-of-the-art approach introduced by [52], and our **RL agent exhibited a superior performance, achieving a significant 3.43% improvement** in travel cost.

The primary focus of this work was to introduce the first RL model capable of addressing the VRP with stochastic demands and travel times and to showcase its superiority over classical models commonly employed in industrial settings. The results indicate that RL has the potential to replace traditional approaches and significantly reduce routing costs, consequently leading to reductions in logistics expenses and negative environmental impact. Future research in this field can delve deeper into the application of RL for the SVRPTT and explore the development of alternative architectures and training methods to further enhance the performance of the model. These **efforts can contribute to the advancement of efficient and sustainable routing solutions** in logistics and transportation domains.

3.2.7 Implementation

Parameters. The resulting outputs, representing $P(p_{1:K}^{t+1}|\cdot)$, are passed through a Critic network, which consists of two fully-connected layers, with D and 1 neurons. The Rectified Linear Unit (ReLU) activation function is employed. The dimensionality of the embeddings D is set to 128.

Training. We employ the Xavier initialization method. During the training phase, we utilize the Adam optimizer with a learning rate of 10^{-4}. To prevent overfitting, we apply dropout with a

Table 3.21: The results obtained from various delivery types on the SVRPTT test dataset.

	10		20		50		100	
Delivery Type	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.
Full delivery only	3.85	3.73	7.86	7.63	16.40	15.90	36.00	34.92
Partial delivery	**3.51**	**3.40**	**7.16**	**6.95**	**14.93**	**14.48**	**32.78**	**31.80**

probability of 0.1. The model is trained on a GPU system consisting of a NVIDIA A100 SXM 40GB GPU and 2x AMD EPYC 7742 CPUs (8 cores) with 256GB RAM, for a total of 10,000 iterations for each problem size. The implementation of the model is available online [1].

3.2.8 Results

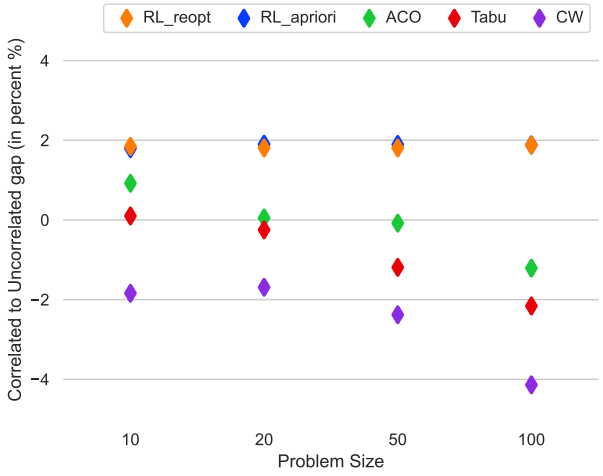


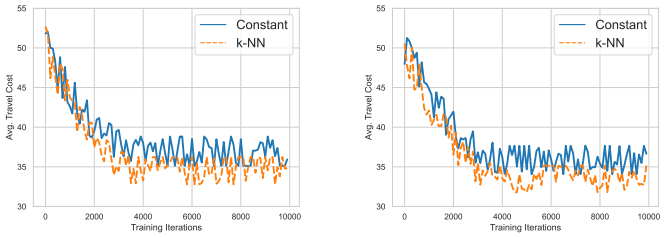
Figure 3.8: Percentage gap in travel times of correlated variables setting to $\mathrm{A}, \mathrm{B}, \Gamma = 0.8, 0.2, 0.0$ to uncorrelated variables with $\mathrm{A}, \mathrm{B}, \Gamma = 0.8, 0.0, 0.2$.

3.2.9 Training Curves

The training curves depicting the performance of different variable estimates are presented in Figure 3.9. The k-NN estimate outperforms the constant method, indicating its ability to learn and adapt to the problem dynamics more effectively.

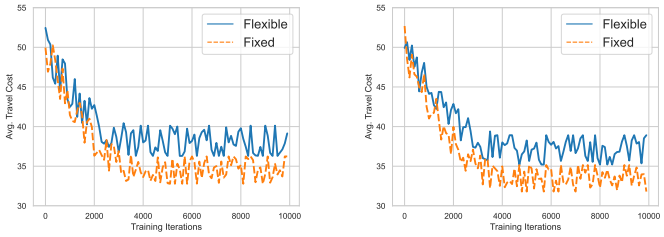
Figure 3.10 displays the training curves comparing the performance of different customer positioning approaches. Figure 3.11 presents the training curves illustrating the performance of different delivery types.

[1] `https://github.com/Zangir/SVRP`



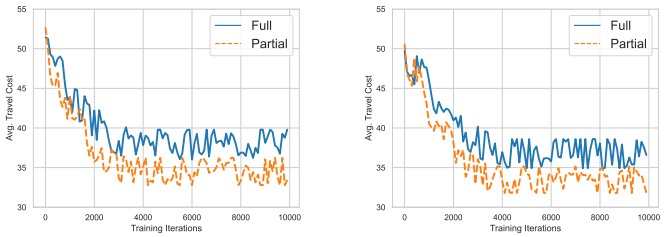
A priori Reoptimization

Figure 3.9: The travel cost during the training phase, using two types of variable estimates: Constant and k-NN. The results indicate robust learning in both cases.



A priori Reoptimization

Figure 3.10: The travel cost during the training phase, using two types of customer positions: fixed and flexible.



A priori Reoptimization

Figure 3.11: The travel cost during the training phase, using two types of delivery: full and partial.

3.3 Learning Algorithm for Stochastic Vehicle Routing Problem with Time Windows

3.3.1 Introduction

Stochastic Vehicle Routing Problem with time windows (SVRPTW) is a more complex formulation of the VRP, characterized by its unknown parameters that are only revealed after routes are completed. These uncertain parameters make the SVRP more challenging compared to the deterministic VRP; however, regardless of its challenges, SVRPTW reflects real-world scenarios frequently faced in the industrial sector, and therefore, solutions to the SVRPTW are more desir-

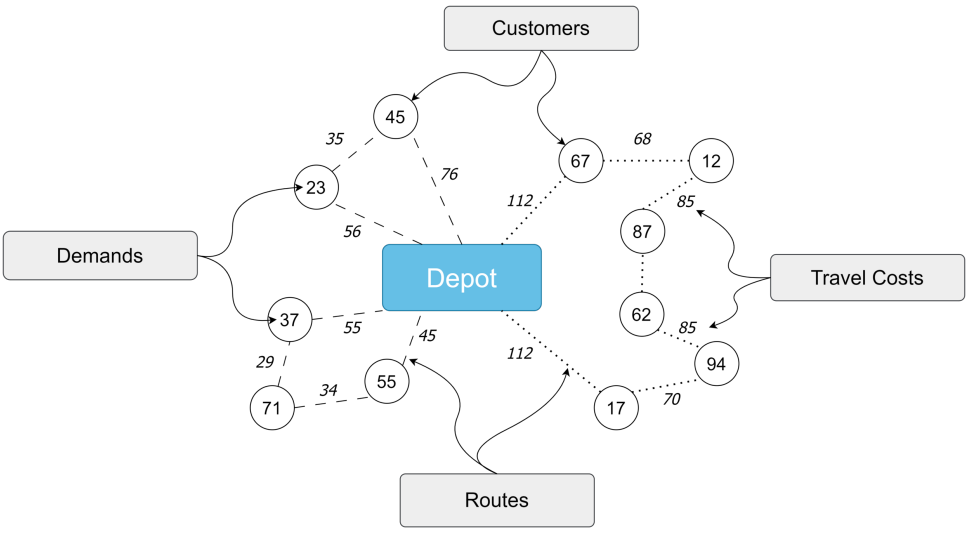


Figure 3.12: Vehicle Routing Problem depicted within this figure is an instance of the problem, characterized as a graph serving as a map representation. This graph comprises two distinct types of nodes: customer nodes and a central depot. Each of these nodes is geospatially situated in a two-dimensional coordinate system. The customer nodes are associated with numerical values representing the respective demand for specific goods by each customer. Additionally, the interconnections between nodes are annotated with travel times, denoting the expenses incurred while moving from one node to another. Notably, the depot node serves as the point of origin for the fleet of vehicles commencing their routes from this pivotal location. The operational procedure entails the vehicles departing from the depot and subsequently visiting the customer nodes individually in a sequential fashion. During these visits, the vehicles deliver the required quantity of goods as per the demand specified by the respective customer. It is worth emphasizing that the number of feasible routes is bounded, yet it grows exponentially with an increase in the number of customers in the problem instance.

able ([28, 51]). Traditional practices to tackle SVRPTW have been examined in various studies, but the application of deep RL to this formulation of the VRP remains unexplored.

Motivation. Our primary motivation for investigating the SVRPTW lies in the lack of research in this domain employing RL methodologies, although it may provide substantial advantages for businesses in transportation and logistics. They include, but are not limited to, lowering carbon emissions and reducing operational costs. In this research endeavor, we aim to enhance the applicability of our framework [76, 77] to supply-chain scenarios by incorporating time windows into the SVRP with two important sources of stochasticity - stochastic demand, and stochastic travel times. We conduct experiments to assess the dependence of our proposed model on different settings of the SVRPTW environment, which are frequently explored in classical research. Our specific **contributions** to this field are summarized as follows:

- We propose **the first RL-based model for SVRP** that integrates time windows, stochastic customer demands, and travel times, pivotal elements in practical logistics scenarios. Through a comparative analysis against contemporary SVRP methods, our RL model showcases superior performance, manifesting a 1.73% decrease in travel times in relation to the most proficient classical model [52].
- We conduct comprehensive experiments to evaluate the performance of our model across a spectrum of environmental configurations. These configurations encompass diverse inference methodologies, delivery modalities, levels of stochasticity, fleet and customer magnitudes. Our observations indicate that **the RL agent exhibits robustness across distinct environmental settings**, a crucial attribute for SVRP given its propensity for varied formulations in real-world scenarios.
- We demonstrate that **the model is capable to leverage external information**, facilitating the derivation of improved routing strategies. This aspect holds particular significance in the SVRP industry, an aspect that has hitherto received limited attention in scholarly discourse. We posit that our model constitutes a valuable baseline for both research and industrial applications, providing a foundation for prospective investigations.

3.3.2 Method

Formulation. In the SVRPTW, there exists inherent uncertainty as the actual values of customer demands and travel times remain undisclosed to the vehicle operator in advance. In this scenario, the agent possesses probabilistic or expected values for both the customer demands and travel times, but upon reaching a particular customer, there is a distinct possibility that either the demand or the travel times associated with that customer may deviate from their expected values. In addressing this stochastic variation, industries often employ implicit variables to manage the inherent uncertainty. For instance, companies may utilize dynamic information, such as weather forecasts for the following day, to adapt their expectations regarding travel times. This adaptive approach allows companies to make informed decisions and enhance the resilience of their vehicle routing strategies in the face of uncertain and dynamic factors. We solve the problem in an iterative manner, changing the positions of each vehicle at every time iteration t.

Notations. To formalize the problem, we establish the following mathematical notation. We define a set N, encompassing both the customers and the depot. For each customer i within this set, we introduce a variable d_i to denote their specific demand. Furthermore, we introduce a variable q_k for each vehicle k from a set of vehicles K, signifying the current cargo load carried by vehicle k with maximum load denoted as Q. This load represents the quantity of goods held by vehicle k at any given point in time. We introduce a cost variable c_{ij}, which characterizes the cost associated with traversing the arc connecting node i to node j. It quantifies the time incurred in moving from one node to another within the problem instance. Importantly, our problem formulation allows for the possibility of travel between any two nodes present in the graph. We introduce a binary decision variable x_{ij}, which assumes a value of one when vehicle k traverses the arc between nodes i and j as part of its route and takes on a value of zero otherwise. The binary nature of this variable effectively captures whether a given arc is utilized within the vehicle's route.

Failure Situation. In the context of the SVRPTW, scenarios may arise where a vehicle arrives at a customer location and finds that the customer's demand exceeds both the initially estimated value and the current load of the vehicle. In such instances, the vehicle is compelled to return to the depot to replenish its full capacity, $q_k = Q$, before returning to address the demand of this particular customer. This necessitates incurring an additional cost associated with the so-called failure situation, which entail substantial expenses within industrial contexts and warrant the implementation of improved routing strategies. To address this challenge, one potential solution involves anticipating higher demand from a specific distant customer and strategically refilling the vehicle in advance while in proximity to the depot. However, failure situations may still be presented in the solution. To catch them, we introduce the binary variable r_i representing the enforced decision to undertake the action of returning to the depot from customer i for refilling. Specifically, if $r_i = 1$, it indicates the necessity of the vehicle to proceed to the depot from customer i for refilling purposes.

Objective. The primary objective of the agent is to minimize the aggregate cost, encompassing the total cost of all routes and the costs incurred due to failure situations while ensuring the satisfaction of customer demands within the graph, i.e.:

$$\text{minimize} \quad \sum_{i,j \in C} c_{ij} x_{ij} + 2 \sum_{i \in C} r_i c_{0i}.$$

External Variables. We introduce a set of external variables W, with their values known to the agent. These variables exert a nonlinear influence on the realizations of stochastic variables, specifically demands and travel times. In industrial applications, logistic companies often leverage such external variables to refine their estimations of stochastic variables. For instance, for a company specializing in ice cream sales, knowledge of weather conditions enables more accurate estimations of ice cream demand for the following day. In this context, we define a set of external variables, each conforming to a certain distribution.

Stochastic Variables. The agent lacks knowledge regarding the distributions and exact values of stochastic variables. Nevertheless, the agent may possess estimations for the realizations of these variables, along with the values of external variables. The external variables exert an impact on the stochastic demand and travel cost according to the following relationship:

$$\theta_i = \bar{\theta}_i + \sum_m \sum_n \alpha_{imn} w_m w_n + \epsilon_i, \ \theta_i \in \{d_i, c_{i0}, ..., c_{iN}\},$$

where θ_i serves as a stochastic variable, representing either the demand associated with customer i or the travel cost from customer i to other nodes. The value of θ_i is a composite of a fixed term, denoted as $\bar{\theta}_i$, the cumulative effect of the weighted interaction between all pairs $w_m \in W$ and $w_n \in W$, and the random noise denoted as ϵ_i. Regarding travel times, the fixed term is defined as the Euclidean distance between two nodes divided by a constant speed. The travel cost, in this context, is intended to signify the time required to travel from one node to another. It is noteworthy that the agent lacks explicit knowledge concerning the distribution of variables within the set W or their influence on θ_i. However, the agent is endowed with access to the observed realizations of variables in W, affording the opportunity to implicitly learn the nonlinear mapping from W to stochastic variables.

Time Windows. We define a set of customer types to clusterize the customers (e.g., office workers vs housekeepers). In the problem-solving process, we maintain a record of the current environmental time denoted as t, measured in hours, with travel times also quantified in hours. Additionally, we establish time windows and handout times for each customer, stratified by customer type. Each day is characterized by h_{max} hours. To encapsulate the temporal availability of each customer, we introduce a binary vector H_i for every customer i. This vector conveys the customer's availability at each hour of the day. Specifically, $H_i^t = 1$ signifies the availability of customer i at hour t, indicating the feasibility of delivering the product during that period. Distinct probabilities dependent on customer types govern the availability of customers during specific hours. For each problem instance, we generate the H_i vector based on this probability distribution. The agent possesses information regarding the customer type, current time, and realization of the H_i vector for each customer.

Routing Policy

State. Within the context of the RL framework, the state representation encompasses a multi-faceted set of variables, including the current customer demands, travel times, and the present positions and cargo loads of the vehicles. Our formulation delineates the environment through a state representation (I, L), wherein I encapsulates information pertaining to each customer, and L encapsulates information pertinent to each vehicle.

$$I \doteq \{(W_i, d_i, C_i, H_i, Z_i), i \in N\},\ L \doteq \{(q_k, p_k, t_k), k \in K\},$$

where W_i is the vector of external variables, d_i represents the realization of the demand for customer i, C_i signifies the vector of realizations of travel times from customer i to other nodes. H_i denotes the time windows vector of customer i, and Z_i is a binary vector indicating the type of customer, q_k represents the load of vehicle k, p denotes its position, and t signifies the current time. This constitutes the environment time, which is similar across all vehicles. It is crucial to highlight that the values of customer demand, vehicle capacity, and vehicle position are subject to change as time progresses.

Action. The actions undertaken within this framework pertain to the manipulation of the vehicle's spatial positioning. At every discrete time step t, the agent makes decisions by selecting an action a^t. Here, the action pertains to determining the subsequent position p_k^{t+1} of each vehicle k. The decision-making process is dynamic and is orchestrated by a policy $a^t \sim \pi(\cdot|I^t, L^t)$.

Indicating that the actions at time t are drawn from a probability distribution governed by the policy π, conditioned on the current state (I^t, L^t). Succinctly, the policy dictates the likelihood of selecting the next node for each vehicle based on the prevailing state information.

Transitions. The transition function governing state dynamics encompasses alterations in vehicle position, cargo load, and customer demand. When a vehicle arrives to customer i at time t, it mitigates the demand for time $t+1$ by an extent equivalent to the current cargo load carried by vehicle k. It is imperative to underscore that the minimum allowable demand is set at zero. Simultaneously, the cargo load of vehicle k diminishes by the magnitude of the current demand associated with customer i. When the cargo load of vehicle k reaches zero, it is incumbent upon the vehicle to return to the depot for replenishment, thereby restoring its cargo-carrying capacity as follows:

$$d_i^{t+1} = \max\left\{0, d_i^t - q_k^t\right\},\ d_j^{t+1} = d_j^t, \text{ for } j \neq i, \text{ and } q_k^{t+1} = \max\left\{0, q_k^t - d_i^t\right\}. \tag{3.14}$$

Objective. The objective of the RL model is the minimization of the anticipated travel cost, entailing an optimization process that seeks to enhance the efficiency and cost-effectiveness of vehicle routing. The objective of the agent is to formulate an optimal policy π with parameters Θ through the utilization of the Reinforce algorithm [215]. This algorithm is designed to minimize the cumulative expected cost as follows:

$$\mathcal{J}^\pi(\Theta) = \mathbb{E}[\textstyle\sum_t^T C_t(I^t, L^t, a^t)].$$

The cost function $C_t(I^t, L^t, a^t)$ is defined as the summation of all traversal costs subsequent to the execution of action a^t. Mathematically, this is expressed as $\sum_{k=1}^K c_{i_k j_k}$. In instances where vehicle k necessitates recourse action due to a failure, an additional recourse cost of $2c_{0j_k}$ is incurred. The objective function is aggregated across T time steps, and the gradient of the objective function [183] is computed in the following manner:

$$\nabla_\theta \hat{\mathcal{J}}^\pi(\Theta) \approx \textstyle\sum_{t=1}^T ((C(I^t, L^t, a^t) - b_\phi(I^t, L^t)) \cdot \nabla_\Theta \log \pi_\Theta(a^t | I^t, L^t)).$$

In this context, $b_\phi(I^t, L^t)$ represents the baseline function characterized by parameters ϕ, and these parameters are optimized through training to minimize

$$L(\phi) = \textstyle\sum_{t=1}^T ||b_\phi(I^t, L^t) - C(I^t, L^t, a^t)||^2.$$

Environment Settings

In this section, we explore various approaches for setting up the environment. We deem it crucial to address these approaches, as each setting can find its relevance in diverse industrial contexts.

Inference. For the purposes of inference, we will employ three distinct inference strategies commonly utilized in reinforcement learning literature. The first strategy is greedy inference, wherein we choose only one action at each time step t with the highest probability determined by the policy function [7, 102]. The second strategy is sampling inference, involving the selection of n_s actions at each time step based on their respective probability distributions. The third strategy

is beam search inference, where n_b trajectories of actions with the highest probability distributions are chosen [87, 203].

Signal Ratio. When constructing stochastic variables, we employ the signal ratio, a metric that gauges the impact of each component on the variable itself. Consequently, three distinct signal ratios are computed for the fixed term A_i, external variables term B_i, and noise Γ_i. Each signal ratio quantifies the proportion of the squared value of the stochastic variable contributed by the squared value of its respective terms. The signal ratio is a scalar measure ranging between zero and one computed as follows:

$$A_i = \frac{\bar{\theta_i^2}}{T_i} \quad B_i = \frac{\mathbb{E}\left[(\sum_m \sum_n \alpha_{imn} w_m w_n)^2\right]}{T_i}, \quad \Gamma_i = \frac{\mathbb{E}\left[\epsilon_i^2\right]}{T_i},$$

$$T_i = \bar{\theta_i^2} + \mathbb{E}[(\sum\nolimits_m \sum\nolimits_n \alpha_{imn} w_m w_n)^2] + \mathbb{E}\left[\epsilon_i^2\right].$$

Fill rate. We additionally quantify the fill rate, defined as the ratio of the maximum capacity Q of each vehicle to the aggregated expected demand from all customers

$$\Phi = \frac{Q}{\mathbb{E}[d_i]}.$$

Variable Estimates. Prior to disclosing the actual realizations of stochastic variables, agents are equipped with initial estimates of these variables using fixed terms. The agent employs these fixed terms as initial approximations for the stochastic variables and updates them subsequent to the revelation of their true realizations.

Apriori vs. Reoptimization. Two distinct approaches, namely apriori and reoptimization, are utilized for inference. In the apriori approach, the agent devises all routes solely based on estimates of stochastic variables, with realizations being disclosed after problem resolution. In the reoptimization approach, the agent discloses stochastic variable realizations during the creation of the solution. For instance, the agent discloses the actual demand of specific customers upon reaching them on the route. Both approaches find applicability in industrial scenarios and, therefore, merit evaluation.

Customer Positions. Two approaches are considered regarding customer positions: flexible and fixed. In the flexible approach, it is assumed that for each problem instance, the locations of customers on the map can vary. Conversely, in the fixed approach, the locations of customers remain constant, although stochastic travel demands and travel times can still be incorporated.

Delivery Types. Two delivery approaches can be considered. The first approach is partial delivery, allowing for the fulfillment of demand in part initially, with the remaining fulfillment scheduled for subsequent time steps. The second approach is exclusive to full delivery, restricting the fulfillment of demand to occur only within the same time step.

3.3.3 Architecture

Inputs. The architecture comprises three primary components (Figure 3.13). The first pertains to customer-related information input, encompassing details such as external variables, demand

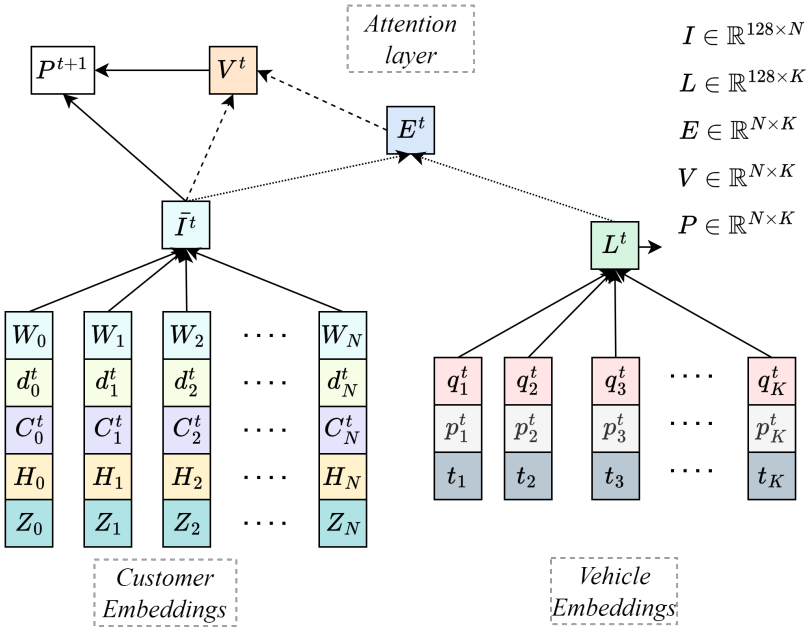


Figure 3.13: **Model Architecture.** The lower segment depicts the input structure of the model. The model produces embeddings for two input types: Customer and Vehicle. These embeddings are then merged through an Attention Layer, yielding probabilities assigned to the nodes. These probabilities indicate the likelihood of each node being the subsequent position for each vehicle. 0 node stands for the depot that has zero demand and is available all the time. Ultimately, the probabilities undergo a masking process to exclude customers within the route that have already been satisfied.

estimation, travel cost estimations, customer type, and time windows. The second input pertains to vehicle information, involving the load, position, and time t for each vehicle. Time is encoded as a one-hot vector, with 1 representing the current hour.

Embeddings. Customer inputs undergo a one-dimensional convolutional layer to generate 128-dimensional embeddings for each customer. Vehicle inputs undergo an LSTM layer, yielding 128-size embeddings for each vehicle. This layer is designed to retain information about the routes traversed by each vehicle. Although previous routes theoretically do not impact future costs, in practice, the memory layer has shown a positive impact on model performance. Vehicle and customer embeddings pass through an attention layer, producing a matrix P^{t+1} with a number of rows equal to the number of nodes and a number of columns equal to the number of vehicles.

Probabilities. The matrix P^{t+1} columns undergo the softmax function to generate probabilities for each vehicle to visit each node (customer and depot). This fundamental architecture serves well for solving problems with varying numbers of customers and vehicles, establishing a baseline for future in-depth research on SVRPTW. These probabilities undergo masking, wherein the probabilities corresponding to customers whose demand has already been fulfilled are set to zero. Additionally, situations are possible where two vehicles could potentially visit the same customer, particularly when a customer's demand exceeds the capacities of each vehicle individually.

3.3.4 Experiments

Dataset. During the training process for each batch, a two-dimensional map with coordinates ranging from zero to one is simulated. The depot is positioned at the center of this map. Uniform distribution is employed to generate values for three external variables, which are then utilized to simulate demands and travel times. The simulation is conducted in a manner that adheres to the specified default signal ratio values for each experiment. Employing the same procedure, a test dataset comprising 1000 problem instances is generated and consistently used across all experiments. The model is trained and tested across scenarios involving 10, 20, 50, and 100 customers, employing one to five vehicles in multi-vehicle settings.

Default Environment. The default environmental configuration involved employing beam search inference with $n_b = 3$, fixed-term stochastic variable estimations, fixed customer positions, partial type delivery, a fill rate of $\Phi = 0.5$, a single vehicle, and signal ratios A, B, Γ set to $0.6, 0.2, 0.2$, respectively.

Results

Baselines. For our baseline comparisons, we implemented methodologies proposed in [161], [112], and [52]. Given the sensitivity of the ACO model, we conducted its hyperparameter search. The outcomes of this search are presented in Table 3.22. Hyperparameters specified in the respective papers were employed for the other baselines.

Subsequently, we assessed the performance of our proposed model against these baselines, as delineated in Table 3.23. The results demonstrate that our model, employing beam search inference, consistently achieves an average reduction of 1.73% in travel times across varying

Table 3.22: The performance of the ACO algorithm. The SVRPTW validation dataset was utilized to assess the efficacy of different hyperparameter settings (only best results are depicted). Unlike the SVRPCD case, this dataset includes stochastic travel times. Each column corresponds to a varying number of customers. The outcomes are depicted in terms of the average travel cost across the validation dataset, where diminished values signify superior performance.

ACO PARAMETERS	10	20	50	100
3 / 4 / 20	6.77	11.51	22.82	54.13
6 / 8 / 24	6.73	11.44	22.67	53.77
4 / 9 / 15	6.67	11.33	22.46	53.27
3 / 10/ 12	**6.48**	**11.02**	**21.84**	**51.81**

Table 3.23: The results obtained from the baseline and RL models on the SVRPTW test dataset.

BASELINES	10	20	50	100
CLARKE-WRIGHT	6.88	11.70	23.19	54.99
TABU SEARCH	6.66	11.31	22.42	53.17
ANT-COLONY OPTIMIZATION	6.48	11.02	21.84	51.81
RL-BEAM SEARCH	**6.37**	**10.83**	**21.46**	**50.90**

problem sizes compared to the ACO model. To ensure a fair comparison, we maintained a consistent default environment setting with apriori inference across all models in this evaluation.

Robustness to Environment Changes. To assess the model's robustness across various formulations of the SVRP environment, we conducted evaluations under diverse settings. Our experimentation spanned different inference strategies, varying levels of environmental stochasticity, signal ratios, fill rates, and customer positions.

In terms of inference strategies, we compared the performance of greedy inference against random sampling inference (with a width set to 16) and beam search inference (with a width set to three). The outcomes of these experiments, detailed in Table 3.24, illustrate that random sampling outperforms greedy inference, while beam search exhibits superior performance compared to both. Consequently, we adopted beam search inference for all subsequent experiments.

In the subsequent experiment, we evaluated the model's performance across different levels of stochasticity in the environment. The first experiment involved a fully deterministic setting,

Table 3.24: The effectiveness of inference strategies on the SVRPTW test dataset, with reported outcomes for both the a priori (Apr.) and reoptimization (Reopt.) configurations.

	10		20		50		100	
INFERENCE STRATEGIES	APR.	REOPT.	APR.	REOPT.	APR.	REOPT.	APR.	REOPT.
RL-GREEDY	7.22	7.01	12.28	11.91	24.34	23.61	57.72	55.99
RL-SAMPLING	6.71	6.51	11.41	11.07	22.61	21.93	53.63	52.02
RL-BEAM-SEARCH	**6.37**	**6.18**	**10.83**	**10.50**	**21.46**	**20.82**	**50.90**	**49.37**

Table 3.25: The results of different stochasticity settings on the SVRPTW test dataset.

	10		20		50		100	
Stochasticity Sources	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.
No Stochasticity	2.88	2.79	5.88	5.71	12.26	11.90	26.93	26.12
Stochastic Demand	3.13	3.04	6.39	6.20	13.33	12.93	29.27	28.39
Stochastic Travel Cost	3.38	3.28	6.90	6.70	14.40	13.96	31.61	30.67
Stochastic Demand / Travel Cost	3.51	3.40	7.16	6.95	14.93	14.48	32.78	31.80
Stochastic Demand / Travel Cost / Time Windows	6.37	6.18	10.83	10.50	21.46	20.82	50.90	49.37

Table 3.26: The results derived from different signal ratios on the SVRPTW test dataset.

	10		20		50		100	
$\mathrm{A}/\mathrm{B}/\Gamma$	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.
0.8 / 0.0 / 0.2	6.74	6.59	10.92	10.61	21.83	21.47	51.38	50.61
0.8 / 0.2 / 0.0	**6.31**	**6.12**	**10.73**	**10.41**	**21.27**	**20.63**	**50.45**	**48.93**
0.6 / 0.2 / 0.2	6.37	6.18	10.83	10.50	21.46	20.82	50.90	49.37
0.4 / 0.3 / 0.3	6.83	6.62	11.60	11.25	22.99	22.30	54.54	52.90

where the values of demand and travel times were solely determined by their constant components. In the second experiment, we introduced stochastic demand with default ratio values $A, B, \Gamma = 0.6, 0.2, 0.2$. The third experiment focused on an environment featuring only stochastic travel times, again with default ratio values. Subsequently, the model was tested in an environment with both stochastic demands and travel times. Finally, we introduced time windows to the environment. The outcomes of these experiments are presented in Table 3.25. It is evident that with each added layer of complexity, the average travel times increase. However, the model consistently demonstrates its capacity to adapt to varying problem conditions and optimize routes accordingly.

We fixed the environment, incorporating stochastic demands, travel times, and time windows while manipulating the values of the signal ratios. The outcomes of this experiment are displayed in Table 3.26. The results illustrate that as the impact of the noise term intensifies, the stochastic variable becomes more unpredictable, resulting in higher average travel times. Simultaneously, it is evident that the effect of external variables is less erratic, indicating that the model successfully captures some of the influence of external variables on the stochastic variables. Consequently, the optimal outcome is achieved in a scenario where there is an effect solely due to external variables, without any accompanying noise.

Table 3.27: The results derived from two positioning approaches on the SVRPTW test dataset.

	10		20		50		100	
Customers Position	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.
Flexible customer positions	7.05	6.84	11.99	11.63	23.76	23.05	56.36	54.66
Fixed customer positions	**6.37**	**6.18**	**10.83**	**10.50**	**21.46**	**20.82**	**50.90**	**49.37**

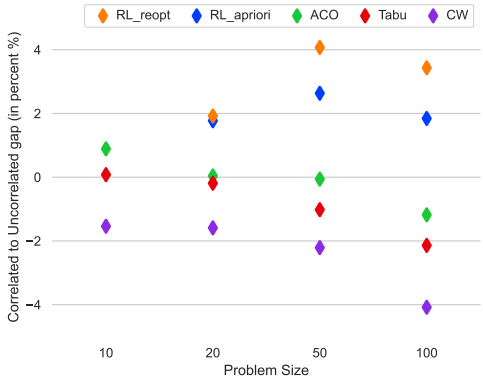


Figure 3.14: Percentage difference in travel times between scenarios with correlated variables (setting $\mathrm{A}, \mathrm{B}, \Gamma = 0.8, 0.2, 0.0$) and uncorrelated variables (setting $\mathrm{A}, \mathrm{B}, \Gamma = 0.8, 0.0, 0.2$).

Table 3.28: The results derived from different fill rates on the SVRPTW test dataset.

	10		20		50		100	
Fill Rate	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.
0.1	7.05	6.84	11.99	11.63	23.76	23.05	56.36	54.66
0.5	6.37	6.18	10.83	10.50	21.46	20.82	50.90	49.37
0.9	**5.97**	**5.79**	**10.15**	**9.85**	**20.12**	**19.52**	**47.72**	**46.29**

In the next experiment, we scrutinized the impact of fixed customer positions versus flexible customer positions. The outcomes of this experiment are depicted in Table 3.27. Evidently, the results favor fixed customer positions due to the less intricate nature of the problem in this scenario. However, the primary objective of this experiment was to assess the model's capacity to learn in both settings. The results affirm that the model can effectively optimize routes in both positioning approaches, with superior performance observed in the case of fixed customer positions.

In the subsequent experiment, we assessed the capacity of both baselines and the RL model to discern correlations among stochastic variables. This correlation was introduced by maintaining identical values of external variables among customers for both demand and travel times. The experiment aimed to examine the performance gap between correlated and uncorrelated settings. The outcomes are illustrated in Figure 3.14, showcasing that CW and Tabu Search methods are unable to leverage the correlation among stochastic variables. The ACO algorithm exhibits improved performance in correlated environments with smaller-sized problems; however, as the problem size increases, its performance diminishes. Notably, the RL model demonstrates a noteworthy ability to discern and utilize correlations among stochastic variables, resulting in enhanced routing optimization.

We assessed the model's performance under varying fill rate conditions, as depicted in Table 3.28. As expected, an increase in the fill rate correlates with improved performance, as larger vehicle capacities allow for the utilization of fewer routes, subsequently reducing the total travel cost. This experiment aimed to investigate the model's adaptability to different fill rate scenarios and ascertain its capability to learn effective policies across various vehicle maximum capacity settings. The results affirm the model's robustness and its capacity to learn optimal routing

Table 3.29: The results obtained from various delivery types on the SVRPTW test dataset.

	10		20		50		100	
Delivery Type	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.
Full delivery only	7.00	6.79	11.89	11.53	23.57	22.86	55.90	54.22
Partial delivery	**6.37**	**6.18**	**10.83**	**10.5**	**21.46**	**20.82**	**50.90**	**49.37**

Table 3.30: The results obtained with different numbers of vehicles on the SVRPTW test dataset.

	10		20		50		100	
Vehicle Number	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.	Apr.	Reopt.
1	6.37	6.18	10.83	10.50	21.46	20.82	50.90	49.37
2	6.23	6.05	10.60	10.28	21.00	20.37	49.81	48.32
3	6.19	6.01	10.53	10.21	20.87	20.24	49.49	48.01
5	**6.17**	**5.99**	**10.49**	**10.18**	**20.79**	**20.17**	**49.31**	**47.83**

strategies within specified fill rate intervals.

Continuing with our exploration, the subsequent experiment delved into examining the model's robustness when confronted with partial versus full delivery types, with results summarized in Table 3.29. In both settings, the model demonstrated its learning capability, exhibiting superior performance in the case of partial delivery.

Multi-Vehicle Scenarios. In all preceding experiments, we employed a single vehicle for simplicity, primarily to assess the base model's robustness across diverse environments. Subsequently, we investigated the model's capacity to learn effective policies in scenarios involving multiple vehicles. The introduction of a larger fleet implies a more expansive action space, offering the potential for enhanced routing strategies. However, due to the exponential growth of the action space, we sought to examine the model's aptitude for learning in such intricate settings. The results, presented in Table 3.30, indicate that the model indeed leverages the fleet of vehicles to enhance performance with each additional vehicle. Nevertheless, the observed improvement diminishes, suggesting a point of diminishing returns. We posit that there exists an optimal number of vehicles beyond which further additions do not yield appreciable performance gains, warranting exploration in future research.

Time Complexity. In a concluding assessment, we examined the inference times of both baseline methods and our proposed model, as detailed in Table 3.31. Evidently, the CW heuristic demonstrates the shortest time required to find a solution. Our model exhibits the next most efficient timing, displaying a linear dependence on the number of customers. This linear scalability suggests that our model is amenable to real-time applications, demonstrating feasibility for scenarios involving hundreds of customers. Future research endeavors could extend this evaluation to encompass larger datasets, potentially involving thousands of customers, thereby assessing the model's practicality for deployment in the logistics domain, especially within significant operations managed by large-scale logistics companies. Conversely, ACO and Tabu Search methods manifest longer inference times, coupled with a scaling pattern that exceeds linear

Table 3.31: Results for the inference time in seconds on the SVRPTW test dataset.

Baseline	10	20	50	100
Clarke-Wright	0.005	0.016	0.058	0.185
Tabu Search	0.964	2.936	8.672	57.12
Ant-Colony Optimization	1.230	4.291	27.64	123.7
RL-greedy	0.061	0.121	0.191	0.394
RL-sampling	0.064	0.125	0.232	0.405
RL-beam search	0.072	0.178	0.293	0.416

dependence on the number of customers

3.3.5 Conclusion

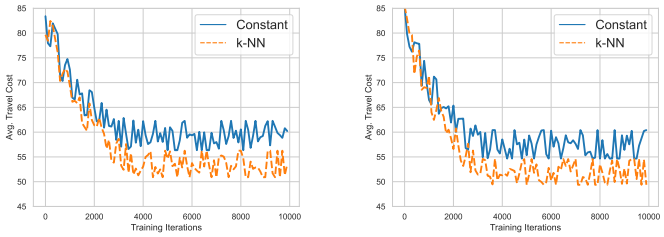
In conclusion, this study introduces a formulation of the stochastic vehicle routing problem incorporating time windows, stochastic demands, and travel times. Bridging a notable gap in the literature, we propose an end-to-end framework to address these SVRPTW challenges. To validate the efficacy and versatility of the model, comprehensive experiments were conducted, evaluating its performance across diverse environmental settings. These experiments encompass variations in inference approaches, delivery types, stochasticity levels, fleet sizes, and customer counts, demonstrating that the model is robust in accommodating a range of SVRPTW configurations.

Additionally, the model design incorporates external variables that closely mirrors real-world scenarios encountered by industry practitioners. Demonstrating a noteworthy 1.73% improvement against the state-of-the-art ACO metaheuristic, our model exhibited enhanced efficiency in minimizing the total travel cost. This outcome underscores the potential for further research avenues exploring the application of RL in the SVRPTW domain. Future research trajectories, both in industry and academia, may delve into more intricate architectures or advanced RL training algorithms aimed at mitigating logistic expenses and alleviating associated environmental impacts. Consequently, our model may serve as a versatile starting point for diverse research in SVRPTW.

3.3.6 Implementation

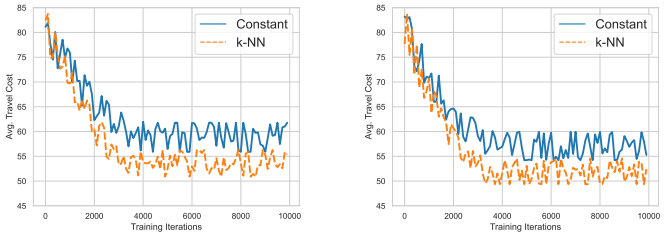
Critic Network. The outputs, denoted as P^{t+1}, derived from this process are subsequently fed into a Critic network. This network comprises two fully connected layers, each with D and one neuron, and employs the Rectified Linear Unit (ReLU) activation function. The dimensionality of the embeddings denoted as D, is defined as 128.

Training Details. We adopt the Xavier initialization method and employ the Adam optimizer with a learning rate set to 10^{-4} during the training phase. To mitigate overfitting, a dropout with a probability of 0.1 is applied. The training process is executed on a GPU system, specifically a NVIDIA A100 SXM 40GB GPU and 2x AMD EPYC 7742 CPUs (8 cores) with 256GB RAM. The training duration consists of a total of 10,000 iterations for each problem size.



A priori Reoptimization

Figure 3.15: The incurred travel cost throughout the training phase, employing two distinct customer position approaches: fixed and flexible.



A priori Reoptimization

Figure 3.16: The travel cost incurred during the training phase, employing two different delivery approaches: full and partial.

3.3.7 Training Curves

Figure 3.15 depicts the training curves, providing a comparative analysis of the performance under distinct customer positioning approaches. Additionally, Figure 3.16 illustrates the training curves, showcasing the performance variations corresponding to different delivery types.

Chapter 4

Large Language Models for Combinatorial Problems

This chapter covers the introduction of a novel prompting strategy called Self-Guiding Exploration (SGE), designed for enhancing the performance of Large Language Models in solving Combinatorial Problems. Combinatorial Problems, such as the Traveling Salesman Problem and Vehicle Routing Problem, are known for their NP-hardness and complexity, making them a challenge for both traditional optimization methods and newer machine learning approaches. This chapter demonstrates how SGE autonomously generates multiple thought trajectories, decomposes these trajectories into actionable subtasks, and refines the solutions to achieve optimal results. The SGE method is shown to significantly outperform existing prompting techniques, such as Chain-of-Thought, Decomposition, and Self-Refinement. With over a 27% improvement in CP optimization performance, SGE emerges as a powerful tool for solving not only combinatorial tasks but also other reasoning challenges, including arithmetic and symbolic reasoning. The chapter also presents experimental evidence validating SGE's effectiveness across a range of combinatorial tasks, such as the Assignment Problem, Knapsack Problem, and Job Scheduling Problem, establishing its adaptability and generalizability across diverse problem domains. This chapter's contribution to utilizing LLMs in solving complex CPs, which traditionally relied on metaheuristics and exact solvers, marks a significant advancement in applying modern AI techniques to real-world optimization challenges.

4.1 Introduction

Large Language Models (LLMs) have emerged as powerful tools capable of executing reasoning tasks across various domains, including arithmetic, commonsense, and symbolic reasoning [16, 26, 153, 188]. These models may leverage prompting techniques such as Exploration-of-Thought [100, 213, 230], Decomposition [93, 248], and Refinement [137] to break down and solve various tasks in a step-by-step manner. Recent research has been directed towards extending these techniques to tackle more sophisticated optimization challenges [226]. Combinatorial problems (CPs) may represent a category of these complex optimization tasks, associated with intricate computational challenges.

Combinatorial Problems are characterized by their NP-hardness and inherent complexity, which result in an exponential growth in the number of potential solutions. This complexity presents substantial challenges in the research [74, 76, 147, 152]. CPs are especially crucial in sectors that require efficient logistics, planning, and scheduling. Currently, the dominant approach in these industries involves metaheuristic methods. These methods combine various simple but fast heuristics to effectively tackle CPs within specific constraints. Nonetheless, the effectiveness of these heuristics can vary significantly depending on the CP task and its associated constraints, necessitating a customized selection of heuristics to achieve optimal performance. In the meantime, research on exploring LLMs to solve CPs reveals substantial gaps. While recent advancements indicate the effectiveness of LLMs in various reasoning tasks [184, 212, 242, 248], their application to CPs has been minimal. Research such as that conducted in [128, 143, 226] suggests that current generative models can address smaller instances of the Traveling Salesman Problem (TSP). However, as problem sizes increase, existing prompting strategies begin to yield inadequate responses, underscoring the need for more sophisticated prompting methods. Moreover, there is a notable scarcity of research addressing other complex CPs, particularly the Vehicle Routing and Job Scheduling Problems, which pose significant challenges in logistics, planning industries, and operations research.

In this work, we introduce a novel prompting strategy: self-guiding exploration (SGE), designed to enhance the problem-solving process for CPs. This algorithm works as a combination of exploration-of-thought, decomposition, and refinement prompting methods. The SGE approach autonomously generates multiple thought trajectories for a given CP, each trajectory representing a specific heuristic to tackle the given task. Each trajectory is then decomposed into subtasks, which are executed one by one, and their outputs are refined and combined into a final solution. Unlike the task-specific prompts utilized in other methods, SGE employs general-purpose prompts, allowing for the adaptive use of specific heuristic solutions tailored to various CPs, such as the Hungarian heuristic for the assignment problem and the Nearest Neighbor heuristic for the vehicle routing problem. Essentially, SGE acts as a versatile metaheuristic capable of identifying, combining, and refining task-specific heuristics for individual CP tasks.

Our work makes following contributions. Firstly, we present a novel investigation into the application of large language models for solving combinatorial problems. Secondly, we introduce a new prompting strategy, SGE, that autonomously generates thought trajectories, splits them into subtasks and refines the answers. Thirdly, we demonstrate that SGE outperforms existing prompting strategies such as Chain-of-Thought, Decomposition, and Self-Refinement, improving CP optimization performance by 27.84%. Lastly, we validate the applicability of SGE across other reasoning tasks, including arithmetic, commonsense, and symbolic tasks, where our method achieves a 2.46% higher accuracy than the best existing results.

4.2 Related Work

CP via Classical Approach. In classical research, combinatorial problems are predominantly tackled using heuristic and metaheuristic methods specifically crafted for particular tasks. Notable examples include the Ant-Colony Optimization and Tabu Search methods for addressing the Vehicle Routing Problem [52, 112, 161], the Shortest Processing Time and Most Work Remaining

Heuristics for Job Scheduling Problems [169], and the Hungarian Algorithm for the Assignment Problem [1]. These approaches are favored in industrial settings due to their simplicity and speed. However, they need to be individually tailored to each task and its constraints' setting. In contrast, exact solvers such as Google-OR-Tools [57] offer general and precise solutions, but their applicability is often limited to smaller-scale problems due to the inherent NP-hardness of combinatorial problems.

CP via Learning-Based Approach. In AI literature, Reinforcement Learning (RL) has been a prominent approach for tackling combinatorial problems since the 1990s [5, 43, 138, 139, 237]. The integration of deep learning, particularly through innovations like Pointer Networks, has significantly enhanced RL's capability to handle more complex combinatorial tasks [7, 198]. Further advancements involve the use of Transformer networks [36, 102, 195], with notable applications in solving the Vehicle Routing Problem [76, 147]. Despite these advancements, RL-based methods continue to exhibit a notable performance gap when compared to traditional solvers [142, 179, 203, 234].

CP with LLMs. To date, [128, 143, 226] are the only studies investigating the use of LLMs (GPT-3.5 and GPT-4) models on the Traveling Salesman Problem through iterative prompting, requesting the model to refine a candidate solution repeatedly. There remains a significant research gap in applying more advanced prompting strategies and exploring LLMs for other combinatorial problems such as vehicle routing and job scheduling.

Prompting Strategies. The expressive capabilities of direct prompting in Large Language Models are theoretically limited to the complexity class TC^0 [144]. To effectively address combinatorial problems with LLMs, sophisticated prompting strategies are required. One basic approach is the Chain-of-Thought (CoT) prompting, introduced in [214], which encourages LLMs to articulate intermediate "thoughts" that inform the generation of the final output. This technique has given rise to advanced variations, including Self-consistency with CoT (CoT-SC), Tree-of-Thoughts (ToT), and Graph-of-Thought methods [207, 230, 239]. Additionally, decomposition prompting strategies can be employed [93, 247], since they simplify complex tasks into smaller, manageable subtasks via symbolic programs or structured algorithms, thus improving the performance of LLMs. In our experiments, we found these techniques to be insufficient, leading us to propose the Self-Guiding Exploration method as a more effective solution for tackling combinatorial problem tasks.

4.3 Preliminaries

We provide an overview of combinatorial problems, highlighting their inherent complexity with the classic example of the Traveling Salesman Problem (TSP) and an example of a combinatorial problem formulation in a prompt for use by a Large Language Model (LLM).

Combinatorial Problems. Combinatorial problems involve decision-making processes where the goal is to assign binary decision variables $x \in \{0, 1\}$ in order to optimize a cost function $g(x_1, ..., x_n)$, subject to task-specific constraints. A classic example of such a problem is the TSP. In the TSP, given a list of n cities and the distances d_{ij} between cities i, j, the objective is to determine the shortest possible route that visits each city exactly once and returns to the starting

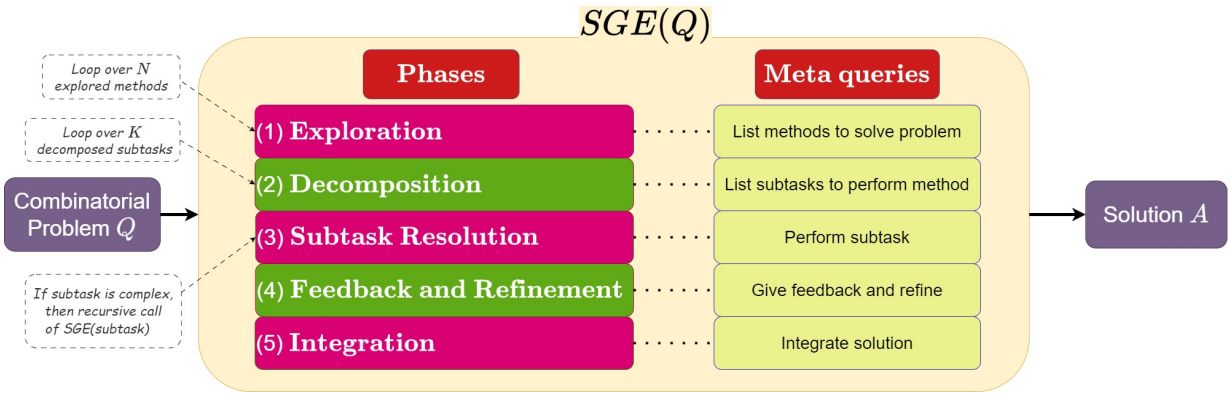


Figure 4.1: **Self-Guiding Exploration**. The generative model autonomously addresses a combinatorial problem task Q through a five-phase process: (1) Exploration of N solution trajectories, where each trajectory offers potential solutions; (2) Decomposition of these trajectories into K subtasks, outlining specific steps for each method; (3) Resolution of each subtask, executing the outlined steps; (4) Feedback and Refinement, where feedback is gathered and used to refine each subtask; (5) Integration of all trajectories into a consolidated final solution A. Distinct from traditional exploration/decomposition techniques, SGE(Q) functions entirely autonomously, eliminating the reliance on task-specific queries or manually created thought exemplars. This independence makes it universally applicable to all CP tasks without necessitating modifications.

city. x_{ij} is used as the action variable, indicating whether the route progresses from city i to city j. The cost function to minimize in TSP is $g(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$, under the condition that all cities visited exactly once $\sum_{i=1}^{n} x_{ij} = 1$ and $\sum_{j=1}^{n} x_{ij} = 1$ for all i, j. Combinatorial problems are generally categorized as NP-hard due to their inherent computational complexity. For instance, a TSP with n cities presents $(n-1)!$ possible routes, rendering the evaluation of all potential solutions impractical and exceedingly time-consuming as n increases.

Prompting Combinatorial Problems in LLMs. To use LLM for solving CP tasks, we define f as the interface function of a generative LLM model, which accepts high-dimensional discrete input tokens and generates outputs within the same token space ($f : W \mapsto W$). For each combinatorial problem task, the input Q to the LLM can be explicitly defined in a textual format. This description delineates the specific goal alongside a list of variables tailored to the task at hand. For instance, the objective of the Traveling Salesman Problem (TSP) can be textually articulated as "Find a route that minimizes the total travel distance, visits each city exactly once, and starts and ends in the same city." Subsequently, the variables, such as the distances between cities d_{ij}, are provided in a format such as "The distance between city i and city j is [number]", laying out all necessary parameters for the model to process and generate solutions. The model will then process this structured input Q to produce the corresponding solution answer A, where both Q and A are within token space W; formally, $A = f(Q)$. Given the inherent complexity of combinatorial problems, direct zero-shot prompting $f(Q)$ is insufficient. Consequently, we propose a self-guiding exploration algorithm that employs metaheuristic-like strategies to effectively solve CP tasks.

4.4 Method

In this section, we introduce Self-Guiding Exploration (SGE) method and provide a detailed explanation of its algorithm designed to tackle combinatorial problems. The method (Fig 4.1), inspired by metaheuristic approaches, synthesizes multiple heuristic methods. It generates various thought trajectories, with each trajectory representing a specific heuristic approach. These trajectories are then integrated to form the final solution. To overcome the challenges of executing complex heuristics through LLMs in one step, our algorithm utilizes a decomposition strategy. This approach breaks down each trajectory into smaller, more manageable subtasks, enabling the solution to progress through sequential, simpler steps. This general-purpose algorithm is tailored to adapt to a wide range of combinatorial problems without the constraints of task-specific exemplars for few-shot solution generation.

4.4.1 Algorithm

The proposed method's algorithm is segmented into five distinct phases, as outlined in Figure 4.1. These phases include exploring thought trajectories, decomposing each trajectory into subtasks, resolving each subtask to generate thoughts, obtaining feedback and refining the thoughts, and finally integrating all thoughts to formulate the answer. Each thought here represents one completed subtask.

Exploration. During the exploration stage, the model tackles the overarching problem Q by engaging with exploration meta-prompt. This prompt $Z_{explore}$ is structured as: "List all possible methods to solve this problem. Return them separated by new lines." This prompt stimulates the model to enumerate potential methodologies pertinent to Q. The sequence is then divided into task-specific trajectories of queries Q^n each incorporating a method to address Q:

$$Q^{\mathbb{N}} = f(Q, Z_{explore}).$$

Decomposition. Following exploration, each trajectory query Q^n is processed through the model to break down the trajectory into actionable steps. The decomposition meta-prompt Z_{decomp} is formulated as: "List all steps to use the method. Return them separated by new lines." This leads to the generation of subtask queries that operationalize the trajectory method:

$$Q^n_{\mathbb{K}} = f(Q, Q^n, Z_{decomp}).$$

Subtask Resolution. Post-decomposition, the subtask queries $Q^n_{\mathbb{K}}$ are each split into K individual queries and processed by the model to generate thoughts. The model initially evaluates if the task is easily solvable using the meta-prompt Z_{check}: "Is this problem easily solvable? Return yes or no": $f(Q^n_k, Z_{check})$. If the response is affirmative, the model executes the subtask query Q^n_k to generate a new thought:

$$T^n_k = f(Q, T^n_{k-1}, Q^n_k).$$

Otherwise, the model engages a recursive instance of the self-guiding exploration algorithm on Q^n_k instead of the main task Q to navigate and decompose the complex subtask, producing:

$$T^n_k = f(Q, T^n_k, Q^n_{k_{feedback}}).$$

Feedback and Refinement. In this stage, the model utilizes an additional meta-prompt $Z_{feedback}$ - "Give feedback to the proposed solution" - to generate feedback queries

$$Q^n_{k_{feedback}} = f(Q, Q^n_k, T^n_k, Z_{feedback})$$

. This guides the model in refining the initial responses through reevaluation and enhancement of the thoughts:

$$T^n_k = f(Q, T^n_k, Q^n_{k_{feedback}}).$$

Integration. Upon completion of all trajectories and their associated subtasks, the model employs a final meta-prompt $Z_{integrate}$ - "Integrate all previous findings and provide the final answer" - to amalgamate the last thoughts into a definitive solution answer:

$$A = f(Q, T^1_K, ...T^N_K, Z_{integrate}).$$

SGE draws inspiration from metaheuristic methods used to solve combinatorial problem tasks. Yet, due to its general-purpose nature and meta-prompts, it is also suitable for other tasks, beyond CPs. Essentially, it integrates elements of exploration-of-thought, decomposition, and refinement prompting strategies, but it does so without relying on task-specific prompts or solution exemplars.

4.4.2 Baseline Prompting Techniques

Combinatorial problems are too complex for solving using direct approach. To solve in several shots, different methods can be used such as few-shot prompting, chain-of-thought, exploration-of-thought, decomposition, and self-refine advanced prompting techniques. Traditional few-shot prompting involves teaching an LLM to derive an answer A to a query Q using a limited set of contextual examples $D = \{E_1, ..., E_{|D|}\}$, where $A = f(Q, D)$. In the simplest few-shot setup, examples are formatted as $E_j = (Q_j, A_j)$, where each Q_j, A_j are corresponding prompt and solution of example problem j. For Chain of Thought (CoT) prompting, the objective shifts to generating a sequence of intermediate reasoning steps, or "thoughts" T, and subsequently deriving the final answer from T. These in-context examples are structured as $E_j = (Q_j, (T_{j,1}, \ldots, T_{j,k}), A_j)$. Exploration-of-Thought techniques, such as those described in [207, 230], focus on dividing the problem Q into K subproblems and auto-generating thoughts T_k through subproblem queries Q_k with $T_k = f(Q_k, T_{k-1})$.The ultimate answer is then computed as $A = f(Q, T_K)$. In addition to that, in Tree-of-Thought (ToT) and Graph-of-Thought (GoT) prompting strategies, the problem is split into N thought trajectories T^n_k, using pre-determined subqueries Q^n_k, with the final answer determined by $A = f(Q, T^1_K, ..., T^N_K)$. These strategies necessitate manually identifying effective subqueries Q^n_k for each specific task (e.g., arithmetic, commonsense, symbolic). Decomposition prompting strategies, as referenced in [93, 247] get Q^n_k subqueries through in-context examples formatted similarly to CoT exemplars: $E_j = \big((Q_j, \big(Q_{j,1}, T_{j,1}), ..., (Q_{j,k_j}, T_{j,k_j})\big), A_j\big)$. Another prompting method called self-refinement [137] uses feedback and refine procedures to generate feedback queries $Q^n_{k_{feedback}}$ and subsequently refine thoughts T^n_k. In our research we combined these advanced techniques and updated them into new method called self-guiding exploration to enhance performance of LLMs for CPs.

4.4.3 Combinatorial Problems

Combinatorial problems are decision problems where solver needs to assign binary decision variable $x \in \{0, 1\}$ to minimize some cost function or maximize reward function given input C.

Assignment Problem. Despite not being classified as NP-hard and solvable in polynomial time using the Hungarian algorithm, the Assignment Problem remains a fundamental combinatorial challenge. This problem entails optimally assigning n tasks to n workers, aiming to minimize the total cost or maximize the total efficiency of the assignments. The input array C is represented as $n \times n$ cost matrix, where the element at the i^{th} row and j^{th} column represents the cost of assigning the j^{th} task to the i^{th} worker. The goal is essentially about finding a one-to-one matching between workers and tasks with the objective of minimizing the total cost, i.e.

$$\min \ g(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}, \tag{4.1}$$

$$\text{s.t.} \sum_{i=1}^{n} x_{ij} = 1; \quad \sum_{j=1}^{n} x_{ij} = 1; \quad x_{ij} \in \{0, 1\} \quad \forall i, j. \tag{4.2}$$

Knapsack Problem. The knapsack problem is a classic combinatorial problem, focusing on resource allocation. It is a decision problem in which the goal is to pack items from a set of items with given weights w and values v into a container with a maximum capacity W. The input array C is represented as $n \times 2$ matrix with volume and value information of every item i. The goal is to maximize total value of packed items without exceeding container capacity, i.e.

$$\max \ g(x) = \sum_{i=1}^{n} v_i x_i, \tag{4.3}$$

$$\text{s.t.} \sum_{i=1}^{n} w_i x_i \leq W; \quad x_i \in \{0, 1\} \quad \forall i. \tag{4.4}$$

Bin Packing Problem. The bin packing problem is a combinatorial problem that involves efficiently packing n objects of different sizes w into a finite number of k bin containers of fixed capacity W in a way that minimizes the number of bins used. The input array C is represented as $n \times 1$ vector with size information of every item i,

$$\min \ g(x) = \sum_{j=1}^{k} x_j, \tag{4.5}$$

$$\text{s.t.} \sum_{j=1}^{k} x_{ij} = 1; \quad \sum_{i=1}^{n} w_i x_{ij} \leq W; \quad x_{ij} \in \{0, 1\} \quad \forall i, j. \tag{4.6}$$

Travelling Salesman Problem. In the travelling salesman problem, given a list of n cities and the distances d between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city. The input array C is represented as $n \times n$ cost matrix, where the element at the i^{th} row and j^{th} column represents the cost d_{ij} of travelling between these two cities,

$$\min \ g(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}, \tag{4.7}$$

$$\text{s.t.} \sum_{i=1}^{n} x_{ij} = 1; \quad \sum_{j=1}^{n} x_{ij} = 1; \quad x_{ij} \in \{0, 1\} \quad \forall i, j. \tag{4.8}$$

Vehicle Routing Problem (VRP). The Vehicle Routing Problem is an extension of the TSP where a fleet of vehicles must deliver goods to a set of customers from a central depot. The objective is to determine the optimal set of routes for the vehicles such that the total travel cost is minimized, while satisfying constraints such as vehicle capacity and customer demands [54]. Let $G = (V, E)$ be a complete graph where $V = \{v_0, v_1, \ldots, v_n\}$ represents the set of nodes (with v_0 being the depot and $v_1, \ldots, v_n$ being the customers), and $E = \{e_{ij}\}$ represents the edges between nodes. Each edge e_{ij} has a travel cost c_{ij}. Let K represent the set of vehicles, each with a capacity Q_k. The problem is formulated as:

$$\min \ g(x) = \sum_{k \in K} \sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} x_{ij}^{k}, \tag{4.9}$$

$$\text{s.t.} \quad \sum_{k \in K} \sum_{j=1}^{n} x_{ij}^{k} = 1, \quad \forall i \in V \setminus \{v_0\}, \tag{4.10}$$

$$\sum_{j=0}^{n} x_{ij}^{k} = \sum_{j=0}^{n} x_{ji}^{k}, \quad \forall i \in V, \forall k \in K, \tag{4.11}$$

$$\sum_{i=1}^{n} d_i \sum_{j=1}^{n} x_{ij}^{k} \leq Q_k, \quad \forall k \in K, \tag{4.12}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij}^{k} \leq |S| - 1, \quad \forall S \subseteq V \setminus \{v_0\}, \forall k \in K, \tag{4.13}$$

$$x_{ij}^{k} \in \{0, 1\}, \quad \forall i, j, k. \tag{4.14}$$

Here, x_{ij}^{k} is a binary decision variable indicating whether vehicle k travels directly from node i to node j. The first constraint ensures that each customer is visited exactly once by one vehicle. The second constraint enforces flow conservation, ensuring that if a vehicle arrives at a node, it must also leave. The third constraint ensures that the total demand served by a vehicle does not exceed its capacity Q_k. The fourth constraint eliminates sub-tours by ensuring that no subset of customers forms a cycle disconnected from the depot. The VRP is widely applied in logistics to optimize delivery routes, with applications in industries like courier services and waste collection,

where efficient routing reduces costs and environmental impact. Exact solutions are feasible for small instances using mixed-integer linear programming, while larger instances are tackled using heuristics and metaheuristics such as the Clarke-Wright savings algorithm, genetic algorithms, and ant colony optimization. Recent advances leverage reinforcement learning to develop adaptive routing strategies.

Job Scheduling Problem. The job scheduling problem focuses on scheduling n jobs on m machines, where each job i consists of a sequence of m operations that need to be processed in a specified order. Each operation requires a specific machine for a certain period of time, and each machine can handle only one operation at a time. The input array C is represented as $n \times m \times 2$ cost matrix, which stores machine id and completion time for every operation j of every job i. The primary objective is to minimize the makespan, which is the total time required to complete all jobs, effectively reducing the time from the start of the first operation to the completion of the last operation across all jobs.

Challenges in Solving Combinatorial Problems. The difficulty of the combinatorial problems like TSP, VRP, JSP, and Knapsack problem stems from several intrinsic and mathematical characteristics common among them. These problems are typically classified as NP-hard, which fundamentally contributes to their computational complexity. As the number of elements (cities, jobs, items, etc.) increases, the number of possible combinations or permutations explodes exponentially. For instance, the TSP with n cities has $(n-1)!$ possible routes to evaluate. This exponential growth means that the time required to examine all possible solutions becomes impractically long even for relatively small n. In addition to that combinatorial problems involve decisions that are interdependent, where the choice for one element affects the options and costs for others. For example, in the JSP, the order in which jobs are processed on one machine can affect the scheduling for other machines.

Applicability of SGE to Combinatorial Problems. For combinatorial problems, obtaining exact solutions is generally computationally prohibitive. Consequently, approximation algorithms and heuristic methods are frequently employed to tackle these challenges. Typically, these problems are broken down into more manageable subproblems. Initial solutions are then generated using heuristic functions, which are subsequently refined through additional heuristic techniques to enhance solution quality. The SGE approach is particularly well-suited for combinatorial problems as it embodies this multi-stage process: it systematically decomposes the problem, explores potential solving methods, and iteratively refines the solutions.

4.5 Experiments

This section details the experimental setup and presents the results of our proposed method applied to combinatorial problem tasks, as well as its performance on other reasoning tasks commonly explored in LLM research. Our implementation is available online.[1]

[1] `https://github.com/Zangir/LLM-for-CP`

4.5.1 Setup

CP tasks. The experiments were conducted on six combinatorial tasks: Assignment Problem, Knapsack Problem, Bin Packing Problem, Traveling Salesman Problem, Vehicle Routing Problem, and Job Scheduling Problem. The Assignment Problem, classified as P-complete, can be optimally solved using the Hungarian Algorithm. In contrast, the other tasks are NP-hard and ordered by increasing complexity. For a more detailed discussion of these CP tasks, refer to Section 4.4.3. We included five distinct problem sizes, involving 5, 10, 15, 20, and 30 elements (nodes) such as cities in the TSP/VRP. To facilitate these experiments, a dataset was created, comprising 100 randomly generated instances for each problem size. These instances were characterized by uniformly distributed variables, such as the positioning of cities in TSP/VRP or bin volume in the Bin Packing Problem, over an interval from 0 to 100. The experiments utilized an NVIDIA A100 SXM 40GB GPU, paired with two AMD EPYC 7742 CPUs (8 cores each) and 256GB RAM.

Baselines. We utilized four baseline prompting methods: Input-Output (IO) Direct Prompting, Chain-of-Thought Prompting, Self-Refine (Refine) Prompting, and Decomposition Prompting. The Input-Output (IO) approach involves a single prompt where the model is asked to provide a solution directly, without complex prompting. In this approach, we generate N sample candidates by repeatedly prompting the model with the same query Q, N times. The responses are then aggregated through majority voting to identify the most common solution among the N outputs. We employ the Self-Refine (Refine) method [137], which includes a feedback-refinement procedure that aligns closely with phase four of SGE. Additionally, we use the zero-shot Chain-of-Thought method [100], which is the basic technique among Exploration-of-Thought methods. Lastly, we implemented the Decomposition method as described in [248]. In our experiments, these baseline methods were tested across a range of five LLM models including GPT-4, GPT-3.5 by OpenAI, Gemini-1.5 by Google, and the Llama-2 series from Meta, which includes models with 70 billion and 7 billion parameters. We did not include prompting methods previously used in [128, 143, 226], as their prompting strategies showed inferior results compared to the zero-shot Chain-of-Thought approach when tested with our data.

Metrics. In our study, each method's performance is evaluated relative to IO (Input-Output) prompting. To quantify the improvement, we first measure the solution cost g_{io} for each combinatorial problem task using IO prompting (e.g., for the TSP, $g_{io} = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$). We then calculate the cost g_{method} using alternative methods. The percentage improvement is computed as $100 \times \frac{g_{io} - g_{method}}{g_{io}}$. For problems of smaller sizes, we are able to obtain optimal solutions using the Google-OR-Tools solver through a brute force approach. In such instances, we measure the cost of the optimal solution g_{opt} and determine the optimality gap as $100 \times \frac{g_{method} - g_{opt}}{g_{opt}}$.

4.5.2 Results on CP Tasks

To evaluate the general performance of SGE on combinatorial problems, we conducted experiments comparing performance improvement to IO of SGE and CoT, Decomposition, and Refinement baselines using GPT-4 and Gemini-1.5 LLM models. Table 4.1 gives the results on six combinatorial problem tasks. The results analysis shows that the SGE method consistently outperforms CoT, Refine, and Decomposition methods across all tasks. Notably, the magnitude

	GPT-4				Gemini-1.5			
Task	CoT	Refine	Decomp	Ours	CoT	Refine	Decomp	Ours
Assignment	11.46	14.47	33.80	**41.33**	11.66	13.98	31.94	**40.46**
Knapsack	15.37	17.16	51.95	**70.39**	13.85	16.85	48.62	**65.87**
Bin Packing	14.06	17.12	39.57	**74.72**	11.89	15.43	35.74	**67.63**
Travelling Salesman	13.64	15.75	38.49	**72.10**	14.34	15.90	36.36	**68.09**
Vehicle Routing	14.27	16.94	36.73	**71.92**	11.88	15.13	33.59	**68.02**
Job Scheduling	13.84	16.37	38.20	**75.33**	13.41	15.75	36.36	**67.89**

Table 4.1: **Percentage performance improvement compared to IO** on CP tasks using GPT-4 and Gemini-1.5 models. CoT uses majority voting, with the number of candidates equal to the number of thoughts produced by SGE. The metrics is quantified as percentage improvement in cost with respect to IO solution (the bigger it is the better).

of improvement escalates with the increasing complexity of the problems, from polynomial to exponential. The margin with the second-best method, Decomposition, ranges from 7.53% for the Assignment Problem to 37.13% for the JSP. This trend suggests that the IO method may struggle with the computational demands of NP-hard problems, where more sophisticated strategies like SGE provide significant advantages.

To qualitatively evaluate the performance of the Exploration, Decomposition, and Refinement phases of SGE method, we assessed the LLM outputs for each phase across five random instances of each combinatorial problem task. Figure 4.2 illustrates an example of how our method addresses the TSP, showcasing outputs during each of the three phases of SGE. In the Exploration phase, the first box of Figure 4.2 displays how LLM f generates a list of potential algorithms suitable for solving the TSP, such as heuristic approaches like Nearest Neighbor, metaheuristic techniques like Ant Colony, and Mixed Integer Linear Programming (MILP) method. This phase adapts to different combinatorial problems by suggesting tailored algorithms, like the Hungarian algorithm for the Assignment problem, Greedy algorithms for the Knapsack problem, and Clustering methods for the Vehicle Routing Problem (VRP). Each list of algorithms forms the foundation for generating diverse candidate solutions tailored to each specific problem. The Decomposition phase, depicted in the second box of Figure 4.2, breaks down each identified algorithm into specific subtasks. This example shows the decomposition of the Nearest Neighbor algorithm for the TSP, where the initial subtasks are simple enough for direct processing by model f. However, more complex tasks, such as loops, undergo further decomposition using SGE in a recursive manner, with computational or programming tasks being handled using Python within models like GPT-4 and Gemini-1.5 equipped with a Code Interpreter. Finally, the Refinement phase, illustrated in the third box of Figure 4.2, focuses on enhancing the candidate solutions developed in the previous stage. This example pertains to refining a solution derived from the Nearest Neighbor algorithm for the TSP by implementing the 2-opt algorithm. Renowned for its effectiveness in TSP and VRP contexts, the 2-opt algorithm optimizes the initial solution to find locally optimal solutions within a specific neighborhood, thus improving the overall quality of the candidate solutions. This example shows that SGE method adapts its approach to suit different combinatorial problems, finding a special set of heuristics for each task.

Example of Exploration of TSP

- **Prompt**: You are given a list of points with coordinates: (1): (0, 49); (2): (26, 41); (3): (2, 29); (4): (49, 15); (5): (0, 23); We need to find the route that has a minimum total length. List all possible methods to solve this problem. [Return conditions].
- **Exploration Output**:
  - Nearest Neighbor Algorithm
  - Christofides' Algorithm
  - Ant Colony Optimization
  - Integer Linear Programming

Example of Decomposition of TSP

- **Prompt**: [Problem definition]. Nearest Neighbor Algorithm. List all steps to use the method. [Return conditions].
- **Decomposition Output**:
  - Select a Starting Point
  - Find the Nearest Neighbor
  - Update the Tour
  - Repeat
  - Return to the Starting Point

Example of Refinement of TSP

- **Prompt**: [Problem definition]. [Nearest Neighbor candidate solution]. Give feedback to the proposed solution.
- **Refinement Output**:
  - Choose two edges
  - Remove these edges and reconnect the two paths that were created in such a way that the new connections do not cross over each other.
  - Check if the new tour has a shorter distance than the previous tour. If it does, keep the change. Otherwise, revert to the previous tour.
  - Repeat this process until no improvements can be made.

Figure 4.2: **Example of SGE inference** across the Exploration, Decomposition, and Refinement phases for the Traveling Salesman Problem. The figure displays three boxes, each illustrating the prompt structure and corresponding example output for each phase.

Size	Method	Assignment	Knapsack	Bin Packing	TSP	VRP	JSP
5 NODES	IO	45.45	90.10	108.2	100.3	102.0	105.3
	CoT	39.33	66.88	78.24	81.15	78.17	79.41
	Refine	36.42	61.98	77.40	71.62	72.49	71.72
	Decomp	14.66	21.56	40.00	43.62	40.65	44.15
	Ours	**2.500**	**8.050**	**9.060**	**8.27**	**11.92**	**9.300**
8 NODES	IO	46.84	103.5	112.8	116.9	116.3	108.2
	CoT	39.70	73.84	85.08	89.01	89.48	85.21
	Refine	37.32	72.62	86.25	85.59	83.31	78.43
	Decomp	18.49	26.43	52.73	53.48	54.43	49.81
	Ours	**8.290**	**14.88**	**20.95**	**15.19**	**19.65**	**21.26**
12 NODES	IO	49.11	101.5	120.7	121.6	118.5	117.6
	CoT	41.70	79.33	93.84	86.84	90.05	89.29
	Refine	40.35	77.09	82.23	88.57	88.40	87.02
	Decomp	21.12	35.82	55.40	57.51	59.19	56.01
	Ours	**11.26**	**16.82**	**22.38**	**16.12**	**24.00**	**22.86**

Table 4.2: **Optimality gap** of prompting methods using `gpt-4` w/ code interpreter. The results are represented as performance percentage difference compared to optimal solutions (the smaller it is, the better).

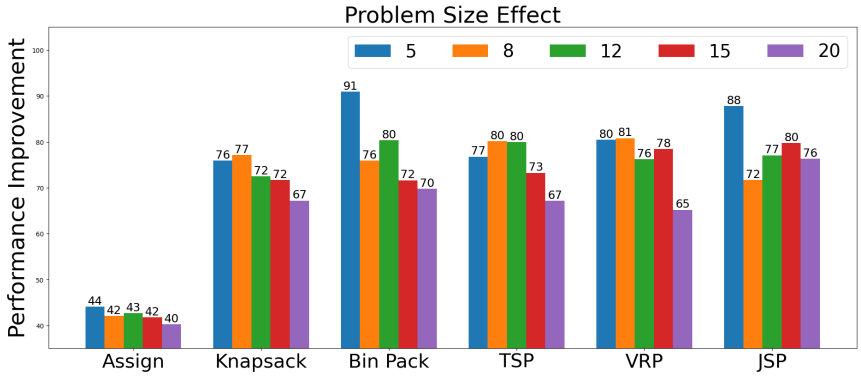


Figure 4.3: **Effect of Problem Size on Performance Improvement** relative to the IO solution using `gpt-4` w/ code interpreter. The analysis spans problem instances of varying sizes, systematically presented from the smallest to the largest, specifically ranging from $n = 5$ to $n = 20$ nodes. Results are organized to highlight the impact of increasing problem complexity on the effectiveness of the solution.

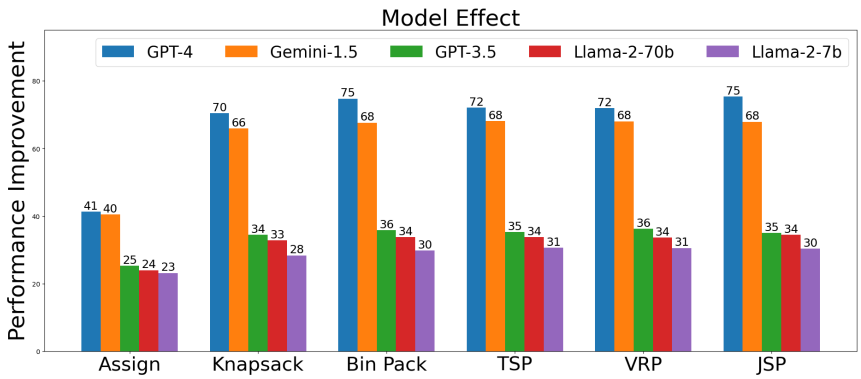


Figure 4.4: **Effect of Model Choice on Performance Improvement** relative to the IO solution.

Effect of Problem Size on SGE Performance. To evaluate the effect of problem size on SGE performance, we conducted experiments on all six tasks with input sizes of 5, 8, 12, 15, and 20 nodes using the GPT-4 model. Figure 4.3 gives the results of these experiments. The results analysis shows that generally, an increase in problem complexity, as determined by the size of the problem input, negatively influences performance improvement; larger problem sizes result in diminished performance improvement of the SGE method compared to IO. Specifically, tasks with 20 input nodes consistently exhibit lower performance improvements relative to the IO method than tasks with 5 input nodes. However, when comparing less disparate sizes, such as 8 and 12 nodes, the differential impact on performance is less pronounced and can occasionally be positive.

Gap Between SGE Performance and the Global Optimum. To evaluate the gap between SGE solutions and global optimum solutions, we conducted experiments on small problem sizes involving 5, 8, and 12 nodes, utilizing the brute force method via Google-OR-Tools to determine the global optimum. Table 4.2 provides the results of these experiments in terms of the percentage gap between the performance of SGE and optimal solutions. The results analysis shows that generally, the SGE method exhibits a smaller optimality gap across all tasks when compared to baseline methods. This advantage is particularly pronounced for more complex problems such as Bin Packing, TSP, VRP, and JSP. For instance, SGE achieves a 12.16% smaller optimality gap on the Assignment task than the next best Decomposition method and a 34.85% smaller gap on the JSP.

Effect of LLM Selection on SGE Performance. To evaluate the impact of model selection on SGE performance, we conducted experiments comparing different models, including GPT-4, Gemini-1.5, GPT-3.5, Llama-2-70b, and Llama-2-7b models. Figure 4.4 provides the results of these experiments. The results analysis shows that GPT-4 and Gemini-1.5 demonstrate significantly better performance compared to other models across all tasks. A notable feature of both models is the integration of a Code Interpreter (CI) tool, which appears crucial for combinatorial problem tasks as it enables the models to execute generated code and evaluate solution performance directly. In contrast, models lacking a CI tool, such as GPT-3.5, Llama-2-70b, and Llama-2-7b, exhibit poorer outcomes, with GPT-3.5 slightly outperforming the Llama models. The comparison between Llama models indicates that the size of the model with 70 billion versus 7 billion parameters does not significantly influence performance. This suggests that model size

Method	Arithmetic				Commonsense			Symbolic	Avg.
	AQUA	GSM8K	SVAMP	ASDiv	StrategyQA	CSQA	ARC	LastLetter	
IO Prompting	67.30	87.04	88.34	90.10	78.40	81.14	87.52	81.98	82.73
CoT Prompting	69.57	89.76	91.58	93.32	81.16	84.15	90.80	85.18	85.69
Refine Prompting	69.68	89.80	91.00	93.10	81.26	83.50	91.09	84.82	85.53
Decomp Prompting	69.99	91.85	92.16	94.08	82.08	84.88	91.76	85.64	86.43
Ours	**74.63**	**97.35**	**98.16**	**97.24**	**83.49**	**85.68**	**93.28**	**86.96**	**89.60**

Table 4.3: **Results for reasoning tasks** using `gpt-4` with a code interpreter are presented as accuracies on benchmark test sets.

Method	Performance Improvement	Function Calls
CoT	13.77	58.32
Refine	16.30	58.32
Decomp	39.79	31.04
Ours	67.63	58.32

Table 4.4: **Performance vs Efficiency** of prompting methods using `gpt-4` w/ code interpreter. The results are represented as performance percentage improvement compared to IO solution and number of model f calls.

alone does not guarantee substantial performance improvements in combinatorial tasks.

Trade-off Between Performance and Cost in SGE. To evaluate the cost-effectiveness of different methods, we conducted experiments to compare the average performance improvement per method against the average number of LLM calls utilized to solve each combinatorial problem instance. Table 4.4 gives the results of these experiments, where the number of function calls in CoT and Refine methods was explicitly controlled to make them equal to SGE function calls. The results analysis shows that the SGE method achieves a 27.84% better performance compared to the Decomposition method but requires 87.89% more function calls. Thus, while SGE offers superior performance, it does so at a marginally higher operational cost. Therefore, the application of this method is particularly justified in scenarios where performance gains are prioritized over cost efficiency.

4.5.3 Results on Reasoning Tasks

To evaluate the versatility of SGE in handling different types of tasks, we conducted experiments across eight datasets commonly referenced in LLM research, categorized into three distinct task types: arithmetic, commonsense reasoning, and symbolic reasoning. Table 4.3 gives the results of these experiments, with each dataset comprising train and test splits where SGE and baseline methods were applied to the test splits. The results analysis shows that the SGE method demonstrates incremental but consistently superior performance across all task categories. Notably, the method demonstrates significant strength in arithmetic tasks, achieving an average

improvement of 4.83% over IO prompting, along with gains of 1.24% in commonsense reasoning tasks and 1.32% in symbolic reasoning tasks. This demonstrates the method's applicability and effectiveness across a diverse range of tasks, extending beyond combinatorial problems.

Method	**Arithmetic**				**Commonsense**			**Symbolic**	**Avg.**
	AQUA	GSM8K	SVAMP	ASDiv	StrategyQA	CSQA	ARC	LastLetter	
IO Prompting	44.16	57.05	60.02	65.43	54.70	55.68	64.35	56.21	57.20
CoT Prompting	58.35	75.84	80.30	87.39	72.97	74.36	85.48	74.63	76.17
Refine Prompting	58.81	76.10	80.44	87.35	73.38	74.13	85.71	74.78	76.34
Decomp Prompting	71.64	82.11	84.34	89.83	76.16	78.84	87.48	77.89	81.11
Ours	**72.84**	**86.18**	**86.44**	**90.17**	**77.44**	**79.09**	**87.83**	**78.78**	**82.27**

Table 4.5: **Results for reasoning tasks** using `gpt-3.5` with a code interpreter are presented as accuracies on benchmark test sets.

4.6 Conclusion

This study has explored the application of Large Language Models to Combinatorial Problems, a category of tasks known for their NP-hardness. Our research introduced a 'Self-Guiding Exploration' prompting strategy that effectively utilizes the inherent strengths of LLMs. By generating multiple thought trajectories tailored to various CPs and autonomously decomposing them into manageable subtasks. Our findings confirm that SGE outperforms existing strategies, improving optimization performance by 27.84% and achieving a 2.46% higher accuracy in reasoning tasks. Notably, SGE shows a 34.85% smaller gap with the global optimum on complex tasks like the Job Sheduling Problem compared to baseline methods. These results underline the potential of advanced LLM strategies in complex problem-solving scenarios, suggesting that the right techniques can enhance the utility of LLMs in critical logistics and resource management applications.

Despite the performance improvements demonstrated by the SGE, several limitations have emerged that merit attention. Firstly, SGE performance depends on the choice of language model. Secondly, the operational costs associated with SGE are notably higher; it requires 87.89% more function calls than the Decomposition method. These issues present clear avenues for future research. Enhancing SGE's computational efficiency while maintaining its high performance could broaden its applicability and make it a more practical choice for a wider range of problems.

Size	Method	Assignment	Knapsack	Bin Packing	TSP	VRP	JSP
5 NODES	Avg.	102.23	710.53	13.480	679.26	809.67	3873.98
	IO	119.25	604.83	15.220	784.98	951.89	4499.38
	CoT	111.61	637.73	14.040	701.50	853.91	4032.66
	Refine	96.350	750.91	14.150	711.28	828.42	4048.06
	Decomp	94.350	738.32	12.920	643.25	746.36	3628.41
	Ours	**89.610**	**820.85**	**11.080**	**555.29**	**667.76**	**3161.4**
8 NODES	Avg.	144.99	1053.98	23.080	1261.02	1436.94	8714.13
	IO	166.38	870.54	26.850	1425.48	1634.52	9896.3
	CoT	157.67	937.96	23.910	1341.56	1527.66	9202.0
	Refine	137.22	1118.63	23.500	1293.09	1504.8	9068.51
	Decomp	137.40	1122.79	21.670	1195.11	1327.99	8088.35
	Ours	**126.27**	**1220.0**	**19.450**	**1049.85**	**1189.73**	**7315.47**
12 NODES	Avg.	218.82	1705.26	32.940	1695.44	2149.65	22142.9
	IO	257.33	1462.01	37.420	1914.84	2424.01	24954.7
	CoT	239.36	1553.15	34.380	1758.64	2305.24	23697.5
	Refine	204.69	1798.81	34.020	1775.3	2219.27	22721.6
	Decomp	205.31	1790.05	30.880	1589.09	1992.09	20553.1
	Ours	**187.44**	**1922.26**	**27.970**	**1439.32**	**1807.66**	**18787.4**
15 NODES	Avg.	254.60	2412.26	37.230	2536.35	2704.09	38224.9
	IO	294.53	2010.03	42.810	2855.74	3070.15	42092.4
	CoT	281.95	2208.41	38.840	2743.79	2817.55	40660.3
	Refine	242.95	2537.45	38.710	2625.05	2807.14	40123.6
	Decomp	235.55	2606.89	34.770	2354.43	2584.13	36515.9
	Ours	**218.01**	**2698.54**	**31.040**	**2102.76**	**2241.5**	**31732.2**
20 NODES	Avg.	323.14	2799.89	47.160	3122.83	3317.71	58569.7
	IO	377.67	2433.27	53.060	3490.25	3798.23	66151.8
	CoT	350.43	2523.88	50.330	3342.51	3504.49	62343.8
	Refine	311.98	2867.08	48.550	3202.12	3356.6	61383.3
	Decomp	298.35	2988.4	43.690	2924.58	3120.93	54426.8
	Ours	**277.27**	**3186.8**	**40.180**	**2654.7**	**2808.3**	**48542.6**
25 NODES	Avg.	411.80	3732.26	14.640	3787.11	3991.53	103789.0
	IO	467.70	3208.8	16.740	4246.8	4470.97	117662.0
	CoT	449.08	3310.65	15.670	4023.17	4152.43	111329.0
	Refine	387.31	3854.93	15.150	3943.46	4101.81	105411.0
	Decomp	394.03	3988.58	13.560	3590.42	3832.48	97377.8
	Ours	**360.87**	**4298.35**	**12.060**	**3131.7**	**3399.95**	**87164.7**
30 NODES	Avg.	512.38	4653.31	83.860	4995.58	5725.6	144036.0
	IO	595.35	3957.27	92.660	5605.85	6501.08	162140.0
	CoT	568.30	4162.98	88.780	5308.59	6051.78	147656.0
	Refine	481.16	4796.2	88.280	5108.91	5853.66	152181.0
	Decomp	476.20	5028.73	79.800	4762.89	5449.52	134230.0
	Ours	**440.87**	**5321.36**	**69.790**	**4191.68**	**4771.98**	**123974.0**

Table 4.6: Comparison of various combinatorial problems based on their average cost per problem size, using `gpt-3.5`. The Knapsack problem aims to maximize returns, while other problems focus on minimizing costs.

Size	Method	Assignment	Knapsack	Bin Packing	TSP	VRP	JSP
5 NODES	Avg.	198.69	370.00	7.4800	372.00	442.95	2131.42
	IO	267.83	267.03	9.5800	459.92	555.57	2701.22
	CoT	234.43	300.42	8.2000	415.99	490.05	2360.95
	Refine	174.66	400.32	8.1600	394.11	474.43	2259.75
	Decomp	166.67	412.43	6.4400	329.82	386.85	1896.89
	Ours	**149.84**	**469.81**	**5.0200**	**260.18**	**307.84**	**1438.3**
8 NODES	Avg.	278.35	571.94	13.030	710.49	821.54	4878.37
	IO	365.34	415.19	16.160	890.27	1029.37	6024.32
	CoT	324.71	456.91	14.050	775.93	901.69	5359.79
	Refine	249.70	623.87	14.140	761.91	872.33	5163.49
	Decomp	240.23	628.28	11.600	630.08	734.92	4335.16
	Ours	**211.78**	**735.45**	**9.1800**	**494.25**	**569.41**	**3509.08**
12 NODES	Avg.	405.56	944.25	18.730	962.85	1238.87	12501.5
	IO	534.81	680.40	23.640	1215.49	1537.63	15581.6
	CoT	475.38	791.10	20.760	1024.87	1337.63	13557.7
	Refine	359.52	1031.5	19.520	1034.36	1325.99	13395.1
	Decomp	351.35	1044.55	16.640	864.00	1120.36	11173.7
	Ours	**306.72**	**1173.71**	**13.110**	**675.53**	**872.74**	**8799.38**
15 NODES	Avg.	478.16	1349.74	21.780	1469.81	1555.83	21885.7
	IO	634.70	996.41	27.150	1822.75	1958.65	27565.6
	CoT	561.76	1133.85	23.670	1608.72	1670.3	23247.9
	Refine	415.50	1460.7	22.820	1567.79	1639.87	23778.5
	Decomp	409.32	1446.91	19.420	1297.62	1412.5	19499.5
	Ours	**369.55**	**1710.83**	**15.820**	**1052.15**	**1097.85**	**15336.9**
20 NODES	Avg.	587.33	1581.38	27.780	1822.89	1918.24	34243.4
	IO	754.98	1189.83	34.320	2262.86	2356.68	42589.9
	CoT	669.13	1326.23	30.270	1981.0	2065.07	36955.8
	Refine	536.41	1678.17	29.380	1902.42	1998.58	36585.5
	Decomp	525.00	1723.18	24.700	1614.36	1744.24	30926.3
	Ours	**451.14**	**1989.48**	**20.220**	**1353.84**	**1426.64**	**24159.4**
25 NODES	Avg.	743.93	2148.32	8.6300	2249.6	2381.92	61530.6
	IO	959.89	1622.78	10.430	2690.03	2872.54	75770.1
	CoT	843.97	1827.16	9.3000	2430.33	2563.55	67624.2
	Refine	678.44	2238.86	9.3300	2445.45	2510.81	64490.3
	Decomp	659.79	2341.61	7.8200	2002.71	2177.05	55070.7
	Ours	**577.56**	**2711.18**	**6.2700**	**1679.49**	**1785.68**	**44697.5**
30 NODES	Avg.	912.48	2707.35	50.790	3070.36	3388.03	87633.7
	IO	1166.32	2079.35	62.520	3735.78	4104.37	104798.0
	CoT	1049.36	2254.12	55.550	3296.34	3602.09	95887.8
	Refine	844.18	2932.01	52.290	3278.37	3546.61	93372.0
	Decomp	788.12	2920.33	46.390	2808.58	3146.44	80674.5
	Ours	**714.42**	**3350.95**	**37.180**	**2232.74**	**2540.65**	**63435.8**

Table 4.7: Comparison of various combinatorial problems based on their average cost per problem size, using `gpt-4`. The Knapsack problem aims to maximize returns, while other problems focus on minimizing costs.

Size	Method	Assignment	Knapsack	Bin Packing	TSP	VRP	JSP
5 NODES	Avg.	185.01	395.98	7.8900	392.96	475.25	2247.28
	IO	249.50	293.42	9.7400	488.09	581.55	2781.78
	CoT	212.33	325.58	8.6400	432.11	530.63	2460.15
	Refine	161.90	424.53	8.4800	414.98	500.49	2410.6
	Decomp	161.13	433.07	7.0600	346.36	432.60	2007.1
	Ours	**140.19**	**503.30**	**5.5400**	**283.29**	**330.98**	**1576.76**
8 NODES	Avg.	262.29	596.51	13.650	739.72	853.62	5263.2
	IO	348.38	450.87	17.040	917.04	1055.48	6532.72
	CoT	296.24	489.46	14.860	803.14	938.79	5783.3
	Refine	238.56	637.94	14.520	781.85	903.12	5646.28
	Decomp	230.29	653.73	12.050	672.67	769.21	4636.82
	Ours	**197.98**	**750.57**	**9.7700**	**523.89**	**601.50**	**3716.89**
12 NODES	Avg.	382.15	1005.24	19.690	1012.68	1291.47	13134.0
	IO	490.19	729.67	24.330	1249.43	1583.87	15971.6
	CoT	437.93	848.80	21.730	1080.36	1396.44	14301.9
	Refine	346.04	1068.12	20.700	1077.78	1360.73	14185.8
	Decomp	339.04	1129.44	17.490	927.75	1194.11	11726.4
	Ours	**297.53**	**1250.18**	**14.200**	**728.06**	**922.21**	**9484.31**
15 NODES	Avg.	444.10	1449.26	23.150	1564.27	1653.6	23061.3
	IO	585.87	1090.76	27.950	1940.9	2036.78	28629.8
	CoT	509.47	1216.37	25.330	1658.02	1801.66	25246.4
	Refine	399.71	1541.78	24.940	1687.24	1715.89	23949.6
	Decomp	385.44	1585.02	20.670	1416.65	1510.22	20343.8
	Ours	**340.02**	**1812.35**	**16.830**	**1118.55**	**1203.46**	**17137.1**
20 NODES	Avg.	548.21	1678.75	29.180	1945.87	2009.98	35753.1
	IO	701.27	1280.46	35.360	2396.6	2438.29	44349.1
	CoT	620.99	1428.82	31.960	2111.46	2166.27	38059.4
	Refine	497.49	1810.0	30.120	2025.49	2142.69	37385.7
	Decomp	480.97	1800.23	26.790	1740.08	1806.83	32493.8
	Ours	**440.31**	**2074.24**	**21.690**	**1455.74**	**1495.83**	**26477.4**
25 NODES	Avg.	708.54	2233.73	9.2200	2352.5	2496.48	64955.1
	IO	911.14	1694.31	11.280	2890.44	2978.91	77142.4
	CoT	791.74	1878.46	9.9700	2504.54	2720.89	70181.8
	Refine	642.79	2401.42	9.7100	2478.42	2670.85	69104.9
	Decomp	634.67	2434.54	8.4200	2126.79	2266.56	60013.5
	Ours	**562.34**	**2759.92**	**6.7300**	**1762.3**	**1845.18**	**48333.0**
30 NODES	Avg.	863.39	2855.59	52.430	3200.01	3534.72	91402.1
	IO	1108.6	2189.92	62.270	3807.69	4232.51	111716.0
	CoT	1002.04	2442.99	56.720	3411.03	3757.36	96055.7
	Refine	784.26	2991.77	55.310	3447.36	3764.56	95672.6
	Decomp	751.20	3134.08	47.690	2886.03	3265.4	83939.4
	Ours	**670.85**	**3519.17**	**40.140**	**2447.95**	**2653.76**	**69626.6**

Table 4.8: Comparison of various combinatorial problems based on their average cost per problem size, using `gemini-1.5`. The Knapsack problem aims to maximize returns, while other problems focus on minimizing costs.

Size	Method	Assignment	Knapsack	Bin Packing	TSP	VRP	JSP
5 NODES	Avg.	89.950	802.42	15.550	775.06	926.24	4397.27
	IO	104.24	698.34	17.110	853.58	1041.09	4958.95
	CoT	98.820	734.35	16.520	821.09	979.18	4575.45
	Refine	84.600	845.41	15.890	787.03	939.69	4634.51
	Decomp	82.460	839.18	14.810	745.98	873.03	4156.86
	Ours	**79.630**	**894.84**	**13.410**	**667.62**	**798.20**	**3660.59**
8 NODES	Avg.	126.29	1197.13	26.320	1418.23	1645.42	9991.67
	IO	147.10	1013.08	28.850	1566.87	1832.76	11162.0
	CoT	136.66	1092.28	27.860	1498.77	1709.73	10389.7
	Refine	119.83	1252.94	27.120	1499.56	1710.88	10509.7
	Decomp	118.10	1274.02	25.100	1343.11	1572.56	9305.07
	Ours	**109.75**	**1353.31**	**22.700**	**1182.83**	**1401.16**	**8591.86**
12 NODES	Avg.	191.22	1966.31	37.090	1921.87	2465.52	24978.0
	IO	220.40	1731.58	40.930	2146.2	2697.85	27795.8
	CoT	204.53	1810.85	39.550	1984.84	2596.84	25572.4
	Refine	181.54	2045.48	38.930	1992.89	2594.67	26383.4
	Decomp	178.63	2063.66	34.910	1834.9	2338.43	24007.4
	Ours	**170.98**	**2179.96**	**31.130**	**1650.52**	**2099.78**	**21131.3**
15 NODES	Avg.	228.48	2739.37	42.610	2889.57	3017.02	43004.7
	IO	256.26	2351.63	48.170	3212.82	3331.89	47876.6
	CoT	250.10	2553.18	44.260	3025.74	3113.82	45938.3
	Refine	220.66	2843.52	44.470	3039.47	3077.27	43884.0
	Decomp	216.12	2896.08	40.350	2695.41	2901.24	41025.1
	Ours	**199.24**	**3052.45**	**35.790**	**2474.42**	**2660.9**	**36299.3**
20 NODES	Avg.	287.07	3209.13	52.710	3548.1	3703.92	65481.9
	IO	329.28	2836.21	58.190	3983.11	4240.29	72279.5
	CoT	307.87	2933.81	54.780	3746.41	3837.66	69172.2
	Refine	277.27	3249.57	55.810	3667.07	3743.99	69032.1
	Decomp	268.09	3413.37	50.130	3373.55	3477.77	61580.1
	Ours	**252.84**	**3612.7**	**44.640**	**2970.35**	**3219.91**	**55345.8**
25 NODES	Avg.	368.58	4255.19	16.310	4292.01	4503.47	116702.0
	IO	425.08	3693.28	18.060	4674.8	5083.89	127229.0
	CoT	395.84	3938.03	16.780	4529.96	4718.81	121452.0
	Refine	356.35	4438.0	17.280	4418.16	4667.3	123046.0
	Decomp	344.61	4534.51	15.410	4140.85	4267.0	110197.0
	Ours	**321.02**	**4672.15**	**14.030**	**3696.27**	**3780.34**	**101586.0**
30 NODES	Avg.	452.94	5279.99	93.580	5711.01	6309.75	160913.0
	IO	504.64	4594.52	103.15	6409.08	7164.38	177461.0
	CoT	502.24	4891.09	98.630	5995.11	6626.95	166083.0
	Refine	431.79	5446.81	96.390	5957.74	6419.75	169324.0
	Decomp	426.99	5619.47	89.550	5402.13	5940.75	153129.0
	Ours	**399.05**	**5848.08**	**80.200**	**4791.01**	**5396.94**	**138570.0**

Table 4.9: Comparison of various combinatorial problems based on their average cost per problem size, using `llama-2-7b`. The Knapsack problem aims to maximize returns, while other problems focus on minimizing costs.

Size	Method	Assignment	Knapsack	Bin Packing	TSP	VRP	JSP
5 NODES	Avg.	96.760	757.12	14.650	719.93	864.61	4130.53
	IO	111.85	632.09	16.800	831.07	980.70	4706.85
	CoT	105.51	700.25	15.590	744.12	917.25	4290.67
	Refine	92.620	788.01	15.130	740.45	897.79	4297.34
	Decomp	89.850	815.64	13.340	689.33	807.57	3915.78
	Ours	**83.980**	**849.63**	**12.410**	**594.70**	**719.74**	**3442.0**
8 NODES	Avg.	134.41	1140.45	24.710	1321.35	1538.71	9429.18
	IO	155.54	967.89	28.290	1477.89	1768.47	10645.8
	CoT	145.07	1035.49	25.540	1399.97	1602.0	9988.41
	Refine	128.91	1186.02	26.100	1355.37	1601.33	9835.02
	Decomp	126.39	1209.13	23.090	1263.55	1424.77	8794.53
	Ours	**116.14**	**1303.73**	**20.560**	**1109.98**	**1296.99**	**7882.13**
12 NODES	Avg.	201.99	1834.2	35.110	1805.11	2315.91	23294.2
	IO	231.77	1562.82	40.340	2081.41	2610.96	26535.8
	CoT	220.85	1690.42	36.560	1922.4	2462.96	24542.6
	Refine	190.52	1865.06	36.340	1831.0	2363.75	23820.7
	Decomp	192.52	1937.35	32.570	1666.65	2169.12	21807.0
	Ours	**174.27**	**2115.36**	**29.750**	**1524.11**	**1972.79**	**19764.7**
15 NODES	Avg.	239.40	2575.28	40.140	2728.94	2902.12	40452.0
	IO	271.97	2237.81	45.310	3091.61	3270.84	46491.8
	CoT	255.93	2296.4	41.940	2852.78	3034.48	42731.8
	Refine	234.52	2625.13	41.180	2850.11	2987.35	42280.3
	Decomp	226.70	2761.56	37.990	2533.08	2736.93	37362.8
	Ours	**207.87**	**2955.51**	**34.280**	**2317.12**	**2480.98**	**33393.3**
20 NODES	Avg.	301.13	2972.25	50.240	3334.42	3499.69	61957.2
	IO	341.58	2565.48	56.260	3758.49	4025.37	69824.9
	CoT	330.69	2705.18	51.940	3462.22	3620.96	63469.1
	Refine	282.33	3031.65	52.210	3438.04	3567.26	64022.3
	Decomp	287.94	3116.33	47.640	3167.42	3297.34	58800.7
	Ours	**263.10**	**3442.61**	**43.130**	**2845.92**	**2987.54**	**53669.1**
25 NODES	Avg.	389.69	4024.08	15.510	4029.51	4184.99	110414.0
	IO	443.99	3518.82	17.390	4600.6	4735.73	126686.0
	CoT	418.18	3677.69	15.880	4148.72	4358.13	115617.0
	Refine	373.97	4197.63	16.380	4136.03	4306.45	111541.0
	Decomp	369.43	4170.69	14.700	3822.3	3975.41	104862.0
	Ours	**342.89**	**4555.58**	**13.180**	**3439.89**	**3549.23**	**93361.4**
30 NODES	Avg.	490.92	5026.68	88.930	5310.47	5986.86	154018.0
	IO	562.14	4332.75	99.580	5902.09	6848.47	171767.0
	CoT	535.73	4629.8	92.530	5670.09	6113.72	158276.0
	Refine	472.51	5217.75	92.730	5409.26	6165.51	160831.0
	Decomp	455.30	5339.41	85.010	4948.58	5562.01	146910.0
	Ours	**428.95**	**5613.7**	**74.780**	**4622.34**	**5244.57**	**132308.0**

Table 4.10: Comparison of various combinatorial problems based on their average cost per problem size, using `llama-2-70b`. The Knapsack problem aims to maximize returns, while other problems focus on minimizing costs.

Example of VRP Prompting

- **Task Description**: You are given a list of customers with coordinates:
  - (1): (87, 39);
  - (2): (1, 90);
  - (3): (28, 65);
  - (4): (85, 85);

  and a list of customer demands:

  - (1): 12;
  - (2): 8;
  - (3): 16;
  - (4): 5.

  There is a depot with coordinates (50, 50) and one vehicle with a maximum capacity of 50. The goal is to find the route that has the minimum total length and go through all the customers, starting and ending at the depot.

Example of VRP Exploration

- **Prompt**: [Task Description]. List heuristic methods to solve this problem. Return only method names separated by new lines.
- **Exploration Output**:
  - Nearest Neighbor Algorithm
  - Savings Algorithm
  - Sweep Algorithm
  - Clarke and Wright Algorithm
  - Genetic Algorithm
  - Simulated Annealing
  - Tabu Search
  - Ant Colony Optimization
  - Particle Swarm Optimization

Example of VRP Decomposition

- **Prompt**: [Task Description]. Nearest Neighbor Algorithm. List the steps to use this heuristic method. Return only the steps, separated by new lines.
- **Decomposition Output**:
  - **Initialize at the Depot:** The starting point is the depot located at coordinates (50, 50).
  - **Find the Nearest Customer:** Calculate the Euclidean distance from the depot to each customer, and move to the closest one.
  - **Mark the Customer as Visited:** Once a customer is visited, they should not be revisited.
  - **Repeat Finding and Moving:** Continue finding the nearest unvisited customer from the current location until all customers have been visited.
  - **Return to the Depot:** After visiting the last customer, calculate the distance back to the depot to complete the route.
  - **Calculate the Total Length of the Route:** Sum up all the distances traveled from start to finish.

Example of Subtask Resolution, Step One

- **Prompt**: [Task Description]. [Apply the heuristic's step one].
- **Subtask Resolution Output**:

  Initialize at the Depot
  - Current location: Depot at (50, 50).

Example of Subtask Resolution, Step Two

- **Prompt**: [Task Description]. [Apply the heuristic's step two].
- **Subtask Resolution Output**:

  Find the Nearest Customer

  - Distance to Customer 1 at (87, 39): $\sqrt{(87-50)^2+(39-50)^2}$
  - Distance to Customer 2 at (1, 90): $\sqrt{(1-50)^2+(90-50)^2}$
  - Distance to Customer 3 at (28, 65): $\sqrt{(28-50)^2+(65-50)^2}$
  - Distance to Customer 4 at (85, 85): $\sqrt{(85-50)^2+(85-50)^2}$

  The initial distances from the depot to each customer are as follows:

  - Customer 1: 38.60
  - Customer 2: 63.25
  - Customer 3: 26.63 (nearest)
  - Customer 4: 49.50

Example of Subtask Resolution, Step Three

- **Prompt**: [Task Description]. [Apply the heuristic's step three].
- **Subtask Resolution Output**:

  Mark the Customer as Visited.
  The nearest customer is Customer 3 at coordinates (28, 65). We will visit Customer 3 first.

Example of Subtask Resolution, Step Six

- **Prompt**: [Task Description]. [Apply the heuristic's step six].
- **Subtask Resolution Output**:

  Calculate the Total Length of the Route.
  The complete route, following the Nearest Neighbor Algorithm from the depot, visiting all customers, and returning to the depot, totals a distance of approximately 232.22 units.
  Here's a summary of the route taken:

  - Start at Depot (50, 50)
  - Visit Customer 3 (28, 65)
  - Visit Customer 2 (1, 90)
  - Visit Customer 4 (85, 85)
  - Visit Customer 1 (87, 39)
  - Return to Depot (50, 50)

Example of Feedback Prompting

- **Prompt**: [Task Description]. [Nearest Neighbor candidate solution]. Give feedback to the proposed solution.
- **Feedback Output**:

  Implement route optimization post-processing, such as 2-opt or 3-opt techniques, which iteratively refine the route by removing crossings and reducing travel distance.

Example of Refinement Prompting

- **Prompt**: [Task Description]. [Nearest Neighbor candidate solution]. [Feedback]. Improve the solution given feedback.
- **Refinement Output**:
  - we'll apply a 2-opt optimization to the route. This will involve swapping segments of the route to find a shorter path. Given the route '1 -> 2 -> 3 -> 4', we'll try different swaps and see if we can reduce the total distance further.
  - Let's simulate the 2-opt optimization on this route.
  - After applying the 2-opt optimization technique, the final optimized route has been adjusted to: Customer 1 -> Customer 3 -> Customer 2 -> Customer 4, then returning to the depot. This rearrangement was made to minimize the total distance traveled.

Example of Solution Integration

- **Prompt**: [Task Description]. [All candidate solutions]. Integrate all previous findings and provide the final answer. Return the answer only as a Python list of elements.
- **Refinement Output**: [1, 3, 2, 4]

Chapter 5

Conclusions and Future Work

5.1 Summary of Key Findings

This chapter provides a detailed summary of the key findings and contributions derived from the research presented in this thesis. The results are synthesized from the five foundational studies, each addressing different aspects of applying machine learning techniques to combinatorial optimization problems.

5.1.1 Recap of Results and Contributions to the Field

Learning Algorithm for Job-Shop Scheduling Problem. The introduction of Reinforced Adaptive Staircase Curriculum Learning (RASCL) provided a novel approach to improving the performance of reinforcement learning models in solving the Job Shop Scheduling Problem. The RASCL strategy effectively reduced the average optimality gap from 19.35% to 10.46% on Taillard's instances and from 38.43% to 18.85% on Demirkol's instances, showcasing its superiority over traditional scheduling heuristics and other RL-based models. The model demonstrated size-agnostic behavior and robust generalization across various JSP instances, proving its applicability to different problem scales. This study contributes to the integration of curriculum learning strategies within RL frameworks for complex scheduling problems. The RASCL method presents a scalable and efficient solution, with the potential to be applied to other combinatorial optimization problems where progressive learning is beneficial. The findings emphasize the importance of adaptive learning strategies in improving the effectiveness of RL models, particularly in complex and large-scale industrial applications.

Learning Algorithm for Vehicle Routing Problem with Correlated Stochastic Demands. The RL framework introduced for the Stochastic Vehicle Routing Problem with Correlated Demands demonstrated an improvement in handling uncertainty and demand variability in vehicle routing scenarios. The model outperformed state-of-the-art metaheuristics, with an average performance gain of 3.26% over traditional approaches. This improvement is particularly notable given the complexity of incorporating correlated stochastic variables into the routing decision

process. The approach's robustness was further validated through its adaptability to varying vehicle capacities, demand correlations, and environmental conditions, highlighting its practical applicability in dynamic logistics environments. This study advances the use of reinforcement learning in solving stochastic vehicle routing problems, particularly in scenarios where demand correlations significantly impact routing efficiency. By integrating observable stochastic variables into the RL framework, this work opens new avenues for developing more accurate and adaptable routing algorithms. The study underscores the potential of RL to not only match but exceed the performance of traditional metaheuristic approaches in real-world logistics applications, paving the way for broader adoption of machine learning in operational decision-making.

Learning Algorithm for Vehicle Routing Problem with Stochastic Travel Times. The RL-based approach for solving the Stochastic Vehicle Routing Problem introduced in this study achieved superior performance compared to traditional metaheuristics, such as Ant Colony Optimization (ACO), with an average reduction of 3.43% in travel times. The model's ability to incorporate stochastic customer demands and travel times, along with the use of an attention mechanism, led to a more robust and adaptable routing strategy. The study also highlighted the importance of real-time adaptability in vehicle routing, showing that the RL model could effectively handle various levels of uncertainty and environmental factors. This research contributes to the growing body of work on RL applications in logistics by providing a framework that can handle both stochastic demands and travel times, factors that are often oversimplified in traditional models.

Learning Algorithm for Stochastic Vehicle Routing Problem with Time Windows. The RL framework developed for the Stochastic Vehicle Routing Problem with Time Windows achieved an average reduction of 1.73% in travel times compared to traditional methods like Ant Colony Optimization and Tabu Search. The model successfully integrated time windows into the routing decision process, demonstrating its capability to handle the additional complexity introduced by strict delivery deadlines. This paper advances the application of RL in vehicle routing by addressing the challenges posed by time-constrained deliveries under uncertainty. The successful integration of time windows into the RL framework represents a key innovation, making this approach highly relevant for industries that require precise delivery scheduling.

Large Language Models for Combinatorial Problems. The Self-Guiding Exploration (SGE) strategy introduced for solving combinatorial problems using Large Language Models significantly outperformed traditional prompting strategies, with a 27.84% improvement in optimization performance. The approach demonstrated a 2.46% higher accuracy in solving a variety of reasoning tasks, including arithmetic, commonsense, and symbolic reasoning, compared to existing LLM prompting techniques. The SGE method was particularly effective in handling large and complex problem instances, such as the Traveling Salesman Problem, Vehicle Routing Problem, and Job Scheduling Problem. The findings highlight the potential of LLMs to go beyond traditional NLP tasks and contribute to solving some of the most challenging optimization problems, thereby expanding the scope of LLM applications in operations research and beyond.

5.2 Future Research Directions

As the field of machine learning and combinatorial optimization continues to evolve, there are numerous opportunities for further exploration and development. This section outlines potential areas for future research and offers suggestions for enhancing the efficiency and applicability of the models developed in this thesis. By building on the findings and methodologies presented, future research can push the boundaries of what is possible in solving complex combinatorial problems with advanced machine learning techniques.

Expansion of Curriculum Learning Strategies to Other Domains. While the Reinforced Adaptive Staircase Curriculum Learning strategy demonstrated significant success in the Job Shop Scheduling Problem, there is potential to apply this method to other combinatorial optimization problems, such as the Knapsack Problem, Graph Coloring, or even more complex routing problems like the Multi-Depot VRP. Future research could explore how adaptive curriculum learning can be tailored to the unique challenges of these problems, potentially leading to improved performance and generalization in these areas. Another promising area of exploration is the integration of curriculum learning with other machine learning paradigms, such as meta-learning. Combining these approaches could enhance the ability of models to adapt to new tasks or environments more quickly, making them more versatile and efficient in solving a broader range of combinatorial problems.

Deepening the Understanding of Reinforcement Learning in Stochastic Environments. The current research has shown the effectiveness of reinforcement learning in handling stochastic demands and travel times in vehicle routing problems. However, future research could further deepen this understanding by incorporating additional sources of stochasticity, such as stochastic customer availability, or fluctuating fuel costs. Exploring how RL models can adapt to these additional variables could lead to more robust and realistic optimization solutions in logistics and transportation. As RL continues to evolve, there is potential to explore more advanced architectures, such as hierarchical RL or multi-agent RL, in solving complex combinatorial problems. Hierarchical RL could decompose large problems into smaller, more manageable subproblems, while multi-agent RL could model scenarios where multiple decision-makers (e.g., multiple vehicles in VRP) need to coordinate their actions.

Enhancing the Role of Large Language Models in Optimization. While the Self-Guiding Exploration strategy showed significant promise in solving problems like the Traveling Salesman Problem and Vehicle Routing Problem, there is considerable potential to apply this approach to other combinatorial problems. Future research could investigate the application of LLMs to problems such as Network Design, Facility Location, or Resource Allocation, where the ability to generate and refine solutions iteratively could offer substantial benefits. Another area for exploration is the integration of LLMs with other artificial intelligence technologies, such as deep reinforcement learning or evolutionary algorithms. Combining the generative capabilities of LLMs with the adaptive and exploratory nature of these other techniques could lead to more powerful and efficient optimization frameworks.

Cross-Domain Application of Methodologies. Transferability to Non-Optimization Problems: The methodologies developed in this research, particularly those involving reinforcement learning and curriculum learning, could be explored for their applicability to non-optimization problems. For instance, RL with curriculum learning could be used in areas like automated planning, robotics, or adaptive control systems, where decision-making under uncertainty is critical. There is potential for interdisciplinary research that combines insights from operations research, computer science, and other fields such as economics, biology, or sociology. For example, understanding how optimization techniques can be applied to resource allocation in healthcare, or modeling the spread of information or disease in networks, could open new avenues for research and practical applications.

Enhancing Practical Applicability. User-Friendly Interfaces and Tools: To bridge the gap between research and practice, future work could focus on developing user-friendly interfaces and tools that allow practitioners to easily apply the models and techniques developed in this thesis. This could involve creating software libraries, visualization tools, or decision-support systems that integrate machine learning models with optimization techniques in a way that is accessible to non-experts. Conducting pilot projects in collaboration with industry partners could provide valuable insights into the practical challenges and opportunities associated with deploying these models in real-world settings. These projects could help validate the models, identify areas for improvement, and demonstrate their value in operational decision-making processes.

References

[1] Samuel Kwame Amponsah, Dominic Otoo, Said Salhi, and Ebenezer Quayson. Proposed heuristic method for solving assignment problems. *American Journal of Operations Research*, 06:436–441, 01 2016.

[2] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub W. Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Joshua Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39:20 – 3, 2020.

[3] Nyothiri Aung, Sahraoui Dhelim, Liming Chen, Abderrahmane Lakas, Wenyin Zhang, Huansheng Ning, Souleyman Chaib, and Mohand Tahar Kechadi. Vesonet: Traffic-aware content caching for vehicular social networks using deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–12, 2023.

[4] Nyothiri Aung, Weidong Zhang, Sahraoui Dhelim, and Yibo Ai. T-coin: Dynamic traffic congestion pricing system for the internet of vehicles in smart cities. *Information*, 11(3), 2020.

[5] M Emin Aydin and Ercan Öztemel. Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems*, 33(2-3):169–178, 2000.

[6] A. Bdeir, J. K. Falkner, and L. Schmidt-Thieme. Attention, filling in the gaps for generalization in routing problems. In *Proceedings of Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 505–520, 2022.

[7] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[8] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *Proceedings of International Conference on Learning Representations (workshop)*, 2017.

[9] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *ArXiv*, abs/1611.09940, 2017.

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

[11] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 41–48. ACM, 2009.

[12] F. Berto, C. Hua, J. Park, M. Kim, H. Kim, J. Son, H. Kim, J. Kim, and J. Park. Rl4co: A unified reinforcement learning for combinatorial optimization library. In *Proceedings of Advances of Neural Information Processing Systems (workshop)*, 2023.

[13] J. Bi, Y. Ma, J. Wang, Z. Cao, J. Chen, Y. Sun, and Y. M. Chee. Learning generalizable models for vehicle routing problems via knowledge distillation. In *Proceedings of Advances in Neural Information Processing Systems*, pages 31226–31238, 2022.

[14] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, NY, 2006.

[15] Federica Bomboi, Christoph Buchheim, and Jonas Pruente. On the stochastic vehicle routing problem with time windows, correlated travel times, and time dependency. *4OR*, pages 1–23, 2021.

[16] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[17] Pop Petrică C., Cosma Ovidiu, Sabo Cosmin, and Sitar Corina Pop. A comprehensive survey on the generalized traveling salesman problem. *European Journal of Operational Research*, XXX:XXX–XXX, 2024. This paper investigates extensions of the classical TSP, focusing on the generalized traveling salesman problem and its applications.

[18] X. Cai, C. Xia, Q. Zhang, Z. Mei, H. Hu, L. Wang, and J. Hu. The collaborative local search based on dynamic-constrained decomposition with grids for combinatorial multiobjective optimization. *IEEE Transactions on Cybernetics*, 51(5):2639–2650, 2021.

[19] C. Caramanis, D. Fotakis, A. Kalavasis, V. Kontonis, and C. Tzamos. Optimizing solution-samplers for combinatorial problems: The landscape of policy-gradient method. In *Proceedings of Advances in Neural Information Processing Systems*, pages 14035–14069, 2023.

[20] K. J. Chaitanya, L. Thomas, and B. Xavier. On learning paradigms for the travelling salesman problem. In *Proceedings of Advances in Neural Information Processing Systems (workshop)*, 2019.

[21] J. Chen, Z. Gong, M. Liu, J. Wang, Y. Yu, and W. Zhang. Looking ahead to avoid being late: Solving hard-constrained traveling salesman problem. *arXiv preprint arXiv:2403.05318*, 2024.

[22] J. Chen, J. Wang, Z. Zhang, Z. Cao, T. Ye, and C. Siyuan. Efficient meta neural heuristic for multi-objective combinatorial optimization. In *Proceedings of Advances in Neural Information Processing Systems*, pages 56825–56837, 2023.

[23] J. Chen, Z. Zhang, Z. Cao, Y. Wu, Y. Ma, T. Ye, and J. Wang. Neural multi-objective combinatorial optimization with diversity enhancement. In *Proceedings of Advances in Neural Information Processing Systems*, pages 39176–39188, 2023.

[24] H. Cheng, Y. Cong, W. Jiang, and S. Pu. Learning to solve class-constrained bin packing problems via encoder-decoder model. In *Proceedings of International Conference on Learning Representations*, pages 1–24, 2024.

[25] Sheng Jie Chin, Matthias Winkenbach, and Anirudh Srivastava. Learning to deliver: a foundation model for the montreal capacitated vehicle routing problem. *arXiv preprint arXiv:2403.00026*, 2024.

[26] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

[27] Christian H. Christiansen and Jens Lysgaard. A branch-and-price algorithm for the capacitated vehicle routing problem with stochastic demands. *Oper. Res. Lett.*, 35:773–781, 2007.

[28] Jean-François Cordeau, Gilbert Laporte, Martin WP Savelsbergh, and Daniele Vigo. Vehicle routing. *Handbooks in operations research and management science*, 14:367–428, 2007.

[29] C. Dang, C. Bazgan, T. Cazenave, M. Chopin, and P.-H. Wuillemin. Warm-starting nested rollout policy adaptation with optimal stopping. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 12381–12389, 2023.

[30] George B. Dantzig and John Hubert Ramser. The truck dispatching problem. *Management Science*, 6:80–91, 1959.

[31] V.-A. Darvariu, S. Hailes, and M. Musolesi. Graph reinforcement learning for combinatorial optimization: A survey and unifying perspective. *arXiv preprint arXiv:2404.06492*, 2024.

[32] S. Das, A. Abraham, U. K. Chakraborty, and A. Konar. Differential evolution using a neighborhood-based mutation operator. *IEEE Transactions on Evolutionary Computation*, 13(3):526–553, 2009.

[33] A. Delarue, R. Anderson, and C. Tjandraatmadja. Reinforcement learning with combinatorial actions: An application to vehicle routing. In *Advances in Neural Information Processing Systems*, volume 33, pages 609–620, 2020.

[34] Ebru Demirkol, Sanjay Mehta, and Reha Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137–141, August 1998.

[35] T. Dernedde, D. Thyssens, S. Dittrich, M. Stubbemann, and L. Schmidt-Thieme. Moco: A learnable meta optimizer for combinatorial optimization, 2024.

[36] Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. Learning heuristics for the tsp by policy gradient. In *International conference on the integration of constraint programming, artificial intelligence, and operations research*, pages 170–181. Springer, 2018.

[37] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.

[38] David Drakulic, Sébastien Michel, Florian Mai, Arnaud Sors, and Jean-Marc Andreoli. Bq-nco: Bisimulation quotienting for efficient neural combinatorial optimization. In *Proceedings of Advances in Neural Information Processing Systems*, pages 77416–77429, 2023.

[39] Iddo Drori, Anant Kharkar, William R Sickinger, Brandon Kates, Qiang Ma, Suwen Ge, Eden Dolev, Brenda Dietrich, David P Williamson, and Madeleine Udell. Learning to solve combinatorial optimization problems on real-world graphs in linear time. In *Proceedings of IEEE International Conference on Machine Learning and Applications*, pages 19–24, 2020.

[40] J. K. Falkner and L. Schmidt-Thieme. Learning to solve vehicle routing problems with time windows through joint attention. *arXiv preprint arXiv:2006.09100*, 2020.

[41] H. Fang, Z. Song, P. Weng, and Y. Ban. Invit: A generalizable routing problem solver with invariant nested view transformer. In *Proceedings of International Conference on Machine Learning*, 2024.

[42] S. Foa, C. Coppola, G. Grani, and L. Palagi. Solving the vehicle routing problem with deep reinforcement learning. *arXiv preprint arXiv:2208.00202*, 2022.

[43] Thomas Gabel and Martin Riedmiller. Adaptive reactive job-shop scheduling with reinforcement learning agents. *International Journal of Information Technology and Intelligent Computing*, 24(4):14–18, 2008.

[44] C. Gao, H. Shang, K. Xue, D. Li, and C. Qian. Towards generalizable neural solvers for vehicle routing problems via ensemble with transferrable local policy. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2024.

[45] Ricardo García-Torres, Adolfo A Macias-Infante, Santiago E Conant-Pablos, José Carlos Ortiz-Bayliss, and Hugo Terashima-Marín. Combining constructive and perturbative deep learning algorithms for the capacitated vehicle routing problem. *arXiv preprint arXiv:2211.13922*, 2022.

[46] Michael R. Garey and David S. Johnson. Computers and intractability: A guide to the theory of np-completeness. 1979. A foundational text that introduces NP-completeness and NP-hardness, including various combinatorial problems.

[47] A. I. Garmendia, J. Ceberio, and A. Mendiburu. Applicability of neural combinatorial optimization: A critical view. *ACM Transactions on Evolutionary Learning and Optimization*, 2024. in press.

[48] Aitziber Ibañez Garmendia, Josu Ceberio, and Alexander Mendiburu. Neural improvement heuristics for graph combinatorial optimization problems. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[49] Charles Gauvin, Guy Desaulniers, and Michel Gendreau. A branch-cut-and-price algorithm for the vehicle routing problem with stochastic demands. *Comput. Oper. Res.*, 50:141–153, 2014.

[50] S. Geisler, J. Sommer, J. Schuchardt, A. Bojchevski, and S. Gunnemann. Generalization of neural combinatorial solvers through the lens of adversarial robustness. In *Proceedings of International Conference on Learning Representations*, pages 1–21, 2022.

[51] Michel Gendreau, Gilbert Laporte, and René Séguin. Stochastic vehicle routing. *European Journal of Operational Research*, 88(1):3–12, 1996.

[52] Rajeev Kumar Goel, Raman Maini, and Sandhya Bansal. Vehicle routing problem with time windows having stochastic customers demands and stochastic service times: Modelling and solution. *J. Comput. Sci.*, 34:1–10, 2019.

[53] Y. L. Goh, W. S. Lee, X. Bresson, T. Laurent, and N. Lim. Combining reinforcement learning and optimal transport for the traveling salesman problem. *arXiv preprint arXiv:2203.00903*, 2022.

[54] Bruce L. Golden, S. Raghavan, and Edward A. Wasil, editors. *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces Series*. Springer, 2008. This book reflects the most recent scholarship and challenges in vehicle routing, including various algorithms and applications.

[55] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016.

[56] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. In *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[57] Google. Or-tools, 2023.

[58] Nathan Grinsztajn, Daniel Furelos-Blanco, Shikha Surana, Clément Bonnet, and Tom Barrett. Winner takes it all: Training performant rl populations for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:48485–48509, 2023.

[59] X. Guo, X. Liu, E. Zhu, X. Zhu, M. Li, X. Xu, and J. Yin. Adaptive self-paced deep clustering with data augmentation. *IEEE Transactions on Knowledge and Data Engineering*, 32(9):1680–1693, 2020.

[60] Y. Habib and A. Filchenkov. Multi-agent reinforcement learning for multi vehicles one-commodity vehicle routing problem. *Procedia Computer Science*, 212:418–428, 2022.

[61] Y. He, G. Wu, Y. Chen, and W. Pedrycz. A two-stage framework and reinforcement learning-based optimization algorithms for complex scheduling problems. *arXiv preprint arXiv:2103.05847*, 2021.

[62] K. Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126:106–130, 2000.

[63] Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12:966–980, 2017.

[64] Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems: Technical report. 2017.

[65] Dorit S. Hochbaum. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, Boston, MA, 1996.

[66] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[67] Holger H. Hoos and Thomas Stützle. Introduction: Combinatorial problems and search. In *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, 2005. Discusses combinatorial problems, including the TSP, and various search paradigms.

[68] A. Hottung, M. Mahajan, and K. Tierney. Polynet: Learning diverse solution strategies for neural combinatorial optimization. arXiv: 2402.14048, 2024.

[69] A. Hottung, K. Tierney, and B. Bhandari. Learning a latent search space for routing problems using variational autoencoders. In *Proceedings of International Conference on Learning Representations*, pages 1–13, 2021.

[70] André Hottung and Kevin Tierney. Neural large neighborhood search for the capacitated vehicle routing problem. In *Proceedings of European Conference on Artificial Intelligence*, pages 1–8, 2020.

[71] Q. Hou, J. Yang, Y. Su, X. Wang, and Y. Deng. Generalize learned heuristics to solve large-scale vehicle routing problems in real-time. In *Proceedings of International Conference on Learning Representations*, pages 1–37, 2023.

[72] T. Huang, A. M. Ferber, Y. Tian, B. Dilkina, and B. Steiner. Searching large neighborhoods for integer linear programs with contrastive learning. In *International Conference on Machine Learning*, pages 13869–13890. PMLR, 2023.

[73] Zangir Iklassov, Yali Du, Farkhad Akimov, and Martin Takac. Self-guiding exploration for combinatorial problems, 2024.

[74] Zangir Iklassov, Dmitrii Medvedev, Ruben Solozabal Ochoa de Retana, and Martin Takác. On the study of curriculum learning for inferring dispatching policies on the job shop scheduling. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 5350–5358. ijcai.org, 2023.

[75] Zangir Iklassov, Dmitrii Medvedev, Ruben Solozabal, and Martin Takac. Learning to generalize dispatching rules on the job shop scheduling. *arXiv preprint arXiv:2206.04423*, 2022.

[76] Zangir Iklassov, Ikboljon Sobirov, Ruben Solozabal, and Martin Takác. Reinforcement learning approach to stochastic vehicle routing problem with correlated demands. *IEEE Access*, 11:87958–87969, 2023.

[77] Zangir Iklassov, Ikboljon Sobirov, Rubén Solozabal, and Martin Takác. Reinforcement learning for solving stochastic vehicle routing problem. In *Asian Conference on Machine Learning*, 2023.

[78] Zangir Iklassov, Ikboljon Sobirov, Ruben Solozabal, and Martin Takac. Reinforcement learning for solving stochastic vehicle routing problem with time windows, 2024.

[79] Helga Ingimundardottir and Thomas Philip Runarsson. Discovering dispatching rules from data using imitation learning: A case study for the job-shop problem. *Journal of Scheduling*, 21(4):413–428, 2018.

[80] Y. Jiang, Z. Cao, Y. Wu, W. Song, and J. Zhang. Ensemble-based deep reinforcement learning for vehicle routing problems under distribution shift. In *Proceedings of Advances in Neural Information Processing Systems*, pages 53112–53125, 2023.

[81] Y. Jiang, Z. Cao, Y. Wu, and J. Zhang. Multi-view graph contrastive learning for solving vehicle routing problems. In *Proceedings of Conference on Uncertainty in Artificial Intelligence*, pages 984–994, 2023.

[82] Y. Jiang, Y. Wu, Z. Cao, and J. Zhang. Learning to solve routing problems via distributionally robust optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 9786–9794, 2022.

[83] Y. Jin, Y. Ding, X. Pan, K. He, L. Zhao, T. Qin, L. Song, and J. Bian. Pointerformer: Deep reinforced multi-pointer transformer for the traveling salesman problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 8132–8140, 2023.

[84] Waldy Joe and Hoong Chuin Lau. Deep reinforcement learning approach to solve dynamic vehicle routing problem with stochastic customers. In *ICAPS*, 2020.

[85] David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256–278, 1974. Analyzes simple, polynomial-time heuristic algorithms for finding approximate solutions to various NP-complete optimization problems.

[86] C. K. Joshi, Q. Cappart, L.-M. Rousseau, and T. Laurent. Learning tsp requires rethinking generalization. In *Proceedings of International Conference on Principles and Practice of Constraint Programming*, pages 70–98, 2021.

[87] Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

[88] Y. Kaempfer and L. Wolf. Learning the multiple traveling salesmen problem with permutation invariant pooling networks. *arXiv preprint arXiv:1803.09621*, 2018.

[89] Panagiotis Kalatzantonakis, Angelo Sifaleras, and Nikolaos Samaras. A reinforcement learning-variable neighborhood search method for the capacitated vehicle routing problem. *Expert Systems with Applications*, 213:118812, 2023.

[90] M. Karimi-Mamaghan, M. Mohammadi, P. Meyer, A. M. Karimi-Mamaghan, and E.-G. Talbi. Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *European Journal of Operational Research*, 296(2):393–422, 2022.

[91] J. Ke, F. Xiao, H. Yang, and J. Ye. Learning to delay in ride-sourcing systems: A multi-agent deep reinforcement learning framework. *IEEE Transactions on Knowledge and Data Engineering*, 34(5):2280–2292, 2022.

[92] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Proceedings of Advances in Neural Information Processing Systems*, volume 30, pages 6351–6361, 2017.

[93] Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*, 2022.

[94] M. Kim, J. Park, and J. Park. Sym-nco: Leveraging symmetricity for neural combinatorial optimization. In *Proceedings of Advances in Neural Information Processing Systems*, pages 1936–1949, 2022.

[95] M. Kim, J. Son, H. Kim, and J. Park. Scale-conditioned adaptation for large scale combinatorial optimization. In *Proceedings of Advances in Neural Information Processing Systems (workshop)*, 2022.

[96] Minsu Kim, Jinkyoo Park, et al. Learning collaborative policies to solve np-hard routing problems. In *Proceedings of Advances in Neural Information Processing Systems*, pages 10418–10430, 2021.

[97] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2014.

[98] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[99] G. Kizilateş and F. Nuriyeva. On the nearest neighbor algorithms for the traveling salesman problem. In *Advances in Computational Science, Engineering and Information Technology*, volume 225, pages 1–10. Springer, Heidelberg, 2013.

[100] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. In *NeurIPS*, 2022.

[101] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. *Advances in Neural Information Processing Systems*, 12:1008–1014, 1999.

[102] Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

[103] Bernard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, Berlin, Germany, 2008.

[104] Y.-D. Kwon, J. Choo, I. Yoon, M. Park, D. Park, and Y. Gwon. Matrix encoding networks for neural combinatorial optimization. In *Proceedings of Advances in Neural Information Processing Systems*, pages 5138–5149, 2021.

[105] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, 33:21188–21198, 2020.

[106] Gilbert Laporte. Fifty years of vehicle routing. *Transp. Sci.*, 43:408–416, 2009.

[107] Gilbert Laporte and François V. Louveaux. The integer l-shaped method for stochastic integer programs with complete recourse. *Oper. Res. Lett.*, 13:133–142, 1993.

[108] Eugene L Lawler and David E Wood. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

[109] K. Lei, P. Guo, Y. Wang, X. Wu, and W. Zhao. Solve routing problems with a residual edge-graph attention neural network. *Neurocomputing*, 508:79–98, 2022.

[110] Jonathan Levin, Ryan Correll, Takayuki Ide, Tatsunori Suzuki, Tomoki Saito, and Akira Arai. Deep reinforcement learning for multi-truck vehicle routing problems with multi-leg demand routes. *arXiv preprint arXiv:2401.08669*, 2024.

[111] Duanshun Li, Jing Liu, Dongeun Lee, Ali Seyedmazloom, Giridhar Kaushik, Kookjin Lee, and Noseong Park. A novel method to solve neural knapsack problems. In *Proceedings of the 38th International Conference on Machine Learning*, pages 6414–6424. PMLR, 2021.

[112] Guoming Li and Junhua Li. An improved tabu search algorithm for the stochastic vehicle routing problem with soft time windows. *IEEE Access*, 8:158115–158124, 2020.

[113] J. Li, L. Xin, Z. Cao, A. Lim, W. Song, and J. Zhang. Heterogeneous attentions for solving pickup and delivery problem via deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 23(3):2306–2315, 2022.

[114] Junhao Li, Yaoxin Ma, Ruibin Gao, Zongzhang Cao, Andrew Lim, Weiping Song, and Jianye Zhang. Deep reinforcement learning for solving the heterogeneous capacitated vehicle routing problem. *IEEE Transactions on Cybernetics*, 52(12):13572–13585, 2022.

[115] K. Li, T. Zhang, and R. Wang. Deep reinforcement learning for multiobjective optimization. *IEEE Transactions on Cybernetics*, 51(6):3103–3114, 2021.

[116] K. Li, T. Zhang, R. Wang, Y. Wang, Y. Han, and L. Wang. Deep reinforcement learning for combinatorial optimization: Covering salesman problems. *IEEE Transactions on Cybernetics*, 52(12):13142–13155, 2022.

[117] S. Li, Z. Yan, and C. Wu. Learning to delegate for large-scale vehicle routing. In *Proceedings of Advances in Neural Information Processing Systems*, pages 26198–26211, 2021.

[118] X. Li, W. Luo, M. Yuan, J. Wang, J. Lu, J. Wang, J. Lu, and J. Zeng. Learning to optimize industry-scale dynamic pickup and delivery problems. In *Proceedings of IEEE International Conference on Data Engineering*, pages 2511–2522, 2021.

[119] Y. Li, J. Guo, R. Wang, and J. Yan. T2t: From distribution learning in training to gradient search in testing for combinatorial optimization. In *Proceedings of Advances in Neural Information Processing Systems*, pages 50020–50040, 2023.

[120] B. Lin, B. Ghaddar, and J. Nathwani. Deep reinforcement learning for the electric vehicle routing problem with time windows. *IEEE Transactions on Intelligent Transportation Systems*, 23(8):11528–11538, 2021.

[121] B. Lin, B. Ghaddar, and J. Nathwani. Deep reinforcement learning for the electric vehicle routing problem with time windows. *IEEE Transactions on Intelligent Transportation Systems*, 23(8):11528–11538, 2022.

[122] Chun-Cheng Lin, Der-Jiunn Deng, Yen-Ling Chih, and Hsin-Ting Chiu. Smart manufacturing scheduling with edge computing using multiclass deep q network. *IEEE Transactions on Industrial Informatics*, 15(7):4276–4284, 2019.

[123] X. Lin, Z. Yang, and Q. Zhang. Pareto set learning for neural multiobjective combinatorial optimization. In *Proceedings of International Conference on Learning Representations*, pages 1–31, 2022.

[124] Z. Lin, Y. Wu, B. Zhou, Z. Cao, W. Song, Y. Zhang, and S. Jayavelu. Cross-problem learning for solving vehicle routing problems. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2024.

[125] Michal Lisicki, Arash Afkanpour, and Graham W Taylor. Evaluating curriculum learning strategies in neural combinatorial optimization. *arXiv preprint arXiv:2011.06188*, 2020.

[126] Chien-Liang Liu, Chuan-Chin Chang, and Chun-Jan Tseng. Actor-critic deep reinforcement learning for solving job shop scheduling problems. *Ieee Access*, 8:71752–71762, 2020.

[127] F. Liu, X. Lin, Q. Zhang, X. Tong, and M. Yuan. Multi-task learning for routing problem with cross-problem zero-shot generalization. *arXiv preprint arXiv:2402.16891*, 2024.

[128] Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language mode, 2024.

[129] S. Liu, X. Yan, and Y. Jin. An edge-aware graph autoencoder trained on scale-imbalanced data for travelling salesman problems. *Knowledge-Based Systems*, 291:111559, 2024.

[130] François V. Louveaux. An introduction to stochastic transportation models. 1998.

[131] Hao Lu, Xingwen Zhang, and Shuang Yang. A learning-based iterative method for solving vehicle routing problems. In *ICLR*, 2020.

[132] Haoyu Lu, Zhihao Li, Ruijie Wang, Qiuchen Ren, Xiang Li, Ming Yuan, Jialin Zeng, Xiangning Yang, and Junchi Yan. Roco: A general framework for evaluating robustness of combinatorial optimization solvers on graphs. In *Proceedings of International Conference on Learning Representations*, pages 1–23, 2023.

[133] Fei Luo, Xiaofeng Lin, Zhiguang Wang, Tao Xialiang, Mingli Yuan, and Qingfu Zhang. Self-improved learning for scalable neural combinatorial optimization. *arXiv preprint arXiv:2403.19561*, 2024.

[134] Fu Luo, Xi Lin, Fei Liu, Qingfu Zhang, and Zhenkun Wang. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. *Advances in Neural Information Processing Systems*, 36, 2024.

[135] L. Luttmann and L. Xie. Neural combinatorial optimization on heterogeneous graphs. an application to the picker routing problem in mixedshelves warehouses. In *Proceedings of International Conference on Automated Planning and Scheduling*, pages 1–9, 2024.

[136] Q. Ma, S. Ge, D. He, D. Thaker, and I. Drori. Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (workshop)*, 2020.

[137] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. *ArXiv preprint*, abs/2303.17651, 2023.

[138] Sridhar Mahadevan, Nicholas Marchalleck, Tapas K Das, and Abhijit Gosavi. Self-improving factory simulation using continuous-time average-reward reinforcement learning. In *Machine Learning Interantional Workshop*, pages 202–210, 1997.

[139] Sridhar Mahadevan and Georgios Theocharous. Optimizing production manufacturing using reinforcement learning. In *FLAIRS conference*, volume 372, page 377, 1998.

[140] Simon Mak, Lingyi Xu, Tim Pearce, Mikhail Ostroumov, and Alexandra Brintrup. Coalitional bargaining via reinforcement learning: An application to collaborative vehicle routing. *arXiv preprint arXiv:2310.17458*, 2023.

[141] S. Manchanda, S. Michel, D. Drakulic, and J.-M. Andreoli. On the generalization of neural combinatorial optimization heuristics. In *Proceedings of Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 426–442, 2022.

[142] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM special interest group on data communication*, pages 270–288. 2019.

[143] Mahmoud Masoud, Ahmed Abdelhay, and Mohammed Elhenawy. Exploring combinatorial problem solving with large language models: A case study on the travelling salesman problem using gpt-3.5 turbo, 2024.

[144] William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023.

[145] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7(4):326–329, 1960.

[146] David R Morrison, Sheldon H Jacobson, Jason J Sauppe, and Edward C Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016.

[147] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence V Snyder, and Martin Takáč. Reinforcement learning for solving the vehicle routing problem. In *Conference on Neural Information Processing Systems, NeurIPS 2018*, 2018.

[148] Yunyun Niu, Detian Kong, Rong Wen, Zhiguang Cao, and Jian hua Xiao. An improved learnable evolution model for solving multi-objective vehicle routing problem with stochastic demand. *Knowl. Based Syst.*, 230:107378, 2021.

[149] Yunyun Niu, Jie Shao, Jian hua Xiao, Wen Song, and Zhiguang Cao. Multi-objective evolutionary algorithm based on rbf network for solving the stochastic vehicle routing problem. *Inf. Sci.*, 609:387–410, 2022.

[150] A. Nowak, D. Folque, and J. Bruna. Divide and conquer networks. In *Proceedings of International Conference on Learning Representations*, pages 1–15, 2018.

[151] Afshin Oroojlooyjadid, M. Nazari, Lawrence V. Snyder, and Martin Takác. A deep q-network for the beer game: Deep reinforcement learning for inventory optimization. *Manuf. Serv. Oper. Manag.*, 24:285–304, 2022.

[152] Afshin Oroojlooyjadid, Lawrence V Snyder, and Martin Takáč. Applying deep learning to the newsvendor problem. *IISE Transactions*, 52(4):444–463, 2020.

[153] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *NeurIPS*, 2022.

[154] W. Pan and S. Q. Liu. Deep reinforcement learning for the dynamic and uncertain vehicle routing problem. *Applied Intelligence*, 53:405–422, 2023.

[155] X. Pan, Y. Jin, Y. Ding, M. Feng, L. Zhao, L. Song, and J. Bian. H-tsp: Hierarchically solving the large-scale traveling salesman problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 9343–9353, 2023.

[156] C. H. Papadimitriou and K. Steiglitz. On the complexity of local search for the traveling salesman problem. *SIAM Journal on Computing*, 6(1):76–83, 1977.

[157] Junyoung Park, Sanjar Bakhtiyar, and Jinkyoo Park. Schedulenet: Learn to solve multi-agent scheduling problems with reinforcement learning. *arXiv preprint arXiv:2106.03051*, 2021.

[158] B. Peng, J. Wang, and Z. Zhang. A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems. In *Proceedings of Artificial Intelligence Algorithms and Applications*, pages 636–650, 2020.

[159] Fernando Peres and Mauro Castelli. Combinatorial optimization problems and metaheuristics: Review, challenges, design, and development. *Applied Sciences*, 11(14), 2021.

[160] Ulrich Pferschy and Rostislav Staněk. Generating subtour elimination constraints for the tsp from pure integer solutions. *Computational Optimization and Applications*, 64(1):1–20, 2016. Discusses subtour elimination constraints and their role in solving the TSP.

[161] Tantikorn Pichpibul and Ruengsak Kawtummachai. A heuristic approach based on clarke-wright algorithm for open vehicle routing problem. *The Scientific World Journal*, 2013, 2013.

[162] Michael L. Pinedo. Scheduling: Theory, algorithms, and systems. *Springer*, 2008. A widely cited textbook covering scheduling theory, algorithms, and applications, with a focus on the JSSP.

[163] M. Prates, P. H. C. Avelar, H. Lemos, L. C. Lamb, and M. Y. Vardi. Learning to solve np-complete problems: A graph neural network for decision tsp. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4731–4738, 2019.

[164] R. Qiu, Z. Sun, and Y. Yang. Dimes: A differentiable meta solver for combinatorial optimization problems. In *Proceedings of Advances in Neural Information Processing Systems*, pages 25531–25546, 2022.

[165] Mojtaba Rajabi-Bahaabadi, Afshin Shariat, Mohsen Babaei, and Daniele Vigo. Reliable vehicle routing problem in stochastic networks with correlated travel times. *Operational Research*, 03 2021.

[166] A. Rezoug, M. Bader-el den, and D. Boughaci. Application of supervised machine learning methods on the multidimensional knapsack problem. *Neural Processing Letters*, 54:871–890, 2022.

[167] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, and G. Dueck. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2):139–171, 2000.

[168] Nicola Secomandi. Comparing neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands. *Comput. Oper. Res.*, 27:1201–1225, 2000.

[169] Veronique Sels, Nele Gheysen, and Mario Vanhoucke. A comparison of priority rules for the job shop scheduling problem under different flow time-and tardiness-related objective functions. *International Journal of Production Research*, 50(15):4255–4270, 2012.

[170] M. A. L. Silva, S. R. de Souza, M. J. F. Souza, and A. L. C. Bazzan. A reinforcement learning-based multi-agent framework applied for solving routing and scheduling problems. *Expert Systems with Applications*, 131:148–171, 2019.

[171] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[172] V. Snášel, J. Platoš, P. Krömer, and N. Ouddane. Genetic algorithms for the use in combinatorial problems. In *Foundations of Computational Intelligence*, volume 3, pages 1–10. Springer, Berlin, Heidelberg, 2009.

[173] J. Son, M. Kim, S. Choi, H. Kim, and J. Park. Equity-transformer: Solving np-hard min-max routing problems as sequential generation with equity context. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 20265–20273, 2024.

[174] J. Sui, S. Ding, R. Liu, L. Xu, and D. Bu. Learning 3-opt heuristics for traveling salesman problem via deep reinforcement learning. In *Proceedings of Asian Conference on Machine Learning*, volume 157, pages 1301–1316, 2021.

[175] N. Sultana, J. Chan, A. K. Qin, and T. Sarwar. Learning vehicle routing problems using policy optimisation. *arXiv preprint arXiv:2012.13269*, 2020.

[176] N. Sultana, J. Chan, T. Sarwar, and A. K. Qin. Learning to optimise general tsp instances. *International Journal of Machine Learning and Cybernetics*, 13:2213–2228, 2022.

[177] Nahid Sultana, Jeffrey Chan, Tanveer Sarwar, and Andy Ke Qin. Sample-efficient, exploration-based policy optimisation for routing problems. *arXiv preprint arXiv:2205.15656*, 2022.

[178] H. Sun, K. Goshvadi, A. Nova, D. Schuurmans, and H. Dai. Revisiting sampling for combinatorial optimization. In *Proceedings of International Conference on Machine Learning*, pages 32859–32874, 2023.

[179] Penghao Sun, Zehua Guo, Junchao Wang, Junfei Li, Julong Lan, and Yuxiang Hu. Deepweave: Accelerating job completion time with deep reinforcement learning-based coflow scheduling. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 3314–3320, 2021.

[180] Ruiping Sun, Zhihan Zheng, and Zhanhao Wang. Learning encodings for constructive neural combinatorial optimization needs to regret. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 20803–20811, 2024.

[181] Z. Sun and Y. Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. In *Proceedings of Advances in Neural Information Processing Systems*, pages 3706–3731, 2023.

[182] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[183] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[184] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed Chi, Denny Zhou, and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them. In Anna Rogers,

Jordan L. Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 13003–13051. Association for Computational Linguistics, 2023.

[185] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, January 1993.

[186] E. D. Taillard and K. Helsgaun. Popmusic for the travelling salesman problem. *European Journal of Operational Research*, 272(2):420–429, 2019.

[187] Qingchao Tang, Yuchen Kong, Longkang Pan, and Chi-Kin Lee. Learning to solve soft-constrained vehicle routing problems with lagrangian relaxation. *arXiv preprint arXiv:2207.09860*, 2022.

[188] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam M. Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, Yaguang Li, Hongrae Lee, Huaixiu Zheng, Amin Ghafouri, Marcelo Menegali, Yanping Huang, Maxim Krikun, Dmitry Lepikhin, James Qin, Dehao Chen, Yuanzhong Xu, Zhifeng Chen, Adam Roberts, Maarten Bosma, Yanqi Zhou, Chung-Ching Chang, I. A. Krivokon, Willard James Rusch, Marc Pickett, Kathleen S. Meier-Hellstern, Meredith Ringel Morris, Tulsee Doshi, Renelito Delos Santos, Toju Duke, Johnny Hartz Søraker, Ben Zevenbergen, Vinodkumar Prabhakaran, Mark Díaz, Ben Hutchinson, Kristen Olson, Alejandra Molina, Erin Hoffman-John, Josh Lee, Lora Aroyo, Ravindran Rajakumar, Alena Butryna, Matthew Lamm, V. O. Kuzmina, Joseph Fenton, Aaron Cohen, Rachel Bernstein, Ray Kurzweil, Blaise Aguera-Arcas, Claire Cui, Marian Croak, Ed Huai hsin Chi, and Quoc Le. Lamda: Language models for dialog applications. *ArXiv preprint*, abs/2201.08239, 2022.

[189] Daniel Thyssens, Julian Falkner, and Lars Schmidt-Thieme. Supervised permutation invariant networks for solving the cvrp with bounded fleet size. *arXiv preprint arXiv:2201.01529*, 2022.

[190] P. Toth and D. Vigo. Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, 123(1):487–512, 2002.

[191] Paolo Toth and Daniele Vigo. Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, 123(1-3):487–512, 2002.

[192] Paolo Toth and Daniele Vigo. Vehicle routing: Problems, methods, and applications, second edition. 2014.

[193] Learns Old Tricks. A new dog learns old tricks : Rl finds classic optimization algorithms. 2018.

[194] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, u. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of Advances in Neural Information Processing Systems*, pages 6000–6010, 2017.

[195] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[196] T. Vidal. Hybrid genetic search for the cvrp: Open-source implementation and swap neighborhood. *Computers & Operations Research*, 140:105643, 2022.

[197] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

[198] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

[199] Cheng-Pin Wan, Tianyi Li, and June-Ming Wang. Rlor: A flexible framework of deep reinforcement learning for operation research. *arXiv preprint arXiv:2303.13117*, 2023.

[200] C. Wang and T. Yu. Efficient training of multi-task combinarotial neural solver with multi-armed bandits. *arXiv preprint arXiv:2305.06361*, 2023.

[201] C. Wang, Z. Yu, S. McAleer, T. Yu, and Y. Yang. Asp: Learn a universal neural solver! *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(6):4102–4114, 2024. in press.

[202] Chengjia Wang, Peng Cheng, Jianan Li, and Weijie Sun. Leader reward for pomo-based neural combinatorial optimization. *arXiv preprint arXiv:2405.13947*, 2024.

[203] Libing Wang, Xin Hu, Yin Wang, Sujie Xu, Shijun Ma, Kexin Yang, Zhijun Liu, and Weidong Wang. Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning. *Computer Networks*, 190:107969, 2021.

[204] Q. Wang. Alpha-t: Learning to traverse over graphs with an alphazero-inspired self-play framework. 2021.

[205] Runzhong Wang, Yunhao Zhang, Ziao Guo, Tianyi Chen, Xiaokang Yang, and Junchi Yan. Linsatnet: the positive linear satisfiability neural networks. In *International Conference on Machine Learning*, pages 36605–36625. PMLR, 2023.

[206] Xin Wang, Yudong Chen, and Wenwu Zhu. A survey on curriculum learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[207] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

[208] Y. Wang, S. Sun, and W. Li. Hierarchical reinforcement learning for vehicle routing problems with time windows. In *Canadian AI*, June 2021.

[209] Z. Wang, Q. Yan, L. Wang, and X. Hei. Solving the vehicle routing problem with time window based on reinforcement learning. In *2023 19th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pages 1–7. IEEE, 2023.

[210] Bernd Waschneck, André Reichstaller, Lenz Belzner, Thomas Altenmüller, Thomas Bauernhansl, Alexander Knapp, and Andreas Kyek. Optimization of global production scheduling with deep reinforcement learning. *Procedia Cirp*, 72:1264–1269, 2018.

[211] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

[212] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022. Survey Certification.

[213] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc., 2022.

[214] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

[215] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[216] H. Wu, J. Wang, and Z. Zhang. Modrl/d-am: Multiobjective deep reinforcement learning algorithm using decomposition and attention model for multiobjective optimization. In *Proceedings of International Symposium on Intelligence Computation and Applications*, pages 575–589, 2020.

[217] Xuan Wu, Jizong Han, Di Wang, Pengyue Gao, Quanlong Cui, Liang Chen, Yanchun Liang, Han Huang, Heow Pueh Lee, Chunyan Miao, et al. Incorporating surprisingly popular algorithm and euclidean distance-based adaptive topology into pso. *Swarm and Evolutionary Computation*, 76:101222, 2023.

[218] B. Xavier and L. Thomas. The transformer network for the traveling salesman problem, 2021.

[219] Y. Xiao, D. Wang, B. Li, H. Chen, W. Pang, X. Wu, H. Li, D. Xu, Y. Liang, and Y. Zhou. Reinforcement learning-based non-autoregressive solver for traveling salesman problems. *arXiv preprint arXiv:2308.00560*, 2023.

[220] Yubin Xiao, Di Wang, Boyang Li, Mingzhao Wang, Xuan Wu, Changliang Zhou, and You Zhou. Distilling autoregressive models to obtain high-performance non-autoregressive solvers for vehicle routing problems with faster inference speed. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20274–20283, 2024.

[221] W. Xiaohe and S. Xinli. Deep reinforcement learning for the capacitated vehicle routing problem with soft time window. In *2022 14th International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 352–355. IEEE, 2022.

[222] F. Xie, M. Muller, and R. Holte. Adding local exploration to greedy best-first search in satisficing planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 2388–2394, 2014.

[223] L. Xin, W. Song, Z. Cao, and J. Zhang. Multi-decoder attention model with embedding glimpse for solving vehicle routing problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 12042–12049, 2021.

[224] L. Xin, W. Song, Z. Cao, and J. Zhang. Neurolkh: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem. In *Proceedings of Advances in Neural Information Processing Systems*, pages 7472–7483, 2021.

[225] S. Xu, S. S. Panwar, M. Kodialam, and T. Lakshman. Deep neural network approximated dynamic programming for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1684–1691, 2020.

[226] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers, 2024.

[227] F. Yang, T. Jin, T.-Y. Liu, X. Sun, and J. Zhang. Boosting dynamic programming with neural networks for solving np-hard problems. In *Proceedings of Asian Conference on Machine Learning*, pages 726–739, 2018.

[228] Wenzhe Yang, Di Wang, Wei Pang, Ah-Hwee Tan, and You Zhou. Goods consumed during transit in split delivery vehicle routing problems: Modeling and solution. *IEEE Access*, 8:110336–110350, 2020.

[229] Shunyu Yao, Xi Lin, Zhenkun Wang, and Qingfu Zhang. Data-efficient supervised learning is powerful for neural combinatorial optimization. 2023.

[230] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

[231] H. Ye, J. Wang, Z. Cao, H. Liang, and Y. Li. Deepaco: Neural-enhanced ant systems for combinatorial optimization. In *Proceedings of Advances in Neural Information Processing Systems*, 2023.

[232] H. Ye, J. Wang, H. Liang, Z. Cao, Y. Li, and F. Li. Glop: Learning global partition and local construction for solving large-scale routing problems in real-time. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 20284–20292, 2024.

[233] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. Learning to dispatch for job shop scheduling via deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1621–1632, 2020.

[234] Di Zhang, Dong Dai, Youbiao He, Forrest Sheng Bao, and Bing Xie. Rlscheduler: an automated hpc batch job scheduler using reinforcement learning. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2020.

[235] K. Zhang, F. He, Z. Zhang, X. Lin, and M. Li. Multi-vehicle routing problems with soft time windows: A multi-agent reinforcement learning approach. *Transportation Research Part C: Emerging Technologies*, 121:102861, 2020.

[236] Kai Zhang, Feng He, Zhen Zhang, Xiaoping Lin, and Miao Li. Multi-vehicle routing problems with soft time windows: A multi-agent reinforcement learning approach. *Transportation Research Part C: Emerging Technologies*, 121:102861, 2020.

[237] Wei Zhang and Thomas G Dietterich. A reinforcement learning approach to job-shop scheduling. In *IJCAI*, volume 95, pages 1114–1120. Citeseer, 1995.

[238] Y. Zhang, J. Wang, Z. Zhang, and Y. Zhou. Modrl/d-el: Multiobjective deep reinforcement learning with evolutionary learning for multiobjective optimization. In *Proceedings of Joint Conference on Neural Networks*, pages 1–8, 2021.

[239] Yizhou Zhang, Lun Du, Defu Cao, Qiang Fu, and Yan Liu. Prompting with divide-and-conquer program makes large language models discerning to hallucination and deception, 2024.

[240] Z. Zhang, Z. Wu, H. Zhang, and J. Wang. Meta-learning-based deep reinforcement learning for multiobjective optimization problems. *IEEE Transactions on Neural Networks and Learning Systems*, 34(10):7978–7991, 2023.

[241] Z. Zhang, Z. Zhang, X. Wang, and W. Zhu. Learning to solve travelling salesman problem with hardness-adaptive curriculum. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 9136–9144, 2022.

[242] Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. Automatic chain of thought prompting in large language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[243] H. Zhao and K. Xu. Learning efficient online 3d bin packing on packing configuration trees. In *Proceedings of International Conference on Learning Representations*, pages 1–18, 2022.

[244] J. Zheng, K. He, J. Zhou, Y. Jin, and C.-M. Li. Combining reinforcement learning with lin-kernighan-helsgaun algorithm for the traveling salesman problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, number 1, pages 12445–12452, 2021.

[245] J. Zheng, K. He, J. Zhou, Y. Jin, and C.-M. Li. Reinforced lin–kernighan–helsgaun algorithms for the traveling salesman problems. *Knowledge-Based Systems*, 260:110144, 2023.

[246] Shuai Zheng, Chetan Gupta, and Susumu Serita. Manufacturing dispatching using reinforcement and transfer learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 655–671. Springer, 2019.

[247] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

[248] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[249] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 11106–11115, 2021.

[250] J. Zhou, Y. Wu, W. Song, Z. Cao, and J. Zhang. Towards omnigeneralizable neural methods for vehicle routing problems. In *Proceedings of International Conference on Machine Learning*, pages 42769–42789, 2023.

[251] Jiayu Zhou, Zongzhang Cao, Yaru Wu, Weiping Song, Yaoxin Ma, Jianye Zhang, and Chongjie Xu. Mvmoe: Multi-task vehicle routing solver with mixture-of-experts. *arXiv preprint arXiv:2405.01029*, 2024.

[252] Z. Zong, X. Tong, M. Zheng, and Y. Li. Reinforcement learning for solving multiple vehicle routing problem with time window. *ACM Transactions on Intelligent Systems and Technology*, 15(2):1–19, 2024.

[253] Z. Zong, M. Zheng, Y. Li, and D. Jin. Mapdp: Cooperative multi-agent reinforcement learning to solve pickup and delivery problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 9980–9988, 2022.