

THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

Neural Decomposition Strategies for Multigraph Vehicle Routing Problems

FILIP RYDIN

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden, 2026

Neural Decomposition Strategies for Multigraph Vehicle Routing Problems

FILIP RYDIN

Acknowledgements, dedications, and similar personal statements in this thesis reflect the author's own views.

© FILIP RYDIN 2026 except where otherwise stated.

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
Phone: +46 (0)31 772 1000

Cover:

A multigraph VRP instance, solved with a neural method to produce three truck routes.

Note:

Large language models were used solely as assistive tools for grammar checking and language refinement. The content and the analysis in the thesis are the author's own work.

Printed by Chalmers Digital Printing
Gothenburg, Sweden, October 2026

Neural Decomposition Strategies for Multigraph Vehicle Routing Problems

FILIP RYDIN

Department of Electrical Engineering
Chalmers University of Technology

Abstract

Vehicle Routing Problems (VRPs) are a canonical class of combinatorial optimization problems. In recent years, end-to-end machine learning has shown great promise for this class. Besides often achieving competitive solution quality, learning-based methods offer advantages such as amortized inference, scalability and adaptation across problem variants without extensive tuning.

Yet, progress has been confined to VRPs defined on simple graphs, with a single edge between each node pair. In many practically relevant scenarios, routing takes place on *multigraphs*, where many parallel edges with distinct attributes connect the same node pair. Multigraphs capture effects that are important in practice, but they also pose substantial challenges for existing neural methods. Beyond the node order, the routing policy must also decide which parallel edge to traverse, creating a substantially larger action space as instance size and edge multiplicity increase.

This thesis proposes three Graph Neural Network (GNN)-based methods for routing on multigraphs, each trained with reinforcement learning. The methods span a spectrum from operating directly on the dense multigraph to decomposing the policy by separating the choice of node permutation from the choice of connecting edges, improving computational scalability. First, GMS-EB (GNN-based Multigraph Solver, Edge-Based) operates end-to-end on the graph edges, building routes autoregressively one edge at a time. Second, GMS-DH (Dual-Head) prunes the multigraph into a simple graph by retaining one edge per node pair and then selects nodes autoregressively. Third, NEPF (Node-Edge Policy Factorization) reverses the stage order by first fixing the node permutation and then selecting the connecting edges.

We evaluate the three methods on a wide suite of single- and multi-objective routing problems against classical (non-learned) and neural baselines. Across these settings we show that the proposed methods are efficient and effective, remaining competitive with strong classical solvers while retaining the amortization benefits of end-to-end learning. By exploring the design space of neural combinatorial optimization for multigraph routing, this work bridges the gap between learned routing methods and the realistic settings in which they typically must operate.

Keywords: Graph neural networks, vehicle routing problems, reinforcement learning, multi-objective optimization, multigraphs.

To everyone who supported me along the way.

List of Publications

This thesis is based on the following publications:

[A] **F. Rydin**, A. Lischka, J. Wu, M. Haghiri Chehreghani, B. Kulcsár, “Beyond Simple Graphs: Neural Multi-Objective Routing on Multigraphs”. *14th International Conference on Learning Representations (ICLR)*, Rio de Janeiro, Brazil, Apr. 2026.

[B] **F. Rydin**, M. Haghiri Chehreghani, B. Kulcsár, “Two-Stage Learned Decomposition for Scalable Routing on Multigraphs”. Submitted for publication as conference paper, 2026.

Other publications by the author, not included in this thesis, are:

[C] A. Lischka, **F. Rydin**, J. Wu, M. Haghiri Chehreghani, B. Kulcsár, “A GREAT Architecture for Edge-Based Graph Problems Like TSP”. Submitted for publication as journal article, 2024.

[D] K. Bågmark, A. Andersson, S. Larsson, **F. Rydin**, “A Convergent Scheme for the Bayesian Filtering Problem Based on the Fokker–Planck Equation and Deep Splitting”. Submitted for publication as journal article, 2024.

[E] K. Bågmark, **F. Rydin**, “High-Dimensional Bayesian Filtering through Deep Density Approximation”. Submitted for publication as journal article, 2025.

[F] K. Bågmark, **F. Rydin**, “Neural Likelihood Surrogates for Parameter Inference via Log-Density PDE”. *ICLR Workshop on AI and Partial Differential Equations*, Rio de Janeiro, Brazil, Apr. 2026.

Contents

Abstract	ii
List of Publications	vii
Acknowledgements	xiii
Acronyms	xiv
I Overview	1
1 Introduction	3
1.1 Research Questions	6
1.2 Contributions	7
1.3 Thesis Outline	7
2 Combinatorial Optimization and Routing	9
2.1 Combinatorial Optimization Problems	9
Complexity Theory	10
2.2 Vehicle Routing Problems	11
2.3 Classical VRP Solution Strategies	12
Nearest Neighbor Algorithm	13

	Mixed-Integer Linear Programming	13
	Heuristics and Metaheuristics	14
2.4	Multi-Objective Optimization	15
	Solution Strategies	17
	Performance Measurement	18
2.5	Graph Representations for Routing	18
3	Models and Learning Algorithms	23
3.1	Neural Architectures	24
	Attention and Transformers	24
	Graph Neural Networks	27
	Recurrent Neural Networks and Sequence Models	29
3.2	Learning Algorithms	29
	Supervised Learning	30
	Reinforcement Learning	30
	Multi-Objective Reinforcement Learning	32
4	Neural Combinatorial Optimization	35
4.1	Taxonomy of Methodologies	35
4.2	Autoregressive Construction	36
4.3	Non-Autoregressive Construction	40
4.4	Improvement-Based Approaches	44
4.5	Other Learning-Based Strategies	48
4.6	Research Directions and Challenges	49
	Asymmetric Routing Problems	49
	Hard-Constrained Problems	50
	Generalization to Large Instances	51
	Cross-Distribution Generalization	52
	Cross-Problem Generalization	52
	Multi-Objective Routing	54
4.7	Limitations of Prior Methods	55
	Encoding	55
	Decoding	56
	Gap Summary	58
5	Learning to Route on Multigraphs: Challenges and Methodology	61
5.1	Challenges of Multigraph Routing	61

5.2	Design Space and Methodology	63
	End-to-End Edge-Based AR Construction: GMS-EB	65
	NAR Pruning then AR Routing: GMS-DH	66
	AR Routing then NAR Edge Selection: NEPF	67
6	Summary of Included Papers	71
6.1	Paper A	71
6.2	Paper B	73
7	Concluding Remarks and Future Work	77
7.1	Conclusion	77
7.2	Limitations and Future Work	78
	References	81
II	Papers	93
A	Beyond Simple Graphs: Neural Multi-Objective Routing on Multi-graphs	A1
1	Introduction	A3
2	Related Work	A5
2.1	Learning for Routing	A5
2.2	Routing on Multigraphs	A5
3	Problem Formulation	A6
4	GNN-Based Multigraph Solver	A7
4.1	Edge-Based GMS	A8
4.2	Dual Head GMS	A9
4.3	Training Algorithm	A12
5	Experimental Studies	A13
5.1	Performance on Benchmark Problems	A16
5.2	Zero-Shot Generalization	A19
5.3	A Difficult Multigraph Problem	A19
6	Conclusion	A20
	References	A22

B	Two-Stage Learned Decomposition for Scalable Routing on Multi-graphs	B1
1	Introduction	B3
2	Related Work	B5
3	Problem Formulation and Notation	B6
4	Node-Edge Policy Factorization (NEPF) for Multigraph VRPs	B7
4.1	Node Permutation Stage	B7
4.2	Fixed Sequence Arc Selection Problem Stage	B10
4.3	Hierarchical Training Setup	B12
4.4	Multi-Objective Preference Conditioning	B13
5	Experimental Results	B13
5.1	Performance Evaluation	B14
5.2	Scalability Evaluation	B17
5.3	Cross-Backbone Results	B19
5.4	Ablation Study	B20
6	Conclusion	B21
	References	B22

Acknowledgements

First and foremost, my deepest gratitude goes to my supervisors Balázs and Morteza, who have not only guided me scientifically, but also acted as exceptional mentors. I could not have wished for kinder, more patient, more curious or more steadfast supervisors. I am equally grateful to my former co-supervisor Jiaming, who provided invaluable support whenever it was needed during my first two years as a PhD student. My examiner Torsten also deserves a heartfelt mention for welcoming me into the group and creating such an enjoyable research environment.

Special dedications go to my office mates Isac and Ahmet, my “PhD brothers” Hannes and Attila, and my co-author and former thesis supervisor Kasper. I am truly grateful for how all of you have helped shape my PhD journey up until this point. When I think back on the last two and a half years, I realize there are so many more people to name. Thank you Albert, Gabriel, Lorenzo, Kilian, Adam, Alvin, Sondre, Daniel (for helping raise my MDE score), Lasse, Kristian, Faris, Sten Elling, Erik Börve, Sabino, Mattia, Godwin, Erik Brorsson, Francesco Gigante, Francesco Popolizio, Martina, Nanami, Zhitao, Ying, Chiara, Michela, Carl-Johan, Rikard, Kartik, Mohamed, Amal, Arvid, Xiaolei, Ilya, Orlando, Nishant, Ben, André, Fangting, Xiaolingzi and all my other friends and colleagues for making the department such a great place to work.

Last but definitely not least, this degree would not have been possible without the family and friends who have supported me along the way. Know that I am deeply grateful to all of you.

This work was supported by the Swedish Electromobility Centre (SEC) through the research project “LEAR: Robust LEArning methods for electric vehicle Route selection”. The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) at Chalmers e-Commons, partially funded by the Swedish Research Council through grant agreement no. 2022-06725. The work was also partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. I thank all these organizations for making this work possible.

Acronyms

GNN:	Graph Neural Network
GCN:	Graph Convolutional Network
GREAT:	Graph Edge Attention
MLP:	Multilayer Perceptron
RNN:	Recurrent Neural Network
LSTM:	Long Short-Term Memory
MP:	Multi-Pointer
MHA:	Multi-Head Attention
RL:	Reinforcement Learning
LLM:	Large Language Model
MDP:	Markov Decision Process
MOMDP:	Multi-Objective MDP
TSP:	Traveling Salesman Problem
VRP:	Vehicle Routing Problem
CVRP:	Capacitated VRP
COP:	Combinatorial Optimization Problem
MOOP:	Multi-Objective Optimization Problem
MILP:	Mixed-Integer Linear Program
HV:	Hypervolume
NCO:	Neural Combinatorial Optimization
AR:	Autoregressive

NAR:	Non-Autoregressive
AM:	Attention Model
POMO:	Policy Optimization with Multiple Optima
GMS:	GNN-based Multigraph Solver
GMS-EB:	Edge-Based GMS
GMS-DH:	Dual-Head GMS
NEPF:	Node-Edge Policy Factorization
FSASP:	Fixed Sequence Arc Selection Problem

Part I

Overview

CHAPTER 1

Introduction

Modern logistics systems are becoming increasingly complex. The continued rise of e-commerce places growing volumes of goods in motion. In Sweden, the total parcel market reached 368 million parcels in 2024, with domestic volumes having grown by 58% since 2018 [1]. This growth is driven partly by online retail, but also by urbanization, the fragmentation of orders into smaller and more frequent shipments and the spread of delivery-centric business models such as online grocery and meal delivery. At the same time, customers have come to expect same-day delivery, mobility on demand and similarly stringent levels of service, which concentrates pressure on the most expensive and least efficient segment of the supply chain: the last mile, which can account for over half of total delivery cost [2]. Meeting rising demand at high levels of service and at acceptable cost places a growing burden on the systems that plan, coordinate and optimize delivery operations.

Further, the transition to sustainable transport introduces entirely new decisions. Road transport is a major source of greenhouse-gas emissions. In fact, the transport sector accounts for approximately a fifth of global CO₂ emissions, of which freight trucks make up close to a third [3]. Lowering this footprint requires planning that explicitly accounts for emissions. Such

planning ranges from green routing, in which the emissions of a route are optimized alongside its cost or duration, to the joint scheduling of charging and driving for electric fleets, where limited range and charging times become routing constraints.

Each of these trends adds moving parts to an already intricate system. The burden ultimately falls on the planning and optimization algorithms that operate the system. Ideally, such algorithms would handle very large instances, solve them to or near optimality and accommodate new constraints without excessive re-tuning. This is a demanding requirement, especially since the underlying optimization problems are typically combinatorial and often $\mathcal{NP}$ -hard [4].

Traditionally, logistics and delivery problems of this kind have been modeled as Vehicle Routing Problems (VRPs), a canonical class of combinatorial optimization problems studied intensively since their introduction over six decades ago [5, 6], owing both to their practical usefulness and their inherent difficulty. Classical VRP solvers, developed over decades of operations-research effort, are often effective on the settings they target, yet they fall short of the ideal above in several respects. First, many solvers are hand-crafted for a specific problem variant, so that incorporating a new constraint or objective requires considerable expert effort. Second, the strongest exact methods scale poorly with problem size and even state-of-the-art heuristics tend to require careful tuning to be efficient.

Recently, learning-based, or neural, solvers have emerged as an alternative. Rather than designing a heuristic by hand, one trains a neural network to construct solutions, typically with reinforcement learning, so that the resulting policy generalizes across instances of a problem and produces solutions rapidly at inference time. This approach is inspired by the success of end-to-end machine learning in domains such as Natural Language Processing (NLP) and Computer Vision (CV), where hand-crafted features and heuristics have largely become obsolete in frontier systems [7, 8]. A successful neural combinatorial optimization model amortizes the cost of solving over an entire distribution of instances, adapting to new problems through training instead of manual redesign. Since the introduction of pointer networks [9] and their reinforcement-learning-based training [10], a large body of work, including attention-based construction models [11] and symmetry-exploiting schemes such as POMO [12], has shown remarkable success in solving a range of VRPs efficiently and

effectively [13]. Notably, cross-problem foundation models have also begun to emerge [14, 15, 16]. These solve multiple VRP variants with a single model without retraining, mirroring the foundation models that dominate NLP and CV [17].

Until now, however, progress has largely been confined to idealized settings with arguably limited practical relevance. In particular, prevailing neural architectures derive node embeddings from two-dimensional coordinates and thereby assume simple, symmetric, Euclidean instances in which a single edge connects each pair of locations. Real transportation networks rarely satisfy this assumption [18]. Between two locations there is seldom one canonical connection, but rather several qualitatively different ones, for instance a longer motorway that is fast against a shorter urban route that is slow. Whether such alternatives matter depends on the problem. In a single-objective, unconstrained setting, one may simply precompute the shortest connection for each pair of locations and discard the rest, as exactly one path per pair can be optimal. However, if multiple objectives or side constraints are introduced, this reduction no longer holds. For example, the fastest route need not be the cheapest or the least polluting and a longer edge may be the only one that respects a time window, so that no single connection dominates [19, 20, 21].

Such routing problems are naturally posed on *multigraphs*, in which multiple parallel edges, each with their own attributes, connect the same pair of nodes. The multigraph representation enables the path between two locations to be chosen jointly with the overall routing decisions, rather than fixed in advance. This joint optimization can save considerable cost, time or energy in practice [19]. Consider an everyday example: when driving to a destination, a navigation service typically offers several routes, e.g., a faster motorway or a shorter route through the city and the best choice depends on the rest of the journey. Making such choices well across an entire tour and for every pair of stops at once is the goal of multigraph routing.

At the same time, the representation introduces difficulties. A multigraph route is no longer uniquely specified by an ordering of the nodes alone, but additionally by the choice of which parallel edge to traverse, which breaks the assumptions on which most existing neural solvers rely. Moreover, the natural edge-based representation of a multigraph has size $\mathcal{O}(MN^2)$, where N is the number of nodes and M the number of parallel edges per node pair, and quickly becomes a computational bottleneck as graphs grow. In contrast,

node-based representations of size $\mathcal{O}(N)$, common in Euclidean routing, are far easier to scale. Adapting existing architectures and methods to multigraphs, while retaining their efficiency and effectiveness, is thus a challenging task.

Consequently, in this thesis we explore neural methods for VRPs on multigraphs. The aim is to develop learning-based solvers that route on multigraphs directly, handle the parallel-edge structure rather than discarding it and do so at a scale relevant to practical applications. More broadly, this work is part of a wider effort to bring neural routing closer to practice. Concurrent and parallel work addresses hard-constrained [22, 23], asymmetric [24, 25], time-dependent [26] and otherwise realistic [27] settings.

1.1 Research Questions

Essentially all learning-based vehicle routing solvers assume simple graphs, in which a single edge connects each pair of nodes. Yet, many practically relevant routing problems are naturally posed on multigraphs, where several parallel edges with distinct attributes connect the same node pair. Prior neural methods are ill-suited to this setting. Based on this gap in the literature, this thesis aims to answer the following research questions:

- RQ1. Feasibility and quality.** Can learning-based construction solvers be designed to route directly on multigraphs, jointly deciding the node visiting order and which parallel edge to traverse, and attain solution quality competitive with established operations-research baselines in single-objective and multi-objective settings?
- RQ2. Scalability.** The explicit multigraph representation has size $\mathcal{O}(MN^2)$, making end-to-end edge-based construction computationally expensive. Can the routing policy be decomposed into separate node permutation and edge selection stages so that neural multigraph routing scales more favorably in both the node count N and the number of parallel edges M , while retaining the solution quality of a direct approach?
- RQ3. Design.** How should the routing and edge selection stages be organized? That is, which part should be solved first and where should autoregressive versus non-autoregressive components be placed? The design should maximize computational efficiency, while yielding expressive enough models that can generalize well both in- and out-of-distribution.

1.2 Contributions

The thesis addresses these questions through three neural methods for routing on multigraphs, presented across the two appended papers. Paper A introduces *GNN-based Multigraph Solver* (GMS) in two variants. *Edge-Based GMS* (GMS-EB) constructs routes directly on the multigraph and thereby establishes the feasibility and effectiveness of learning-based multigraph routing, competitive with classical and evolutionary baselines, addressing RQ1. *Dual-Head GMS* (GMS-DH) then decomposes the policy into a non-autoregressive pruning stage followed by autoregressive routing, taking a first step towards RQ2. Paper B contributes *Node-Edge Policy Factorization* (NEPF), which reverses this order, routing over nodes before selecting edges non-autoregressively, and collapses the parallel edges before routing. NEPF scales more favorably in N and M than the prior methods and is the strongest neural method in most of our experiments, addressing RQ2 and RQ3.

1.3 Thesis Outline

The thesis is divided into two parts. Part I: Overview provides background, a review of related work and a summary of the methodology, while Part II: Papers contains the full Papers A and B. The remainder of Part I is organized as follows. First, the theoretical background of single-objective and multi-objective combinatorial optimization is examined in Chapter 2, with a particular emphasis on vehicle routing problems. Then, the necessary machine learning background is provided in Chapter 3, while existing learning-based methods for combinatorial optimization are reviewed in Chapter 4. This is followed by a presentation of our proposed approaches for multigraph routing in Chapter 5. Chapter 6 then summarizes the included papers and Chapter 7 concludes.

CHAPTER 2

Combinatorial Optimization and Routing

This chapter reviews necessary historical and theoretical background regarding the problems investigated in this thesis as well as their classical solution strategies. It begins by examining general combinatorial optimization problems, before focusing on vehicle routing problems in particular. Multi-objective extensions, which both appended papers address, are then treated. Finally, we discuss graph representations for routing, bridging the gap between this chapter and the rest of the thesis.

2.1 Combinatorial Optimization Problems

Consider a general optimization problem

$$\min_{x \in X} f(x), \tag{2.1}$$

with feasible set X and objective function $f : X \rightarrow \mathbb{R}$. If the feasible set X is discrete and grows combinatorially with the problem size, the optimization problem is referred to as a Combinatorial Optimization Problem (COP). Hence, while it is technically possible to find the optimal solution through exhaustive search, it is infeasible in practice for problems of sufficient size.

As an example, consider finding the optimal permutation of an array $x = (x_1, x_2, \dots, x_N)$ with N elements. As there are $N!$ such permutations, brute-force search becomes infeasible quickly as N grows. Instead, smarter strategies, often based on decomposition, approximation or problem-specific structure, must be employed.

During the last century, combinatorial optimization has grown into a major subfield of mathematics [4]. In part, this is motivated by the vast number of areas these problems surface in. Examples include manufacturing, scheduling, networking and communications, robotics and control, as well as logistics and delivery. It is important to emphasize, however, that not all combinatorial optimization problems are equally difficult, which can be understood through *complexity theory*.

Complexity Theory

Complexity theory studies the *asymptotic* complexity of algorithms, i.e., how their computational requirements, e.g., runtime or memory, scale with problem size [28]. This scaling is expressed in big- $\mathcal{O}$ notation, where, for non-negative functions g and h , $g(N) = \mathcal{O}(h(N))$ means that there exist constants $c > 0$ and N_0 such that $g(N) \leq c h(N)$ for all $N \geq N_0$. That is, g is asymptotically bounded above by h , up to a constant factor. In the following, we consider *decision problems*, where the objective is to determine whether a given instance satisfies a specified property.

Decision problems that can be solved in polynomial time by a deterministic algorithm belong to the class $\mathcal{P}$. Their runtime scales as $\mathcal{O}(N^k)$ for some constant k , where N denotes the input size. Although polynomial-time algorithms are often considered tractable, the exponent k may be sufficiently large to make them impractical.

The class $\mathcal{NP}$ consists of decision problems for which a proposed solution can be verified in polynomial time by a deterministic algorithm. Equivalently, $\mathcal{NP}$ is the class of problems solvable in polynomial time by a nondeterministic Turing machine, an abstract computational model that can be viewed as exploring many possible solution paths simultaneously and accepting if at least one path leads to a solution. Clearly, $\mathcal{P} \subseteq \mathcal{NP}$, but whether $\mathcal{P} = \mathcal{NP}$ remains a major open question in computer science.

Within $\mathcal{NP}$, the class of $\mathcal{NP}$ -complete problems is of particular interest [28]. These problems are the hardest in $\mathcal{NP}$, in the sense that every problem in $\mathcal{NP}$

can be reduced to an $\mathcal{NP}$ -complete problem in polynomial time. A canonical example is the Boolean satisfiability problem (SAT), which asks whether there exists an assignment of truth values to variables such that a Boolean formula evaluates to true. SAT was the first problem proven to be $\mathcal{NP}$ -complete [29]. Consequently, if any $\mathcal{NP}$ -complete problem can be solved in polynomial time, then $\mathcal{P} = \mathcal{NP}$.

For this thesis, the primary class of interest is $\mathcal{NP}$ -hard problems. These problems are at least as difficult as $\mathcal{NP}$ -complete problems, but are not necessarily members of $\mathcal{NP}$. In particular, they need not be decision problems and may not admit polynomial-time verification. Many combinatorial optimization problems fall into this category. **Unless $\mathcal{P} = \mathcal{NP}$, polynomial-time exact algorithms are not expected for $\mathcal{NP}$ -hard optimization problems** whose decision versions are $\mathcal{NP}$ -complete. Instead, to solve these problems efficiently, heuristic algorithms without formal guarantees are often preferred.

While many of the optimization problems studied in this thesis are $\mathcal{NP}$ -hard, there are combinatorial optimization problems which are solvable in polynomial time. For example, the standard shortest path problem can be solved in polynomial time using Dijkstra's algorithm.

2.2 Vehicle Routing Problems

Vehicle routing problems constitute a class of some of the most extensively studied combinatorial optimization problems. This can partly be attributed to their practical importance in logistics, delivery and transportation planning and partly to their computational difficulty [6]. While the original VRP is commonly attributed to Dantzig and Ramser in 1959 [5], the field continues to attract substantial research interest.

In general, VRPs can be viewed as generalizations of the Traveling Salesman Problem (TSP), arguably the most well-known $\mathcal{NP}$ -hard combinatorial optimization problem. In the TSP, a set of cities must be visited exactly once, with pairwise travel costs given between cities. The objective is to determine the visitation order that minimizes the total travel distance while returning to the starting city. Despite its relatively simple formulation, the TSP remains one of the most intensively studied optimization problems.

A canonical VRP variant is the Capacitated Vehicle Routing Problem (CVRP). In the CVRP, a fleet of vehicles departs from a depot to service a set

of customers, each with a specified demand. Each vehicle has a fixed carrying capacity and every customer must be visited exactly once. The objective is to determine routes for the vehicles that minimize total travel distance while respecting vehicle capacity constraints.

Beyond the TSP and CVRP, numerous VRP variants have been proposed to model realistic transportation scenarios. Some examples are presented in Table 2.1. For this thesis, the ACVRP, VRPTW and OP are of particular interest, as variants of these problems are considered in Papers A and B.

Table 2.1: Common vehicle routing problem variants.

Abbreviation	Full Name	Explanation
ACVRP	Asymmetric CVRP	The travel cost from customer i to j differs from the cost from j to i .
VRPTW	VRP with Time Windows	Each customer must be served within a specified time interval.
OP	Orienteering Problem	The vehicle collects prizes at customers. Total collected prizes are maximized while total distance must remain below a threshold.
MDVRP	Multi-Depot VRP	Vehicles operate from multiple depots instead of a single central depot.
DVRP	Dynamic VRP	Customer requests or travel information may change during route execution.
PDP	Pickup and Delivery Problem	Goods are picked up and delivered between paired stops with precedence constraints.
HFVRP	Heterogeneous Fleet VRP	Vehicles differ in capacity, cost or type.
EVRP	Electric VRP	Routing for electric vehicles considering battery limitations and charging stations.
SVRP	Stochastic VRP	Includes uncertainty in travel times, demands or customer availability.

2.3 Classical VRP Solution Strategies

To enable efficient solution of VRPs, researchers have proposed a multitude of solution strategies, including exact methods with optimality guarantees and heuristic methods. This section reviews some of the most important algorithms.

Nearest Neighbor Algorithm

Arguably one of the simplest and most intuitive approaches for tackling VRPs is the nearest neighbor algorithm. Consider the TSP. In this setting, the nearest neighbor heuristic constructs a tour greedily by repeatedly selecting the closest unvisited city as the next visit, starting from an arbitrary initial city.

This algorithm is easily scalable, as it can be implemented with a time complexity of $\mathcal{O}(N^2)$ for N cities. However, it does not, in general, produce optimal solutions. For metric TSP instances, i.e., when distances are non-negative, symmetric and satisfy the triangle inequality, the nearest neighbor heuristic has a worst-case approximation ratio (the supremum of the ratio of its tour length to the optimal tour length, over all instances of a given size) upper bounded by $\frac{1}{2} \lceil \log_2 N \rceil + \frac{1}{2}$ [30]. This ratio grows without bound in N , meaning the heuristic provides no constant-factor guarantee.

Mixed-Integer Linear Programming

By formulating VRPs as Mixed-Integer Linear Programs (MILPs), researchers and practitioners can leverage general-purpose optimization algorithms and theory to analyze and solve such problems. As an example, consider the TSP. While alternative formulations exist, a classical MILP formulation is the Miller–Tucker–Zemlin (MTZ) formulation [31].

Let $x_{ij} \in \{0, 1\}$ for $i, j = 1, \dots, N$, $i \neq j$, denote binary decision variables indicating whether the tour includes the arc from city i to city j and let c_{ij} denote the travel cost between i and j . Furthermore, let u_i be auxiliary variables used to eliminate subtours. The MTZ formulation of the TSP is given by:

$$\min_{x, u} \quad \sum_{i=1}^N \sum_{\substack{j=1 \\ j \neq i}}^N c_{ij} x_{ij}, \quad (2.2)$$

$$\text{s.t.} \quad \sum_{\substack{j=1 \\ j \neq i}}^N x_{ij} = 1, \quad \forall i \in \{1, \dots, N\}, \quad (2.3)$$

$$\sum_{\substack{i=1 \\ i \neq j}}^N x_{ij} = 1, \quad \forall j \in \{1, \dots, N\}, \quad (2.4)$$

$$u_i - u_j + Nx_{ij} \leq N - 1, \quad \forall i \neq j, i, j \in \{2, \dots, N\}, \quad (2.5)$$

$$1 \leq u_i \leq N - 1, \quad \forall i \in \{2, \dots, N\}. \quad (2.6)$$

Here, constraints (2.3) and (2.4) ensure that each city is entered and left exactly once. The MTZ constraints (2.5), together with the ordering variables u_i , prevent subtours by enforcing a consistent ordering of visited nodes.

After MILP formulations have been obtained, general-purpose solvers such as Gurobi [32] or CPLEX [33] can be used to solve the resulting models. Alternatively, specialized decomposition methods can be developed for improved scalability. An important example is branch-and-price, which combines branch-and-bound with column generation on a set-partitioning reformulation and underlies many exact algorithms for the large-scale CVRP. Branch-and-cut, which strengthens the MILP relaxation with problem-specific valid inequalities, is similarly central to the literature [6].

Heuristics and Metaheuristics

While MILP formulations enable exact optimization of VRPs, solving large-scale instances to proven optimality is often computationally prohibitive due to the combinatorial growth of the solution space. As such, heuristics and metaheuristics are often employed to obtain solutions more efficiently.

Unlike exact optimization methods, heuristics do not guarantee globally optimal solutions. Instead, they exploit problem structure to efficiently search for satisfactory solutions. Metaheuristics extend this idea by providing general algorithmic frameworks for guiding the search process, often incorporating stochastic elements to balance *exploration* of new regions in the solution space with *exploitation* of promising candidate solutions.

Metaheuristics have proven particularly successful for VRPs and other combinatorial optimization problems due to their scalability and flexibility. Many metaheuristics can be adapted to accommodate complex constraints and realistic problem features without requiring major reformulations. Examples of widely used metaheuristics include simulated annealing [34], tabu search [35], genetic algorithms [36], ant colony optimization [37] and large neighborhood

search [38, 39].

These metaheuristics have been adapted and tuned to create powerful problem-specific heuristics. An important example is the Lin–Kernighan–Helsgaun (LKH) solver for the TSP [40], which has been shown to recover the best-known solutions for a wide range of benchmark instances. The method relies on variable-depth k -opt neighborhood moves to iteratively refine segments of the tour. A k -opt move replaces k edges in a tour with alternative connections to improve solution quality. Similarly, for the CVRP, Hybrid Genetic Search (HGS) [41, 42] is a state-of-the-art heuristic that often recovers optimal or best-known solutions within practical computation times. In this thesis, LKH and HGS are important baselines for the computational experiments in Papers A and B.

2.4 Multi-Objective Optimization

The VRP variants and solution strategies discussed so far target a single objective, typically total travel distance or cost. In practice, however, decision-makers in routing and logistics often face several conflicting criteria, e.g., distance, travel time, emissions or risk, motivating a multi-objective treatment.

Multi-objective optimization generalizes the concept of standard single-objective optimization. In general, a Multi-Objective Optimization Problem (MOOP) may be written as

$$\min_{x \in X} f(x) := (f_1(x), f_2(x), \dots, f_m(x)), \quad (2.7)$$

where $f : X \rightarrow \mathbb{R}^m$ is a vector-valued objective function and m denotes the number of objectives.

Since the objectives are typically conflicting, no single solution can minimize all of them simultaneously. Hence, the concept of minimization must be specified further. The standard definition of a solution concept for (2.7) is the *Pareto set* (although alternative solution concepts exist [43]), which builds on the *Pareto dominance* relation. A vector $y \in \mathbb{R}^m$ is said to dominate another vector $y' \in \mathbb{R}^m$, denoted by $y \prec y'$, if

$$y_i \leq y'_i, \quad \forall i = 1, \dots, m, \quad (2.8)$$

$$y_i < y'_i, \quad \text{for at least one } i. \quad (2.9)$$

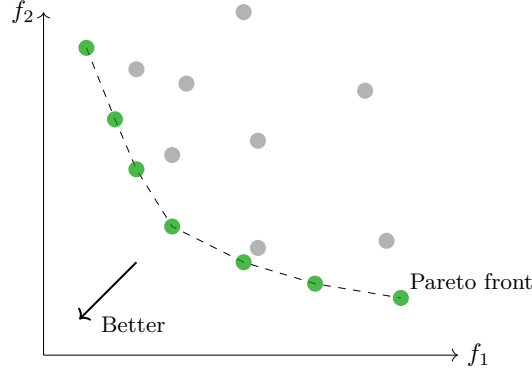


Figure 2.1: A typical bi-objective Pareto front. Dominated solutions are in gray while non-dominated solutions are in green.

Correspondingly, a solution $x \in X$ is said to dominate another solution $x' \in X$, denoted by $x \prec x'$, if $f(x) \prec f(x')$. The Pareto set is then defined as

$$\text{PS} := \{x \in X \mid \nexists x' \in X : x' \prec x\}. \quad (2.10)$$

That is, the Pareto set is the set of all solutions that are not dominated by any other solution. The image of the Pareto set in the objective space is the *Pareto front*:

$$\text{PF} := \{f(x) \in \mathbb{R}^m \mid x \in \text{PS}\}. \quad (2.11)$$

A typical bi-objective case is shown in Figure 2.1, where it can be seen that solutions in the Pareto set provide different trade-offs between the objectives and that it is not possible to improve one objective without degrading the other.

A key motivation for multi-objective optimization is decision support [44]. Rather than committing to a single predefined trade-off between objectives, decision-makers are provided with a set of Pareto-optimal alternatives, which enables preferences to be incorporated at an appropriate organizational or operational level.

Thus, multi-objective optimization is highly relevant in many practical applications. However, MOOPs are often considerably more challenging than their single-objective counterparts. In addition to finding feasible solutions, one must identify a potentially large set of trade-off solutions. This added complexity can fundamentally alter the computational properties of a problem. For example, while the single-objective shortest path problem is solvable in

polynomial time, even its bi-objective variant, with two conflicting path costs, is intractable [45]. Furthermore, the number of Pareto-optimal solutions may grow exponentially with problem size for discrete problems [45, 46]. Consequently, a broad range of exact and heuristic methods have been developed to tackle MOOPs. Some representative approaches are discussed next.

Solution Strategies

There exist various methods to obtain the Pareto set (also known as the non-dominated set) for MOOPs. These can be divided in various ways, but a common distinction is *criterion-space methods* and *decision-space methods*. This distinction separates methods based on how they conduct the search, where the former category searches by partitioning or working in the objective (criterion) space and the latter category searches in the solution (decision) space.

In the decision-space category, an important class of methods are Multi-Objective Evolutionary Algorithms (MOEAs). These typically maintain a *population* of solutions. The goal of the algorithm is to evolve the population, so that it converges to the true Pareto set, while maintaining diversity in the objective space. To this end, genetic operators such as *mutation* and *cross-over* are applied to the population. Widely used evolutionary algorithms include MOEA/D [47], NSGA-II [48] and MOGLS [49].

In the criterion-space category, scalarization techniques are prominent. In general, these rely on transforming the MOOP into several single-objective subproblems, solved sequentially or in parallel. The simplest approach is *linear scalarization*, where subproblems are formed by setting the scalar cost to a linear combination of the multi-objective costs. Since linear scalarization has several disadvantages, a multitude of alternatives exist [50]. A prominent example, used by the methods in this thesis, is *Chebyshev scalarization*.

Both linear and Chebyshev scalarization are parameterized by a *preference vector* $\lambda \in \mathbb{R}^m$ with $\lambda_i \geq 0$ and $\sum_{i=1}^m \lambda_i = 1$, and replace (2.7) with scalar subproblems $\min_{x \in X} g_\lambda(f(x))$. Linear scalarization uses $g_\lambda(f(x)) = \sum_{i=1}^m \lambda_i f_i(x)$ while Chebyshev scalarization instead sets

$$g_\lambda(f(x)) = \max_{i=1, \dots, m} \lambda_i |f_i(x) - z_i^*|, \quad (2.12)$$

with an ideal point $z^* \in \mathbb{R}^m$. Since every Pareto-optimal solution minimizes (2.12) for some λ , Chebyshev scalarization can in principle recover the full

Pareto front [51]. Hence, this scalarization is the principle behind the preference-conditioned methods of Chapters 3–5.

Performance Measurement

Assessing performance in the multi-objective case is considerably more difficult than in the single-objective case. Given two approximate Pareto fronts produced by two methods, it is often not clear a priori which one is preferable, see e.g., Figure 2.2a. Consequently, a large number of performance indicators have been proposed, focusing on measuring aspects such as cardinality, convergence, distribution and spread [52].

In this thesis, we focus on the Hypervolume (HV) indicator [53], which is widely used in the literature since it captures both the convergence and diversity of the evaluated Pareto front [54]. Consider an approximate Pareto front $S = \{y_1, \dots, y_n\} \subset \mathbb{R}^m$ as well as a reference point $r \in \mathbb{R}^m$ such that $y_i \prec r, \forall i = 1, \dots, n$. Let Λ denote the Lebesgue measure in $\mathbb{R}^m$ and define $[y_i, r]$ as the axis-aligned hyperrectangle with corners in y_i and r . The hypervolume of S with respect to r is then the measure

$$\text{HV}(S, r) = \Lambda \left(\bigcup_{i=1}^n [y_i, r] \right). \quad (2.13)$$

Intuitively, this is the volume of the region dominated by at least one point in S . A visualization of the HV indicator is provided in Figure 2.2b. Note that the optimal, ground-truth Pareto front maximizes HV [55].

2.5 Graph Representations for Routing

Having reviewed VRP variants and their single- and multi-objective solution strategies, we next consider the underlying graph representations used to model routing instances.

Vehicle routing problems are inherently graph problems. For example, given a weighted graph $G = (V, E, c)$ with node set V , edge set E and edge cost function $c : E \rightarrow \mathbb{R}$, the TSP can be formulated as

$$\min_{\pi \in \Pi} \sum_{i=1}^{N-1} c(\pi_i, \pi_{i+1}) + c(\pi_N, \pi_1), \quad (2.14)$$

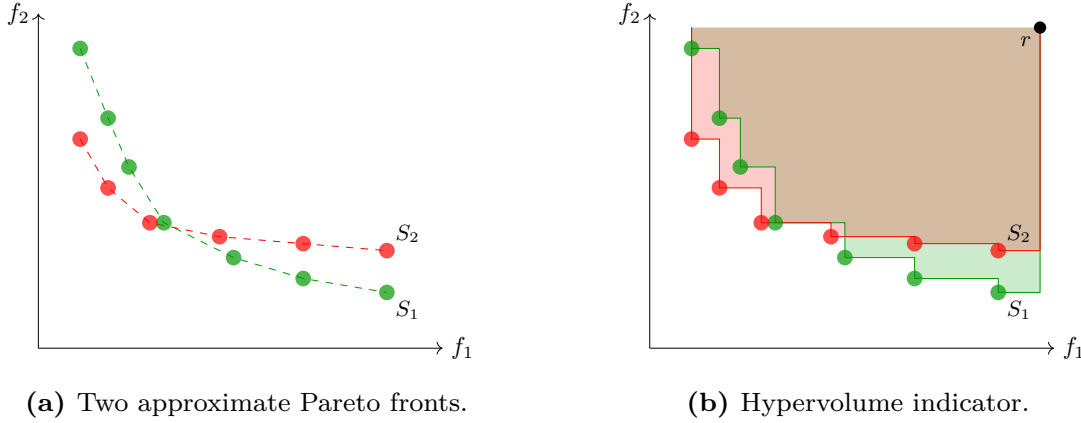


Figure 2.2: An example where it could be difficult to compare the performance of two algorithms, yielding S_1 and S_2 .

where $\pi = (\pi_1, \dots, \pi_N)$ denotes a permutation of the nodes and Π is the set of all such permutations. Although the underlying road network is generally sparse, routing problems are commonly represented on *complete* graphs whose edge costs correspond to shortest-path distances between locations. Hence, the instance graphs in routing problems are usually assumed to be complete (and without self-loops), yielding the edge set

$$E = \{(u, v) \in V^2 \mid u \neq v\}. \quad (2.15)$$

Beyond the assumption of complete graphs, routing problems are often categorized based on the type of instance graph. In *Euclidean routing problems*, pairwise distances are defined as the Euclidean distances given the node coordinates. This is often an idealized assumption in real networks, where vehicles cannot generally travel in straight paths between locations. More generally, metric routing problems assume only non-negative, symmetric costs satisfying the triangle inequality. Unfortunately, these assumptions are also not always satisfied. For example, one-way streets mean that travel distances might not be symmetric. Moreover, if the objective is travel time, traffic might induce asymmetric travel costs. On the other hand, if energy is to be optimized, edge costs may become negative due to regenerative braking or favorable elevation changes. In *asymmetric routing problems*, represented on directed graphs, asymmetric travel costs are allowed. Euclidean and asymmetric routing problems have been widely treated in both classical transportation literature and the learning-for-routing literature [9, 10, 11, 12, 24, 56, 57].

Further extensions are also prominent. *Time-dependent routing problems* allow edge costs to vary with time, whereas *dynamic routing problems* permit new information, such as customer requests, to arrive during execution. Similarly, *stochastic routing problems* allow non-deterministic travel times, introducing possible complexities by requiring solution robustness.

In this thesis, we are concerned with *multigraphs*, featuring multiple directed edges, typically with distinct attributes, between the same node pair. Formally, a directed multigraph G with nodes V and edges E has $N = |V|$ nodes, and each edge $l \in E$ has a start node $\text{start}(l) \in V$, an end node $\text{end}(l) \in V$ and an attribute vector e_l (for example its travel time and distance). Nodes may carry attributes n_u , $u \in V$ (such as customer demands). We write

$$E_{uv} := \{l \in E \mid \text{start}(l) = u, \text{end}(l) = v\}, \quad (2.16)$$

for the set of parallel edges from u to v and $M := \max_{u \neq v} |E_{uv}|$ for the number of parallel edges per node pair, so that $|E| = \mathcal{O}(MN^2)$. A simple graph is the special case $M = 1$.

Multigraphs offer a reasonable modeling choice whenever a single optimal path cannot be chosen a priori between locations, which typically occurs when there are several edge attributes of interest (for example both travel time and distance). Then, reducing each origin–destination pair to a single shortest path may discard relevant trade-offs. Instead, multiple non-dominated paths can be retained as parallel edges, yielding a multigraph representation. Some important graph types for this thesis are visualized in Figure 2.3.

Multigraph routing problems are widely treated in classical operations research literature [19, 20, 21, 58, 59, 60], but have received limited attention with end-to-end learning-based methods. In Chapter 4, the learning-for-routing literature is reviewed. At the end of the chapter we explain why most current methods are unsuitable to solve multigraph routing problems. First, however, the next chapter reviews necessary machine learning background.

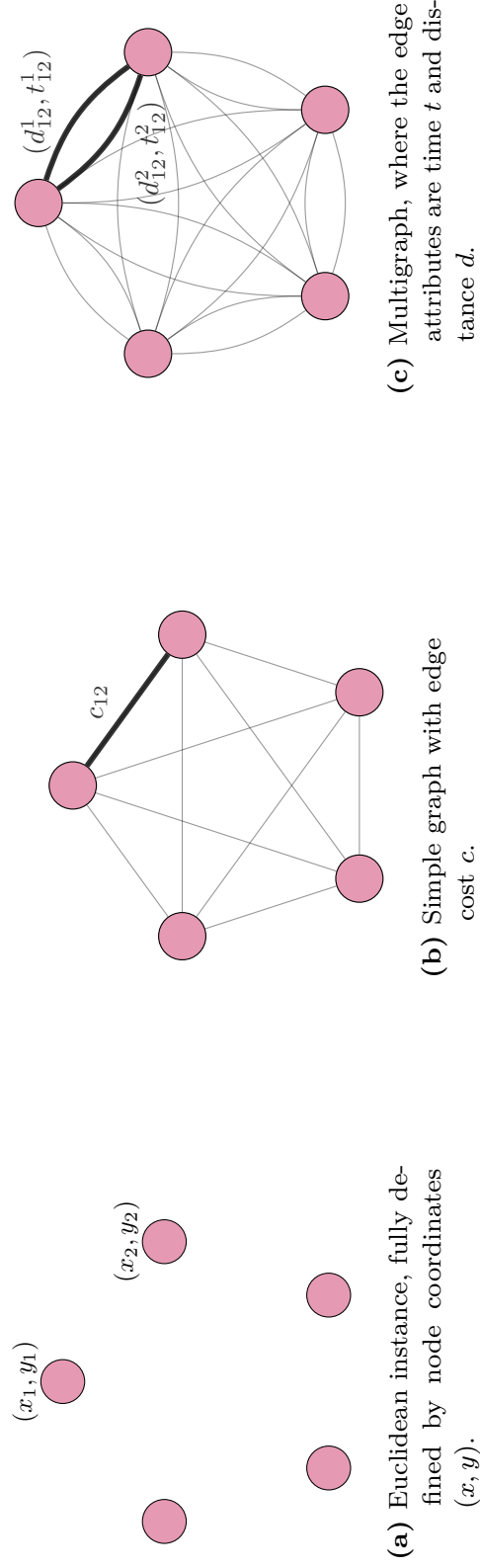


Figure 2.3: Visualization of some underlying representations for VRPs. In this thesis, multigraphs are the representations of interest. For simplicity, the graphs in the figure are undirected.

CHAPTER 3

Models and Learning Algorithms

This chapter presents background on machine learning architectures and training algorithms. The treatment is selective and we introduce only the components that appear in Chapters 4–7 and in the appended papers. A reader interested in a more comprehensive account is referred to standard references [61, 62].

Throughout the rest of the thesis, we adopt a common view of learning-based solvers as *parameterized policies*. Given a distribution $\mathcal{D}$ over problem instances, the aim is to learn parameters θ of a policy p_θ that maps any instance $G \sim \mathcal{D}$ to a solution $\pi \sim p_\theta(\cdot \mid G)$, such that the expected cost

$$\mathbb{E}_{G \sim \mathcal{D}} \left[\mathbb{E}_{\pi \sim p_\theta(\cdot \mid G)} [C(\pi, G)] \right], \quad (3.1)$$

is minimized. Here, $C(\pi, G)$ denotes the cost of solution π on instance (or graph) G . Remark that this amortized formulation is paradigm-agnostic, that is, it is independent of the architectural realization of the policy and its learning algorithm.

3.1 Neural Architectures

The policy p_θ is realized by composing neural building blocks. This section introduces these building blocks, beginning with elementary feed-forward layers, before turning to the attention mechanism and the Transformer, graph neural networks and finally recurrent sequence models.

The most basic component is the *feed-forward* (or fully connected) layer, which maps an input vector $x \in \mathbb{R}^{d_{\text{in}}}$ to

$$\text{FF}(x) = \sigma(Wx + b), \quad (3.2)$$

where W and b are a learnable weight matrix and bias vector and σ is an elementwise nonlinearity. A single such layer with a nonlinear activation is the *perceptron* and a *Multilayer Perceptron* (MLP) is obtained by stacking several layers. Despite their simplicity, MLPs are universal function approximators [63]. In the architectures that follow, they serve as the generic component wherever a learnable, pointwise transformation is required.

We also define the notion of an *embedding*. An embedding is a learned, real-valued vector representation $h \in \mathbb{R}^d$ of an entity of interest, such as a node or an edge of a routing instance, with d the embedding dimension. The architectures below can be viewed as mechanisms for computing informative embeddings. Typically, an *encoder* maps the raw features of an instance, e.g., node coordinates or edge attributes, to a set of embeddings that summarize the relevant context. Then, a *decoder* operates on these embeddings to produce the action distributions that define p_θ .

Attention and Transformers

The attention mechanism is the central architectural component of most modern neural routing methods. At its core, it computes a single output vector from a *query* and a set of *key-value* pairs. Let $q \in \mathbb{R}^{d_k}$ be a query and let $\{(k_i, v_i)\}_{i=1}^n$, with keys $k_i \in \mathbb{R}^{d_k}$ and values $v_i \in \mathbb{R}^{d_v}$, be the set attended over. Attention first scores the query against each key through a scaled inner product,

$$z_i = \frac{q^\top k_i}{\sqrt{d_k}}, \quad i = 1, \dots, n, \quad (3.3)$$

where the normalization by $\sqrt{d_k}$ counteracts the growth of the inner product with the dimension d_k . The scores are turned into non-negative weights that

sum to one by the *softmax* function,

$$\alpha_i = \text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^n \exp(z_j)}, \quad (3.4)$$

and the output is the corresponding convex combination of the values:

$$\text{Attention}(q, \{k_i\}, \{v_i\}) = \sum_{i=1}^n \alpha_i v_i. \quad (3.5)$$

In practice, many queries are attended in parallel by stacking them, together with the keys and values, as the rows of matrices $\mathbf{Q} \in \mathbb{R}^{n_q \times d_k}$, $\mathbf{K} \in \mathbb{R}^{n \times d_k}$ and $\mathbf{V} \in \mathbb{R}^{n \times d_v}$, which yields the compact scaled dot-product form [64]

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right) \mathbf{V}, \quad (3.6)$$

where the softmax is applied row-wise.

When queries, keys and values are linear projections of the same set of embeddings, the mechanism is called *self-attention* and every element attends to every other. A single such operation is limited to one notion of compatibility, so in practice *Multi-Head Attention* (MHA) runs several attention functions, or heads, in parallel and concatenates their outputs, as displayed in Figure 3.1(b). A *transformer* encoder [64] is then formed by stacking layers, each combining an MHA sublayer with a pointwise feed-forward sublayer and wrapping both in a residual connection and layer normalization [65] to stabilize the training. A single such layer is shown in Figure 3.1(a).

Self-attention is agnostic to the order of its inputs. Permuting the input embeddings permutes the outputs identically, a property known as *permutation equivariance*. For sequential data, such as natural language, order is typically reintroduced explicitly through *positional encodings* added to the input embeddings. Permutation equivariance is a desirable property for routing instances, whose nodes carry no canonical ordering, which is why positional encodings are typically omitted in this setting.

A related component, used to produce the action distributions of decoders in Chapter 4, is the pointer, or *Multi-Pointer* (MP), mechanism [66]. Instead of using the compatibility scores to combine the values, as in (3.5), a pointer uses them directly as a distribution over the candidate elements associated

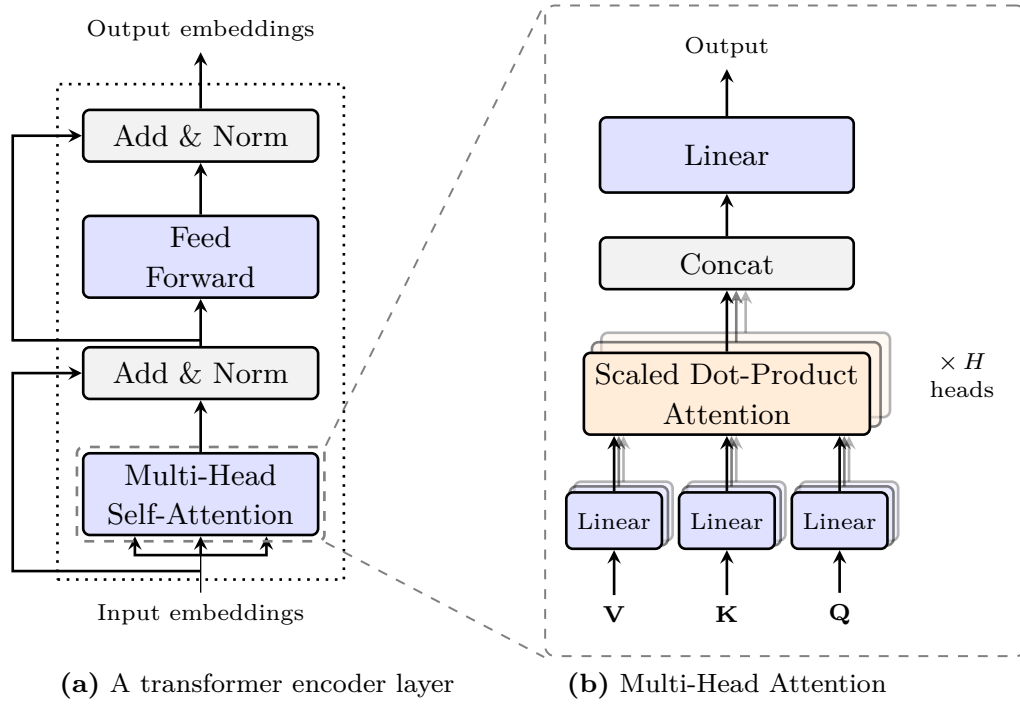


Figure 3.1: A full transformer encoder layer together with an MHA sublayer. (a) Each layer consists of MHA and feed-forward sublayers as well as normalization blocks and residual connections. (b) MHA sublayers utilize H parallel scaled dot-product attention heads, each attending over queries, keys and values obtained by linearly projecting the input embeddings.

with the keys. Given a query q and a set of keys $\{k_i\}_{i=1}^n$, a single pointer forms scaled, tanh-clipped scores and normalizes them with a softmax,

$$z_i = C_{\text{clip}} \cdot \tanh\left(\frac{q^\top k_i}{\sqrt{d_k}}\right), \quad (3.7)$$

$$p_i = \text{softmax}(z)_i, \quad (3.8)$$

where C_{clip} is a clipping constant and any inadmissible candidate can be excluded by masking its score to $-\infty$ before the softmax. A pointer is thus almost the attention layer of (3.5), except that the scores themselves are the quantities of interest. The *multi-pointer* variant runs H such pointers in parallel, each with its own projected query and keys, and averages their scores [66].

Graph Neural Networks

Routing instances are naturally represented as graphs, which motivates architectures that consume graph-structured input directly and respect its symmetries. *Graph Neural Networks* (GNNs) are the corresponding model class. A GNN, visualized in Figure 3.2, computes node embeddings through a sequence of *message-passing* rounds, in each of which every node updates its embedding by aggregating transformed embeddings from its neighbors:

$$h_u^{(\ell+1)} = \text{update}\left(h_u^{(\ell)}, \bigoplus_{v \in \mathcal{N}(u)} \text{message}(h_u^{(\ell)}, h_v^{(\ell)})\right). \quad (3.9)$$

Here, $\mathcal{N}(u)$ denotes the neighbors of node $u \in V$, $\bigoplus$ is a *permutation-invariant* aggregator such as a sum or mean and message and update are learnable functions, typically realized as feed-forward layers. The *Graph Convolutional Network* (GCN) [67, 68] is a widely used variant where the aggregation is a normalized average over the neighborhood.

While (3.9) propagates information over node embeddings, several architectures relevant to this thesis instead maintain and update edge embeddings. The Graph Edge Attention Network (GREAT) [Paper C, 57], used as the encoder in the methods of Chapter 5, is one such edge-based GNN. This property makes GREAT suitable for routing problems, which are naturally edge-centric.

A defining feature of GNNs is their behavior under relabeling of the nodes. Because the aggregation in (3.9) is permutation-invariant, node-level outputs

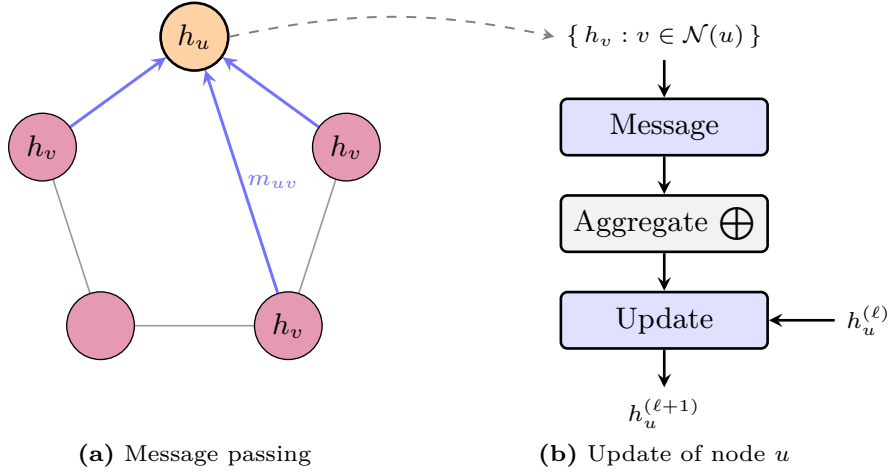


Figure 3.2: Message passing in a graph neural network. (a) Each neighbor $v \in \mathcal{N}(u)$ sends a message m_{uv} along the connecting edge. (b) The messages are combined by a permutation-invariant aggregator $\oplus$ and produce an updated embedding $h_u^{(\ell+1)}$, as in (3.9).

are permutation-equivariant. For graphs and sets that possess no canonical ordering, this is the correct inductive bias. Connecting to the previous section, self-attention over a set of node embeddings is equivalent to a round of message passing on the complete graph in which the messages are weighted by the attention scores of (3.6) [69]. The transformer encoders of the previous section may therefore be viewed as a particular kind of GNN operating on complete graphs.

A degenerate special case arises when the graph has no edges, so that the input is simply an unordered set $\{x_1, \dots, x_n\}$. Any permutation-invariant function of such a set can be written in the form

$$f(\{x_1, \dots, x_n\}) = \rho\left(\sum_{i=1}^n \phi(x_i)\right), \quad (3.10)$$

with ρ and ϕ learnable transformations, a construction known as *Deep Sets* [70]. This provides a principled way to summarize a variable-sized, unordered collection into a single fixed-length embedding. It is also the mechanism by which NEPF in Chapter 5 pools the set of parallel edges between two nodes into a compact representation, without imposing an artificial ordering on them.

Recurrent Neural Networks and Sequence Models

The architectures above are generally order-agnostic. Another family of models is designed explicitly for ordered sequences. A *Recurrent Neural Network* (RNN) processes a sequence $(x_1, \dots, x_T)$ one element at a time, maintaining a hidden state h_t that summarizes the prefix seen so far and emitting an output y_t at each step:

$$h_t = f_\theta(h_{t-1}, x_t), \quad (3.11)$$

$$y_t = g_{\theta'}(h_t), \quad (3.12)$$

with f_θ a learnable recurrence and $g_{\theta'}$ a learnable output map. The hidden state thus carries information forward along the sequence, making RNNs a natural fit for data with sequential dependencies. In their basic form, RNNs are difficult to train over long sequences, as gradients tend to vanish or explode. The *Long Short-Term Memory* (LSTM) network [71] mitigates this through a gated cell state that regulates what information is retained, forgotten and exposed at each step, enabling longer-range dependencies to be captured. When the entire sequence is available in advance, a *bidirectional* RNN processes it in both directions and concatenates the two hidden states, so that the representation at each position is informed by the full sequence. The edge selection stage of NEPF in Chapter 5 uses such a bidirectional recurrence to contextualize the edges along a fixed route.

More broadly, recurrent models have largely been replaced by attention and transformers for sequence modeling, most notably in natural language processing, where the ability of self-attention to relate distant positions directly and to be trained in parallel across a sequence outweighs the sequential processing of RNNs. Nevertheless, RNNs retain a complexity advantage, since their cost grows linearly in the sequence length whereas attention grows quadratically.

3.2 Learning Algorithms

The architectures of Section 3.1 define the function classes for the policy p_θ , but not how its parameters θ are learned. This section describes the learning algorithms used for that purpose. Recall from (3.1) that the goal is to minimize the expected solution cost over an instance distribution. The

paradigms below differ in the training signal they use to pursue this goal. Supervised learning fits the policy to precomputed reference solutions, while reinforcement learning optimizes the cost directly through a reward signal. In both cases, the parameters are updated by gradient-based optimization, with the required gradients computed by backpropagation through the network and the loss minimized with a stochastic gradient method, e.g., Adam [72].

Supervised Learning

In the supervised paradigm, the policy is trained to imitate a set of reference solutions. One assumes access to a dataset of instances, each paired with a high-quality target solution obtained from an *oracle*, typically a state-of-the-art heuristic or an exact solver run offline, and fits p_θ to reproduce these targets.

A common formulation trains the model to predict, for each candidate element l (e.g., an edge) of an instance G a probability $H_\theta(l) \in [0, 1]$ that l belongs to the solution. H_θ is commonly referred to as a *heatmap*. Writing $y_l \in \{0, 1\}$ for the oracle indicator of whether l is part of the target solution, the model is fit by minimizing the binary cross-entropy

$$\mathcal{L}(\theta) = -\mathbb{E}_{(G, y)} \left[\sum_l (y_l \log H_\theta(l) + (1 - y_l) \log(1 - H_\theta(l))) \right], \quad (3.13)$$

which drives the predicted probabilities towards the oracle labels.

Supervised learning is simple and effective whenever oracle solutions can be generated at scale. However, it is bounded by the quality of the oracle and it requires solving a large number of instances offline, which may be prohibitively expensive for large problems.

Reinforcement Learning

Reinforcement Learning (RL) removes the dependence on precomputed solutions, optimizing the objective directly and requiring only the ability to evaluate outcomes. RL is formalized in terms of *Markov Decision Processes* (MDPs). An MDP is a tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$ with a state space $\mathcal{S}$, an action space $\mathcal{A}$, a transition kernel $P : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ (where $\Delta(\mathcal{S})$ denotes the set of probability distributions over $\mathcal{S}$), a reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ and a discount factor $\gamma \in (0, 1]$. The kernel $P(\cdot \mid s, a)$ gives the distribution over next states after taking action a in state s and $r(s, a)$ is the associated scalar reward. At each step t , an agent in state s_t samples an action $a_t \sim p_\theta(\cdot \mid s_t)$

from its policy p_θ , transitions to a next state $s_{t+1} \sim P(\cdot \mid s_t, a_t)$ and receives a reward $r_t = r(s_t, a_t)$. The policy is optimized to maximize the expected *return*, i.e., the discounted sum of rewards $R = \sum_{t \geq 0} \gamma^t r_t$ accumulated along a trajectory.

This template captures the sequential construction of a routing solution as a special case. In essence, a state represents a partially built solution, an action extends it by one component, the transition is deterministic and the reward is deferred to the end of the episode, where the completed solution is evaluated and its negated objective value returned. Optimizing the policy then means maximizing the expected return

$$J(\theta) = \mathbb{E}_{G \sim \mathcal{D}} \mathbb{E}_{\pi \sim p_\theta(\cdot \mid G)} [R(\pi, G)], \quad (3.14)$$

over instances $G \sim \mathcal{D}$ and trajectories $\pi \sim p_\theta(\cdot \mid G)$, with $R(\pi, G) = -C(\pi, G)$ the negative cost of π on G .

The objective (3.14) can be optimized by many different algorithms. One simple and widely used choice, and the one on which the methods of this thesis build, is the REINFORCE algorithm [73], a *policy-gradient* method that ascends J using the gradient estimator

$$\nabla_\theta J(\theta) = \mathbb{E}_{G \sim \mathcal{D}} \mathbb{E}_{\pi \sim p_\theta(\cdot \mid G)} \left[(R(\pi, G) - b(G)) \nabla_\theta \log p_\theta(\pi \mid G) \right], \quad (3.15)$$

approximated in practice with Monte Carlo sampling. The term $b(G)$ is a *baseline*, an action-independent quantity subtracted from the return to reduce the variance of the estimator without introducing bias. Common baseline choices include a learned value estimate produced by a *critic*, the return of a deterministic greedy rollout of the current policy [11] and a shared baseline formed by averaging the returns of several trajectories sampled for the same instance [12].

Remark that REINFORCE is only one method in a larger design space. Reinforcement learning further encompasses value-based methods, which instead learn state- or action-value functions, such as Q-learning, actor–critic methods, which learn a value function alongside the policy and trust-region approaches such as Proximal Policy Optimization (PPO) [74], which constrain the size of each update for stability. These algorithms are crucial for applications beyond combinatorial optimization, from game playing and robotic control to the fine-tuning of large language models. A comprehensive treatment can be found in standard references [62]. As our methods rely on REINFORCE-style training, we do not treat the alternatives in detail.

Multi-Objective Reinforcement Learning

Many sequential decision-making problems of interest involve several conflicting objectives that must be optimized simultaneously, which extends the scalar-reward setting of the previous section to *Multi-Objective MDPs* (MOMDPs). A MOMDP retains the state space, action space and transition kernel of an ordinary MDP, but replaces the scalar reward with a vector-valued one:

$$r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^m, \quad (3.16)$$

so that a trajectory accumulates a vector-valued return

$$R = \sum_{t \geq 0} \gamma^t r_t, \quad r_t = r(s_t, a_t). \quad (3.17)$$

In the route construction setting, this return again corresponds to the negative cost vector $-C(\pi, G) \in \mathbb{R}^m$. One would ideally maximize every component of the expected return $\mathbb{E}[R]$ at once. However, since the objectives typically conflict, no single policy can generally accomplish this. Instead, in analogy with Section 2.4, the goal becomes obtaining a set of policies covering the Pareto front.

Approaches to this problem can be organized along two axes. The first concerns when the trade-off between objectives is resolved. *Outer-loop* methods reduce the multi-objective problem to a family of scalar subproblems and solve them separately, whereas *inner-loop* methods incorporate the trade-off into the learning process and train against a distribution of trade-offs at once. The second axis concerns how many policies are learned. *Multi-policy* methods train a separate model for each trade-off, which is conceptually simple but yields impractically many models when a dense approximation of the Pareto front is required, or when there are many objectives. On the other hand, *single-policy*, or *preference-conditioned*, methods train one model that is conditioned on the desired trade-off and can recover the Pareto front without retraining.

The connection between scalar RL and the multi-objective setting is scalarization. Applying a scalarization function g_λ (see Section 2.4) to the return (3.17) turns the MOMDP into a scalar MDP with return $R_\lambda = g_\lambda(R)$, which can be optimized with the single-objective RL methods from the previous section. A multi-policy outer-loop method solves one such scalarized problem independently for each preference of interest. A single-policy method instead conditions the policy p_θ on the preference λ and trains it over a distribution Λ

of preferences by maximizing the *expected scalarized return*

$$J(\theta) = \mathbb{E}_{\lambda \sim \Lambda} \mathbb{E}_{s_0 \sim \mu} \mathbb{E}_{\pi \sim p_\theta(\cdot | s_0, \lambda)} [R_\lambda], \quad (3.18)$$

with s_0 the initial state and μ its distribution. Now, the gradient can be obtained by the policy-gradient estimator (3.15), with a preference-dependent reward and baseline. In the route construction setting, the initial state s_0 is the sampled instance $G \sim \mathcal{D}$ and scalarizing the cost vector of Section 2.4 gives the per-trajectory reward $R_\lambda = -g_\lambda(C(\pi, G))$, so that (3.18) specializes to

$$J(\theta) = \mathbb{E}_{\lambda \sim \Lambda} \mathbb{E}_{G \sim \mathcal{D}} \mathbb{E}_{\pi \sim p_\theta(\cdot | G, \lambda)} [-g_\lambda(C(\pi, G))], \quad (3.19)$$

i.e., the negative expected scalarized cost. A single trained model thereby approximates the Pareto front, evaluated at any preference λ at inference time.

A common way to realize the λ -conditioning is a *hypernetwork*, an auxiliary network that maps the preference λ to the weights of (part of) the policy. This preference-conditioned, single-policy strategy is commonly adopted by the methods of Chapter 4.

CHAPTER 4

Neural Combinatorial Optimization

This chapter reviews and discusses related work regarding machine learning for combinatorial optimization. The focus is on end-to-end learning, commonly referred to as *Neural Combinatorial Optimization* (NCO), but other paradigms, such as hybrid methods, are treated briefly in Section 4.5. First, a taxonomy of methodologies is presented, before the various classes are discussed in detail. Then, the chapter reviews open challenges and prominent research streams within NCO. Throughout, applications to vehicle routing problems are particularly emphasized, while other COPs (e.g., scheduling problems and graph problems such as the Maximum Independent Set, MIS) are less emphasized. The last section explains directly why prior methods do not cover multigraph VRPs in a satisfactory way.

4.1 Taxonomy of Methodologies

NCO methods can be distinguished along several axes, concerning the learning paradigm, the backbone architecture and the manner in which a solution is constructed.

First, methods can be distinguished by their learning paradigm. Supervised

learning trains the model to imitate ground-truth solutions, typically obtained with state-of-the-art heuristics (e.g., LKH for the TSP) or exact solvers. This is a common and effective strategy whenever such oracle solutions can be generated at scale [75]. However, oracle solutions are not always available, or are too costly to generate for large training sets, motivating reinforcement learning and, to a lesser extent, unsupervised learning as alternative paradigms that train directly from a cost or reward signal.

Second, methods can be distinguished by their backbone architecture. As routing problems are naturally posed on graphs, GNNs are a natural choice of architecture. However, they suffer from *over-smoothing* on the complete graphs typical of routing instances and transformers operating directly on node features have consequently emerged as the more widely used backbone in recent work.

Third, and most central to the taxonomy in this chapter, methods can be distinguished by how a solution is constructed. *Autoregressive* (AR) methods build a solution one component at a time, with a model forward pass between each step to recompute action probabilities conditioned on the partial solution so far. *Non-Autoregressive* (NAR) methods instead perform a single forward pass to output a heatmap of edge or node scores, which is then decoded into a complete solution, either by simple greedy selection (i.e., iteratively selecting elements in order of decreasing score) or by a more advanced search procedure. Finally, *improvement-based* methods start from a complete but suboptimal solution and iteratively refine it, typically through local moves, to obtain a solution with lower objective value. The remainder of this chapter reviews the literature in depth, organized along this AR, NAR and improvement-based classification.

4.2 Autoregressive Construction

In the autoregressive paradigm, a model learns to sequentially construct routes one action at a time. Let $\pi = (\pi_1, \dots, \pi_T)$ denote a route consisting of T components π_t , $t = 1, \dots, T$ (e.g., nodes) and let p_θ denote the neural network parameterizing the routing policy. An AR model iteratively samples

$$\pi_{t+1} \sim p_\theta(\cdot \mid \pi_{1:t}, G), \quad (4.1)$$

where G is the problem instance. This iterative construction naturally fits into the MDP formulation of reinforcement learning (see the previous chapter), where the action is the next component π_{t+1} to insert and the state is given by $\pi_{1:t}$. Hence, many prominent AR methods employ RL for training [10, 11, 12, 24, 76].

In practice, p_θ is often realized as an encoder–decoder architecture. An encoder first maps the problem instance (e.g., node coordinates or other features) into a set of embeddings summarizing the relevant context. At each construction step, a decoder then combines these embeddings with the partial solution $\pi_{1:t}$ to produce the action probabilities $p_\theta(\cdot \mid \pi_{1:t}, G)$, typically through an attention mechanism over the embeddings of the remaining, unvisited elements. Feasibility is commonly enforced through masking, i.e., the probabilities of infeasible actions, such as revisiting an already-visited node, are set to zero before normalizing. This construction process is visualized in Figure 4.1.

The first prominent neural architecture for sequentially constructing combinatorial solutions was the Pointer Network of Vinyals et al. [9]. Standard sequence-to-sequence models, for instance in natural language processing, output a distribution over a fixed-size vocabulary at each decoding step, which is unsuitable for combinatorial problems, since the relevant “vocabulary” is the set of feasible next elements and depends on the size of the input instance. To address this, Vinyals et al. replace the fixed output vocabulary with a content-based attention mechanism over the input elements, so that the decoder outputs a soft pointer to one of the input elements rather than to a fixed dictionary entry. This mechanism allows pointer networks to generalize across instances of varying size. Vinyals et al. train pointer networks with supervised learning to solve the Euclidean TSP.

Bello et al. [10] extend this approach by training a pointer-network policy with RL instead of supervised learning. Framing route construction as a sequential decision process, they optimize the policy with the REINFORCE policy-gradient algorithm [73], using the negative tour length as the reward, and reduce gradient variance with a learned critic network that estimates the expected tour length as a baseline. Beyond RL pretraining on a distribution of training instances, Bello et al. also introduce *active search*, where the policy parameters are further refined for an individual test instance by continuing to optimize the RL objective at inference time, trading additional computation for improved solution quality. Their resulting method outperforms the supervised

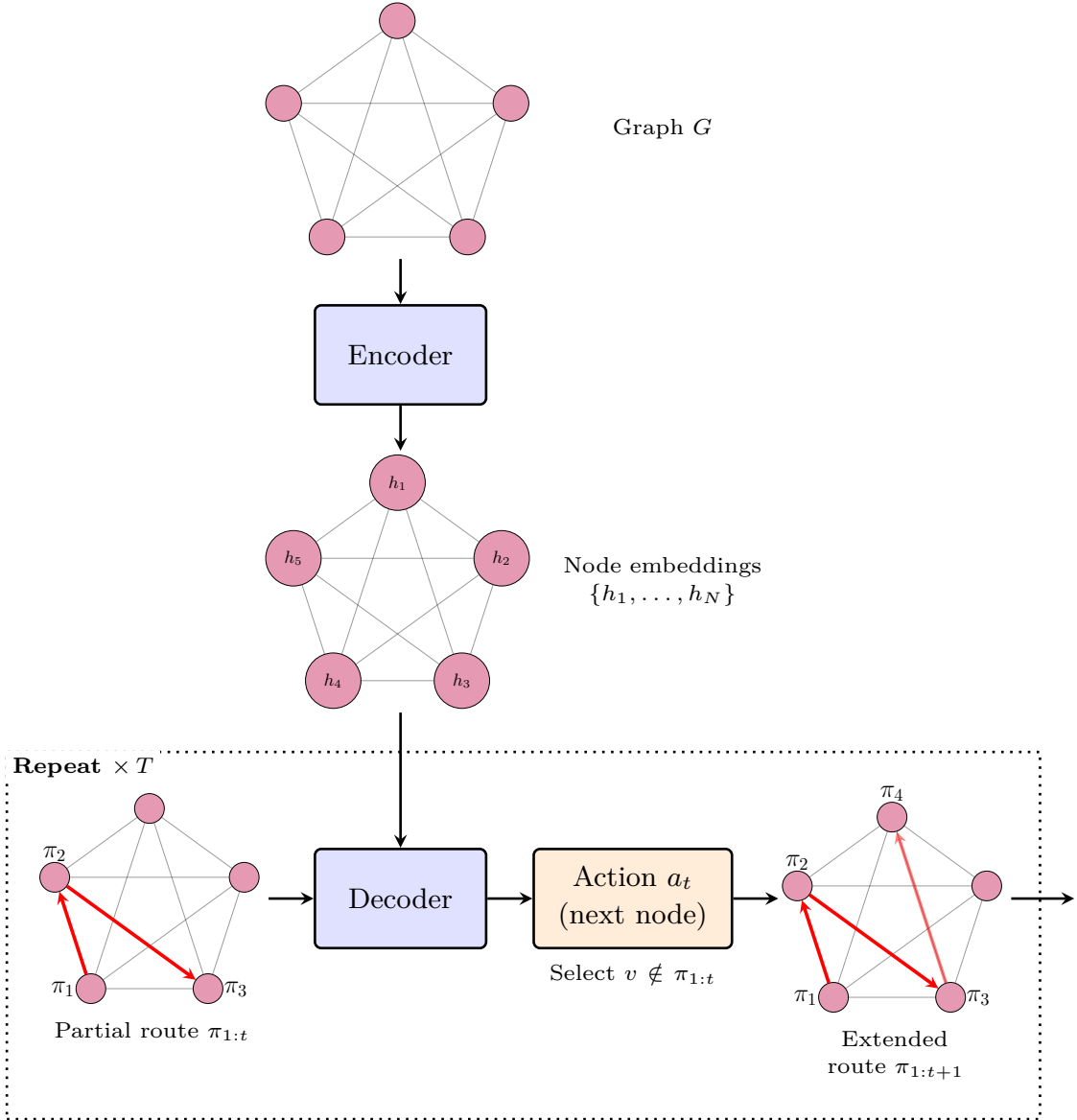


Figure 4.1: A visualization of AR route construction for the TSP. Routes are extended one node at a time based on the partial route context and the encoded instance.

pointer network of Vinyals et al. [9] and attains near-optimal tours on Euclidean TSP instances with up to 100 nodes.

Contemporaneously, Dai et al. [77] pursue a similar RL-based, label-free construction strategy with S2V-DQN, but target a broader class of graph problems, including the Minimum Vertex Cover problem, Maximum Cut problem and the TSP. Rather than an attention-based pointer mechanism, they embed the partial solution with `structure2vec`, a graph embedding technique that recursively aggregates information from a node’s neighborhood, and train a greedy construction policy with fitted Q-learning, using the resulting embeddings to estimate the value of adding each candidate node next. S2V-DQN performs comparably to classical heuristics such as 2-opt on the TSP, while substantially outperforming them on the Minimum Vertex Cover and Maximum Cut.

While Bello et al. [10] retain the recurrent pointer network architecture of Vinyals et al. [9], Kool et al. [11] instead build the encoder on the transformer architecture [64], replacing the LSTM-based encoder with multi-head self-attention layers operating on the node features. Since the resulting node embeddings are permutation-equivariant rather than order-dependent, no positional embedding is required. At each decoding step, the decoder forms a context vector by concatenating a global graph embedding with the embeddings of the first and most recently visited nodes and computes a compatibility score against the embeddings of the unvisited nodes, masked to enforce feasibility. Kool et al. train this Attention Model (AM) with REINFORCE using a deterministic greedy-rollout baseline, which they find to be simpler and more effective than the learned critic of Bello et al. [10]. The AM obtains near-optimal solutions for Euclidean TSP instances with up to 100 nodes and learns competitive heuristics for the CVRP, the Orienteering Problem and a stochastic variant of the Prize Collecting TSP, establishing the transformer-based encoder–decoder as a dominant backbone for subsequent AR methods.

Building directly on the AM, Kwon et al. [12] observe that a single optimal solution typically admits multiple equivalent sequential representations. For instance, a TSP tour can be written starting from any of its N nodes. Prior methods construct only a single trajectory per instance, starting from a learned placeholder token, and so implicitly favor one such representation over the others. Policy Optimization with Multiple Optima (POMO) instead samples N trajectories in parallel, one initialized from each node, so that training explicitly

explores all equivalent representations of a solution. The average reward across these N trajectories then serves as a shared baseline for the REINFORCE gradient, removing the need for a separate critic or rollout network required by Bello et al. [10] and Kool et al. [11]. At inference, the same multi-start idea yields several greedy solutions per instance, which Kwon et al. complement with *instance augmentation*: applying problem-preserving transformations (e.g., reflections and rotations of the unit square for Euclidean instances) to construct further views of the same instance, from which additional candidate solutions can be obtained. POMO reduces the optimality gap on Euclidean TSP100 to 0.14%, a substantial improvement over the AM, and obtains similarly strong results on the CVRP.

Kim et al. [76] build on POMO’s multi-start idea, observing that many COPs and their solutions possess inherent symmetries, e.g., rotational and reflectional invariance for Euclidean instances, that are not explicitly enforced by standard training objectives. Their Sym-NCO instead introduces a symmetry-aware regularization term that encourages the policy to assign consistent probability to symmetric instance–solution pairs. Applied on top of existing AR backbones, the regularizer improves generalization across the TSP, CVRP, Prize Collecting TSP and Orienteering Problem, indicating that exploiting problem symmetries is complementary to architectural advances in improving sample efficiency.

The works by Vinyals et al., Bello et al., Dai et al., Kool et al., Kwon et al. and Kim et al. are foundational in the AR construction regime. Since these contributions, AR methods have been extended and refined to improve efficiency along various dimensions. The treatment of these more recent methods is deferred to Section 4.6, which reviews open problems and prominent directions within NCO.

4.3 Non-Autoregressive Construction

In the non-autoregressive paradigm, a model instead performs a single forward pass to score the elements of a problem instance, typically the edges of the underlying graph, by how likely they are to belong to a high-quality solution, rather than conditioning repeated forward passes on the partial solution. Formally, given a problem instance G , a NAR model outputs a heatmap H_θ assigning a score

$$H_\theta(u, v) = p_\theta((u, v) \in \pi \mid G) \quad (4.2)$$

to each edge (u, v) (or, for node-based problems, a score to each node), all at once. This heatmap must then be decoded into a complete, feasible solution, either through simple greedy decoding, i.e., iteratively selecting the highest-scoring remaining edges or nodes subject to feasibility, or through a more elaborate procedure, such as beam search, sampling or local search. This overall solution strategy is visualized in Figure 4.2. Because no sequential dependency between decoding steps needs to be modeled, NAR methods amortize most of their computation into a single forward pass, typically making them faster at inference than AR methods, particularly on large instances [56, 78]. However, the resulting heatmap still requires a separate decoding stage and final solution quality depends critically on how effectively this decoding procedure exploits it [79, 80].

In practice, H_θ is commonly realized with a GNN operating directly on the instance graph, since predicting a score per edge or node is a natural fit for message-passing operations. More recent work has also explored transformer- and diffusion-based heatmap generators. Whereas the sequential construction process of AR methods aligns naturally with reinforcement learning, NAR methods have more commonly relied on supervised learning, regressing the heatmap towards edge or node indicators derived from precomputed, (near-)optimal solutions (see Section 3.2). Reinforcement learning and other label-free training signals have nonetheless also been explored.

The first prominent NAR architecture for combinatorial optimization is the GCN model of Joshi et al. [68], which applies a deep graph convolutional network directly to a TSP instance graph, learning representations for the nodes and edges that are projected to a single scalar score per edge, representing the heatmap of the likelihood that the edge belongs to the optimal tour. Trained with supervised learning against ground-truth labels and decoded with a parallelizable beam search procedure, Joshi et al.’s model substantially reduces the optimality gap on Euclidean TSP instances with 50 and 100 nodes relative to earlier, non-GNN heuristics, establishing the basic heatmap-and-decode methodology that most subsequent NAR methods follow.

Several subsequent works improve specifically on the decoding stage that converts a predicted heatmap into a feasible solution. Kool et al. [81] propose Deep Policy Dynamic Programming (DPDP), training a GNN to predict a heatmap, but replacing beam search with a Dynamic Programming (DP) procedure whose state space is restricted to only the most promising edges

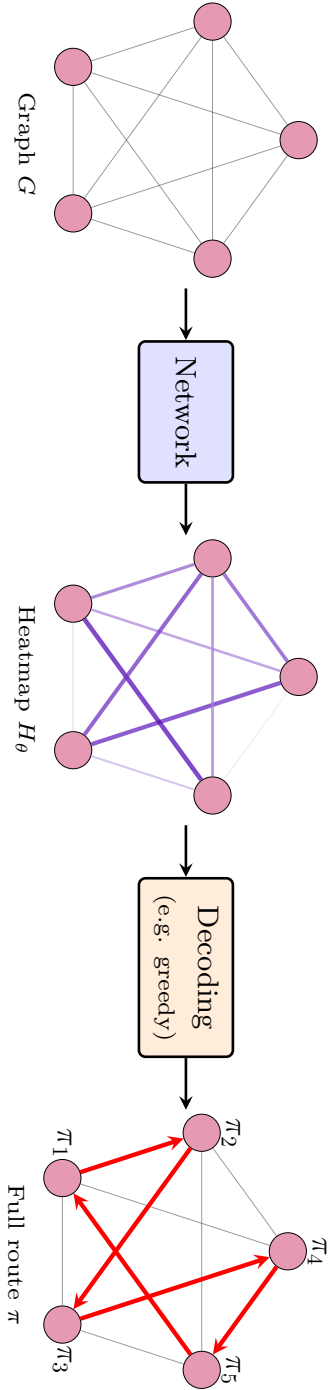


Figure 4.2: Visualization of NAR route construction. The model outputs a heatmap, which is then transformed into a feasible route with some search strategy.

identified by the heatmap. This approach combines DP’s multi-step lookahead with scalability obtained by pruning its prohibitively large state space, which yields strong results on the TSP, the CVRP and the TSP with Time Windows (TSPTW).

Hudson et al. [82] instead target local search. Their model predicts, for every edge, a regret value estimating the cost of forcing that edge into the solution, which is then used to guide the order in which a classical, 2-opt-based local search considers candidate moves. The resulting Guided Local Search finds high-quality TSP solutions substantially faster than unguided local search by focusing computation on the edges the GNN identifies as most likely to be suboptimal.

To address the scalability of heatmap generation, Qiu et al. [83] propose DIMES, which replaces the direct prediction of discrete solution scores used in earlier neural approaches with a compact continuous parameterization of the distribution over candidate solutions. This parameterization enables stable training via REINFORCE and supports efficient instance-specific fine-tuning through massively parallel sampling, without requiring precomputed solution labels. To obtain effective initial parameters for this fine-tuning, Qiu et al. additionally propose a *meta-learning* scheme. DIMES scales to TSP instances with up to 10 000 nodes, considerably larger than the instances addressed by many earlier neural methods.

A further line of work treats heatmap generation as a generative-modeling problem. Sun and Yang [56] propose DIFUSCO, which models the discrete, $\{0, 1\}$ -valued edge-indicator heatmap with a graph-based denoising *diffusion* model. Starting from random noise, an anisotropic GNN iteratively denoises the heatmap over a sequence of diffusion steps, in contrast to the single forward pass of earlier NAR methods. Sun and Yang find Bernoulli-distributed, discrete noise to suit the discrete nature of COPs better than the Gaussian noise standard in image-diffusion models and DIFUSCO substantially outperforms prior NAR solvers on the TSP and MIS.

Li et al. [84] build on this diffusion-based formulation with T2T, observing that training a generative model to fit the full solution distribution, as DIFUSCO does, is misaligned with the goal of finding a single, optimal solution per test instance. T2T therefore retains DIFUSCO’s diffusion model for training but at test time interleaves the denoising process with gradient-based search steps that directly optimize the heatmap against the problem’s objective,

improving solution quality over vanilla DIFUSCO on the TSP and MIS.

Other works depart from the diffusion-based formulation while retaining the idea of learning a distribution over solutions, rather than a single heatmap. Zhang et al. [85] instead train Generative Flow Networks (GFlowNets), which learn a stochastic policy that constructs solutions with probability proportional to their quality, encouraging diverse sample routes instead of collapsing onto a single mode, as standard RL objectives tend to.

Xiao et al. [78], proposing NAR4TSP, instead address a prominent weakness of NAR training directly. Rather than training the heatmap network with supervised learning against precomputed labels, NAR4TSP integrates the non-autoregressive decoding procedure into an RL training loop. This is among the first NAR TSP solvers trained successfully with RL rather than supervised learning and NAR4TSP outperforms several state-of-the-art neural baselines in solution quality, inference speed and generalization to real-world instances.

4.4 Improvement-Based Approaches

The third major class of NCO methods aims to improve already feasible, but potentially suboptimal routes. Instead of constructing a solution from scratch, an improvement-based method starts from a complete solution $\pi^{(0)}$, typically obtained with a simple construction heuristic, and iteratively refines it through a sequence of local moves, such as the well-known 2-opt move, which reverses a segment of a tour. At each iteration k , a learned policy p_θ selects an action a_k , e.g., which move to apply and to which part of the current solution, conditioned on the current solution:

$$a_k \sim p_\theta(\cdot \mid \pi^{(k)}, G). \quad (4.3)$$

A typically fixed and hand-designed operator $\mathcal{T}$ is then applied to obtain the next solution based on the action according to

$$\pi^{(k+1)} = \mathcal{T}(\pi^{(k)}, a_k). \quad (4.4)$$

Unlike AR construction, where each step extends a partial solution by exactly one component, every iterate $\pi^{(k)}$ remains a complete, feasible solution throughout. This solution process is summarized in Figure 4.3.

In practice, p_θ is often realized similarly to the AR encoder-decoder architectures discussed above, with an encoder summarizing the current solution and instance into a set of embeddings and a decoder selecting an action from

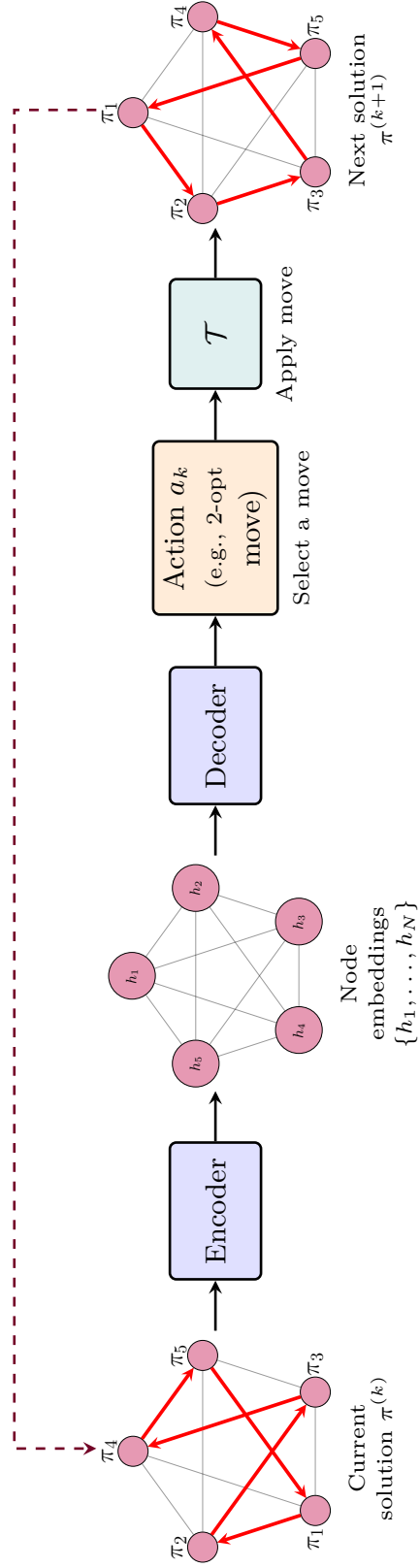


Figure 4.3: Visualization of improvement-based methods. At each step, the model chooses a potentially improving move, yielding a new route to which the process can be repeated.

these embeddings, e.g., a pair of nodes defining a 2-opt move. Because the reward signal, i.e., the resulting change in objective value, is available at every iteration, improvement-based methods are most commonly trained with RL. At inference, the policy is simply applied repeatedly for either a fixed number of iterations or a fixed computational budget, with the best solution encountered along the trajectory returned. This yields a flexible trade-off between solution quality and runtime that AR and NAR methods, which terminate after a single construction or decoding pass, do not as naturally provide. On the other hand, improvement-based methods typically underperform given the same computational budget as other methods, often requiring a large number of moves to converge to high-quality solutions [23, 56].

The earliest prominent neural improvement heuristic is NeuRewriter of Chen and Tian [86], which learns to iteratively rewrite local components of a solution via two cooperating policies: a region-picking policy that selects which part of the current solution to modify and a rule-picking policy that selects which of several hand-specified rewriting rules to apply. Both policies are parameterized by neural networks and trained jointly with an actor-critic RL algorithm. Because rewriting rules can be specified for arbitrary discrete structures rather than only routes, Chen and Tian demonstrate NeuRewriter across three distinct domains, namely expression simplification, job scheduling and vehicle routing.

Two further works target the classical 2-opt operator. Wu et al. [87] propose a self-attention-based policy network that, at each iteration, selects a pair of nodes defining a 2-opt move, or, for the CVRP, an analogous node-swap move, to apply to the current solution, training the policy with RL using the resulting change in tour length as a reward. Costa et al. [88] develop a similar, RL-trained policy, but realize it with a recurrent, pointing-attention-based architecture reminiscent of pointer networks. Both approaches demonstrate that a learned policy, applied repeatedly, can substantially improve initially poor or random solutions, narrowing the gap to optimal solutions faster than the classical, hand-crafted heuristics on which their move operators are based.

Ma et al. [89] address a limitation of such attention-based improvement policies, namely that positional encodings designed for linear, natural-language sequences do not suit the position of a node within a route, which is circular and direction-agnostic. Their Dual-Aspect Collaborative Transformer (DACT) embeds node and positional features in two separate streams, instead of

fusing them as prior improvement models do, and introduces a cyclic positional encoding that respects the circularity and reflective symmetry of a tour. Trained with PPO and a size-based curriculum, DACT outperforms earlier transformer-based improvement models on the TSP and CVRP and generalizes considerably better across problem sizes.

Hottung and Tierney [90] instead use a destroy-and-repair paradigm rather than operating with simpler edge-exchange moves. Their Neural Large Neighborhood Search (NLNS) iteratively destroys part of the current CVRP solution, e.g., by removing a subset of customers from the routes, and then repairs it by learning to reinsert the removed customers with an attention-based neural network, trained with RL to minimize the resulting tour length. Because repair is more difficult than destroy, learning only the repair operator, while destruction follows simple randomized rules, proves sufficient for NLNS to approach the performance of state-of-the-art operations-research solvers on the CVRP and a stochastic variant thereof.

Kim et al. [91] target a more specialized operator with their Neuro CROSS-Exchange (NCE) model, which swaps sub-tours between pairs of routes in multi-vehicle routing problems. Naively evaluating every candidate sub-tour pair for a CROSS-exchange move scales as $\mathcal{O}(N^4)$. Thus, Kim et al. instead train a GNN with supervised learning to directly predict the cost-decrement each candidate exchange would yield, reducing the cost of selecting promising moves to $\mathcal{O}(N^2)$. Applied to min-max variants of the flexible multi-depot VRP, where the objective is to minimize the length of the longest individual route rather than the total length, NCE outperforms classical metaheuristic baselines that rely on the same underlying CROSS-exchange operator.

Most recently, Ma et al. [92] generalize the elementary 2-opt moves used by most prior improvement heuristics into flexible k -opt exchanges with their NeuOpt model. They factorize an arbitrary k -opt move into a short sequence of three basis moves, namely a starting move, an arbitrary number of intermediate moves and an ending move, that a customized, recurrent dual-stream decoder can select step by step. Rather than relying only on feasibility masking, which can overly restrict the moves available to the policy, Ma et al. further introduce Guided Infeasible Region Exploration (GIRE), which permits the policy to temporarily traverse infeasible solutions. Combined with diversified search at inference time, NeuOpt substantially outperforms prior masking-based improvement heuristics on the TSP and CVRP.

4.5 Other Learning-Based Strategies

Beyond the end-to-end learning-based methods, divided into the AR, NAR and improvement-based categories reviewed above, a number of other learning-based strategies have emerged. Given that these methods are less relevant for this thesis, the following is intended as a short outlook rather than an exhaustive review.

LLM-based methods. Instead of training a network to construct or improve solutions directly, a growing body of work uses Large Language Models (LLMs) to automatically design the heuristics, expressed as executable code. Romera-Paredes et al. [93] demonstrate this with FunSearch, pairing an LLM with an evolutionary search procedure that proposes, evaluates and refines candidate heuristics as short programs, discovering improved constructions for problems such as the cap set problem and online bin packing. Ye et al. [94] adapt this idea specifically to combinatorial optimization with ReEvo, adding a second, “reflector” LLM that critiques each generation of heuristics in natural language to guide the next. Comprehensive benchmarks have since demonstrated the effectiveness of such LLM-based *hyper-heuristics*, showing their competitiveness across multiple combinatorial optimization problems on difficult instances [95].

Hybrid methods. A second direction integrates learned components directly into classical, non-neural solvers. Xin et al. [96] do so with NeuroLKH, training a graph neural network to predict edge scores and penalties that guide the candidate set of the LKH heuristic, substantially improving on vanilla LKH across several routing problems. Ye et al. [97] pursue a similar strategy for Ant Colony Optimization (ACO) with DeepACO, training a neural network to predict the heuristic and pheromone-update measures that guide ACO’s probabilistic construction, consistently outperforming classical ACO variants across eight COPs with a single model and hyperparameter set. Gasse et al. [98] instead target general mixed-integer linear programs, training a GCN to imitate the branching rule within a branch-and-bound solver, reducing the number of nodes explored while retaining optimality guarantees. Overall, such hybrid methods represent a promising direction, where both the advantages of learning-based and classical methods can be utilized.

4.6 Research Directions and Challenges

Next, some of the main streams of research in neural combinatorial optimization are reviewed. In particular, the focus lies on methods and perspectives relevant for the rest of the thesis. The treatment is not exhaustive and an interested reader is referred to recent review articles for further reading [13, 75, 99]. Notably, a clear majority of the methods discussed below either belong to the AR category or feature autoregressive components, reflecting AR’s status as the most extensively studied construction paradigm.

Asymmetric Routing Problems

A first such direction concerns asymmetric routing problems. The foundational architectures discussed above derive node embeddings from coordinates and consequently assume symmetric, Euclidean instances. As discussed in Section 2.5, this assumption is frequently violated in practice, e.g., when travel times are direction-dependent due to traffic or road topology.

Kwon et al. [24] address this with the Matrix Encoding Network (MatNet), which dispenses with coordinate-based node embeddings and instead incorporates distance information via *mixed-score attention layers* utilizing the instance distance matrix. The model maintains separate sets of embeddings for the row- and column-entities of the matrix and computes attention scores that combine these embeddings with the corresponding distance matrix entry. This matrix-based encoding allows MatNet to solve problems, such as the ATSP and the flexible flow-shop problem, for which no natural Euclidean representation exists.

Lischka et al. [57] also depart from coordinate-based encoding, but operate on edges rather than nodes. Their Graph Edge Attention Network maintains a learned embedding for every directed edge and updates it through attention sublayers. Because each directed edge has its own embedding, GREAT can represent arbitrary asymmetric cost structures, without assuming any underlying Euclidean or symmetric geometry, and the authors report strong performance on asymmetric routing instances.

Two more recent works, by Son et al. [27] and Yi et al. [25], instead retain node-based architectures but augment them with mechanisms specifically targeting asymmetry. Son et al. propose RRNCO, which fuses sampled, direction-dependent distances with coordinate features into an Adaptive Node

Embedding and introduces a Neural Adaptive Bias that jointly conditions attention on distance, travel-time and directional-angle matrices, enabling the model to capture asymmetries that arise in real-world road networks. Yi et al. introduce RADAR, which constructs asymmetry-aware initial node embeddings via a truncated singular value decomposition of the distance matrix.

Hard-Constrained Problems

A second challenge concerns problems with hard, complex constraints that are difficult to enforce through simple feasibility masking. For constraints such as the time windows of the TSPTW or the draft limits of the TSP with Draft Limit (TSPDL), determining which of the remaining actions are feasible is itself an $\mathcal{NP}$ -hard problem, so that the masking procedure AR methods rely on for simpler constraints (e.g., avoiding already visited nodes) breaks down.

Bi et al. [22] address this with Proactive Infeasibility Prevention (PIP), which augments the construction policy with a Lagrangian-multiplier-based constraint-awareness term and a preventative masking mechanism that discourages actions likely to lead to infeasibility later in construction. To avoid the cost of repeatedly evaluating the mask during training, Bi et al. also propose PIP-D, which employs an auxiliary decoder trained to predict the mask. Applied on top of several backbone constructors for the TSPTW and TSPDL across different constraint-hardness levels, PIP substantially reduces the fraction of infeasible solutions while improving solution quality.

Rather than modifying the masking procedure or the architecture, Fang et al. [100] propose Universal Constrained Preference Optimization (UCPO), a fine-tuning objective that rewards a pretrained policy for preferring feasible over infeasible solutions. Since it requires no architectural changes and warm-starts from an already-trained model, UCPO only needs a small fraction of the original training budget.

Finally, Bi et al. [23] propose Construct-and-Refine (CaR), which pairs an AR construction policy with a lightweight, learned improvement procedure sharing the same encoder, training the two jointly so that the constructor learns to produce diverse candidate solutions that the improvement procedure can repair into feasibility in circa ten refinement steps. On the TSPTW, CaR attains a $6\text{--}8\times$ speedup over state-of-the-art neural baselines while improving both feasibility and solution quality, even recovering feasible solutions that LKH-3 fails to find.

Generalization to Large Instances

A third challenge concerns generalization to instance sizes much larger than those encountered during training. NCO models are typically trained on small instances for computational reasons, yet many real-world applications require solving instances with thousands of nodes and applying a small-instance-trained policy to such instances often degrades performance substantially.

Drakulic et al. [101] address this by reformulating solution construction as an MDP whose state space is reduced through *bisimulation quotienting*. Rather than conditioning the policy on the full partial solution, it is conditioned only on a problem-specific summary state, e.g., the unordered set of unvisited nodes together with the current location for the TSP and the CVRP, that suffices to determine an optimal continuation. Training a transformer-based encoder–decoder over this reduced state with supervised learning, their BQ-NCO model obtains strong out-of-distribution generalization across four different routing problems.

Concurrently, Jin et al. [66] target the memory and computational cost of attention-based encoder–decoders with Pointerformer, which combines a reversible residual encoder, whose intermediate activations need not be stored for backpropagation, with a light multi-pointer decoder. This substantially reduces Pointerformer’s computation time and memory consumption, enabling applications to large instances, while performance remains comparable to similar encoder–decoder models.

Luo et al. [102] take this redistribution of computation further with a Light Encoder, Heavy Decoder (LEHD) architecture. A minimal, single-layer encoder is paired with a deep decoder that, at every construction step, re-attends over the embeddings of all remaining nodes to dynamically recompute their relationships, rather than relying on a static encoding computed once. Despite being trained only on small instances, LEHD generates near-optimal solutions for TSP and CVRP instances with up to 1 000 nodes and generalizes well to the real-world TSPLIB and CVRPLIB benchmarks.

Another strategy for scaling to large instances is to decompose them into smaller subproblems that existing methods can solve. Ye et al. [79] adopt this strategy in GLOP, a hierarchical framework that first uses a fast NAR model to partition a large instance into smaller TSPs, e.g., one route per vehicle for the CVRP, and then decomposes each resulting TSP further into shortest Hamiltonian path subproblems. Each subproblem is finally solved with a

lightweight AR constructor. This hybrid design allows GLOP to solve large-scale TSP, ATSP, CVRP and Prize Collecting TSP instances with thousands of nodes in real time.

Cross-Distribution Generalization

A fourth challenge concerns generalization across instance distributions, rather than problem sizes. Most NCO methods are trained on a fixed distribution of training instances, e.g., uniformly sampled node coordinates, and their performance can degrade substantially when deployed on differently structured or clustered instances at test time.

Zhou et al. [103] address this, jointly with generalization across instance size, with a meta-learning framework built on top of POMO. They train a single set of initial parameters with a meta-learning objective over a curriculum of distributions and sizes, complemented by a multi-task batch-construction scheme and a scaled sampling procedure that stabilizes meta-training. At deployment, the resulting model can be rapidly fine-tuned to a new, previously unseen combination of size and distribution with only a few gradient steps, substantially outperforming zero-shot transfer of a standard, single-distribution-trained POMO model on the TSP and CVRP.

Bi et al. [104] pursue robustness to unseen distributions without any test-time fine-tuning, through an Adaptive Multi-Distribution Knowledge Distillation (AMDKD) scheme. Several teacher models, each trained on a different distribution, e.g., uniform, clustered or mixed, are *distilled* into a single, lightweight student model via attention- and value-based distillation losses. The resulting student model, without any fine-tuning, generalizes more robustly across both unseen in-distribution and out-of-distribution instances than a comparable single-distribution-trained model, while also being smaller and faster at inference.

Cross-Problem Generalization

A fifth challenge concerns cross-problem generalization. Rather than training and deploying a separate model for every problem variant, a single model is instead trained to solve many variants at once, ideally generalizing to variants unseen during training.

Zhou et al. [14] pursue this with MVMoE, which augments a standard AR

encoder–decoder by replacing its feed-forward layers with a *Mixture-of-Experts* (MoE), i.e., several parallel feed-forward sub-networks (experts), of which only a subset is activated for any given input via a learned gating function. Since different combinations of VRP attributes, e.g., capacity limits, time windows or backhauls, can thereby be associated with different experts, this increases model capacity without a significant increase in computation. Zhou et al. also introduce a hierarchical gating mechanism, which first decides whether an input requires only a shared, general-purpose computation or the additional, more specialized capacity of a second, sparsely gated MoE layer, improving the trade-off between performance and computational complexity. The resulting model exhibits strong zero-shot generalization to ten previously unseen combinations of VRP attributes by appropriately recombining the trained experts.

Drakulic et al. [15] target an even broader notion of cross-problem generalization with GOAL (Generalist combinatorial Optimization Agent Learner), which couples a single, shared transformer backbone with lightweight, problem-specific adapters that project each problem’s particular node-, edge- and instance-level features into and out of the backbone’s representation. For problems with multiple heterogeneous node or edge types, e.g., depots versus customers, the attention blocks are further duplicated across type-combinations while still sharing the same underlying parameters. Trained jointly across a set of routing, scheduling and classical graph problems, GOAL is only slightly inferior to specialized, single-problem baselines, while also being fine-tunable to new, previously unseen problems.

Outlook towards foundation models. Taken together, these works can be seen as first steps towards *foundation models* for combinatorial optimization, i.e., single models trained on a broad range of problems that can be applied to new tasks with little or no adaptation. The most recent efforts push further in this direction. Zhou et al. [16] propose URS, which replaces the explicit enumeration of VRP attributes with a unified data representation, allowing a single RL-trained constructive model to generalize zero-shot to over one hundred VRP variants, many of which feature attribute combinations never encountered during training. Nevertheless, current routing foundation models remain modest compared with their counterparts in natural language processing and computer vision, both in scale and in the diversity of problem structures they cover. In particular, they are predominantly trained on symmetric,

Euclidean instances. In fact, Chen et al. [105] observe that generalist solvers tend to degrade considerably on asymmetric variants unless the representation is explicitly designed to capture direction-dependent costs. Multigraphs are to our knowledge not supported by any existing generalist solver.

Multi-Objective Routing

Last, and especially relevant for this thesis, is the challenge concerning multi-objective extensions. Here, rather than optimizing a single scalar objective, a solution must trade off two or more competing objectives, such that the goal becomes approximating the Pareto set of non-dominated solutions instead of a single optimum.

Lin et al. [106] address this with PMOCO, an autoregressive method which adopts a hypernetwork that, conditioned on a preference vector specifying a particular trade-off, generates the parameters of an attention-model-style decoder. Trained via multi-objective RL over a distribution of preferences, a single PMOCO model thereby approximates the entire Pareto front, in contrast to prior approaches that train one model per trade-off weight, yielding impractically many models. Experimentally, PMOCO performs competitively compared to scalarization-based classical heuristics and evolutionary algorithms, both in terms of solution quality and runtime.

Chen et al. [107] identify that preference-based decomposition, when conditioned only on the instance, tends to produce solutions that cluster and repeat across nearby preferences. Their NHDE therefore introduces a heterogeneous graph attention mechanism relating the instance graph to a second graph encoding relations among the Pareto solutions found so far. Chen et al. also introduce an indicator-enhanced policy gradient that rewards improvements to a Pareto front quality indicator and a sampling strategy that actively seeks out multiple, diverse Pareto-optimal solutions per preference.

Similarly, Fan et al. [108] observe that such methods are typically conditioned on only the instance context, without jointly accounting for preference and problem size, and propose Conditional Neural Heuristic, which combines a dual-attention encoder relating the preference vector to the instance context with a size-aware decoder that explicitly encodes the instance size, allowing one trained model to generalize across both preferences and instance scales.

Most recently, Fan et al. [109] propose POCCO, which replaces the standard feed-forward layers with a conditional computation block that sparsely routes

each input to either a feed-forward expert or a lightweight identity expert depending on context. Fan et al. also train the resulting model with a preference-driven loss over pairs of winning and losing solutions for a given trade-off, thereby avoiding the need to specify an explicit multi-objective reward signal. POCCO obtains state-of-the-art results across bi- and tri-objective variants of the TSP and CVRP.

None of the methods above, nor to our knowledge any other learning-based method prior to Paper A, addresses routing on multigraphs in either single-objective or multi-objective settings. The omission is particularly notable in the multi-objective case, where multigraphs arise naturally. As discussed in Section 2.5, parallel edges typically represent non-dominated paths between two locations with respect to several attributes, e.g., travel time and distance. With a single objective and no side constraints, the best parallel edge between each node pair can be identified a priori, reducing the multigraph to a simple graph. With multiple objectives every parallel edge may be part of some Pareto-optimal route, so no simple reduction is possible without discarding parts of the Pareto front. Which edge to traverse then depends on the targeted trade-off and must be decided jointly with the node visit order. The next section describes directly why prior methods cannot readily be applied to multigraph routing.

4.7 Limitations of Prior Methods

Multigraph routing represents a gap in the neural combinatorial optimization literature. We expand on this next, focusing on limitations of prior methods in the encoding and decoding stages.

Encoding

The most widespread encoders are transformers operating on node coordinates [11, 12, 14, 22, 23, 66, 76, 102, 103, 106, 107, 108, 109], which restricts them to Euclidean VRPs where coordinates serve as node features. In principle, asymmetric graphs and multigraphs could be pre-processed to extract informative node features that are then encoded by such transformers. Nevertheless, this pre-processing is non-trivial and has thus far only been applied to asymmetric graphs, e.g., by Yi et al. [25]. It remains unclear whether a similar approach

could be designed for multigraphs.

To handle asymmetric instances, several methods instead condition their attention-type layers on a distance matrix, either through mixed-score attention [15, 24, 101] or the related Adaptation Attention Free Module [16, 27, 110]. On multigraphs, however, such a matrix is ill-defined, since a node pair may be joined by several parallel edges with distinct attributes and no canonical scalar entry exists. One could retain this information in vector-valued entries, and certain layer types can indeed consume such multivariate distance matrices. Yet, forming these entries by concatenating parallel-edge features imposes an arbitrary edge ordering, thereby breaking edge permutation invariance, and assumes a fixed number of edges between each node pair. A cleaner alternative is to aggregate the parallel edges into compact node-pair features with a permutation-invariant pooling component, which can then condition standard mixed-score attention layers. This is precisely the approach we propose in Paper B to form latent distance matrices and thereby extend several common architectures to the multigraph setting.

GNN-based encoders, most prominent in the NAR paradigm [56, 68, 78, 84, 85, 111], are often more naturally suited to multigraphs, since many variants retain edges as separate entities and propagate edge embeddings. Nevertheless, most of these architectures still presuppose a unique notion of adjacency or unique edge features between each node pair. For example, the GCN of Joshi et al. [68] and the anisotropic GNN of Sun and Yang [56] require unique edges between node pairs and are therefore not natively transferable to multigraphs. Extending them might not be inherently difficult, but requires deliberate modifications that have not yet been undertaken.

In summary, encoding presents challenges, yet it is rarely an insurmountable obstacle. With sufficient adaptation, many architectures could likely embed multigraphs. The decisive limitation often lies elsewhere. Indeed, even the methods whose encoders are directly transferable, e.g., GREAT of Lischka et al. [57], which treats edges as separate entities and does not assume uniqueness between node pairs, typically fail in the decoding stage instead.

Decoding

Decoding is where the applicability of prior methods to multigraphs is most limited. In essence, this is because a multigraph route is not uniquely described by a node permutation, but additionally requires selecting which parallel edge

to traverse between consecutive nodes.

This limitation is most evident for autoregressive methods, which are almost exclusively node-based, i.e., they construct a solution by sequentially outputting a node permutation [11, 12, 15, 22, 23, 24, 25, 57, 76, 101]. Adapting them to instead construct a sequence of edge selections is non-trivial in general and requires substantial alterations.

A notable exception is the edge-based construction of Zhou et al. [112], released concurrently with Paper A, which is in principle extendable to multi-graph routing with few modifications. However, it was originally designed for asymmetric multi-objective routing and, to our knowledge, has not been applied to multigraph VRPs.

Rather than modifying the decoder, one could transform the multigraph into a simple graph using a deterministic rule, so that node-based AR construction becomes *directly* applicable. Figure 4.4 shows two such transformations, namely taking the line graph of the original graph (left) and introducing a virtual node at the midpoint of every original edge (right). While both are possible in theory, they significantly inflate the size of the representation and thereby the computational complexity. Moreover, routing in a transformed graph arguably diminishes interpretability, and both transformations require altered encoding, masking and decoding procedures. For example, the second transformation turns the TSP on the original multigraph into a *Steiner TSP*, where only a subset of nodes must be visited.

Improvement-based methods face an analogous difficulty, as the local move operators they use to refine a solution are almost exclusively defined over node permutations. Examples include the 2-opt moves of Wu et al. [87] and Costa et al. [88], the generalized k-opt exchanges of Ma et al. [92] and the CROSS-exchange of Kim et al. [91], where a 2-opt move reverses a segment of the node sequence and a CROSS-exchange swaps sub-sequences of nodes between routes. On a multigraph, such a move does not determine a unique successor solution, since it leaves open which of the parallel edges connecting the reconnected nodes should be traversed. Both the operators and the policies selecting among them would thus need to be augmented to jointly choose a reconnection and its associated parallel edges, considerably enlarging the action space. Moreover, improvement-based methods require an initial solution, typically produced by a construction heuristic, and thereby inherit the node-based construction difficulties discussed above.

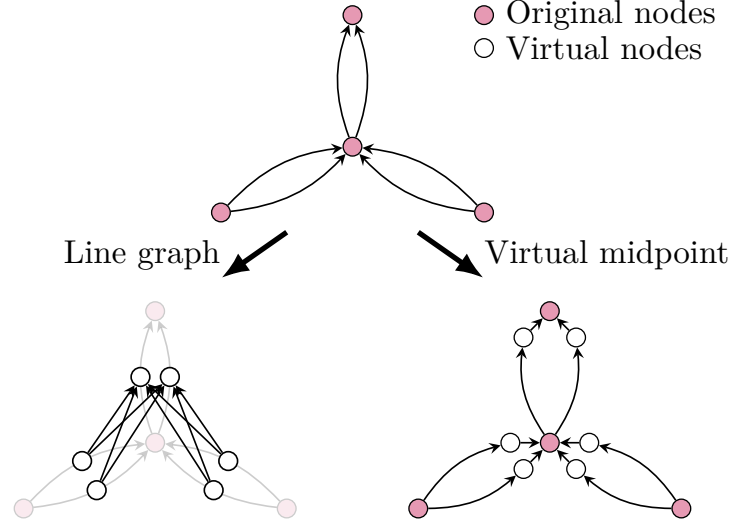


Figure 4.4: Two possible graph transformations to ensure node-based construction is applicable to multigraph VRPs. For simplicity, the displayed original graph is not complete.

Non-autoregressive methods, in contrast, come with decoding schemes that transfer more directly to multigraph routing. Since their heatmaps score the edges of the instance graph, they can in principle assign a separate score to each parallel edge. The subsequent decoding, e.g., greedy selection, beam search or heatmap-guided local search, then operates over edges and can therefore discriminate among parallel edges. Nevertheless, this transferability often breaks down in the details. As noted in the encoding discussion, several heatmap generators and their decoders presuppose a unique edge between each node pair, e.g., through adjacency-matrix-based message passing, and require adaptation before parallel edges can be represented. Even where the heatmap accommodates parallel edges, the feasibility masking applied during decoding must be extended to reason about edge selections. Moreover, it remains an open question whether decoding procedures tuned for simple graphs exploit multigraph heatmaps as effectively.

Gap Summary

The discussion above shows that most prior methods require substantial and challenging adaptation to handle multigraph routing. Autoregressive methods

are limited primarily by their decoding and improvement-based methods by their operators and initialization. Non-autoregressive methods are the most naturally edge-centric, yet still presuppose structures that must be adapted before parallel edges can be represented. Moreover, for existing methods that could in principle be extended, there is no guarantee their performance would transfer cleanly. Consequently, Papers A and B design methods specifically for multigraph routing problems, filling a clear gap in the literature. The next chapter presents the proposed methods, showing how they differ in their overall solution strategy and handling of the parallel edge structure.

CHAPTER 5

Learning to Route on Multigraphs: Challenges and Methodology

Section 2.5 introduced multigraphs and discussed the prominence of this representation in operations research. Section 4.7 further argued that prior learning-based methods are unsuitable for VRPs on multigraphs. This chapter now continues by describing our proposed approaches. It starts with an outline of why the multigraph setting is computationally difficult, motivating our decomposition-based approaches. Then, the general idea behind the methods from Papers A and B is described. The methodological details are deferred to the papers themselves, enclosed later in the thesis.

5.1 Challenges of Multigraph Routing

Learning-based routing on multigraphs raises challenges from both an optimization and a computational perspective. In the simple-graph setting of Chapter 4, a solution is fully specified by a permutation of the nodes, since exactly one edge connects each visited pair. On a multigraph this is no longer the case, since a route is determined jointly by the order in which nodes are

visited and by which of the parallel edges is traversed between consecutive nodes. This coupling is the source of the difficulties discussed below.

First, edge selection is non-trivial in general. Because parallel edges encode trade-offs, for instance between travel time and distance, the appropriate choice generally depends on the remaining route and, in the multi-objective case, on the region of the Pareto front one aims to reach. Fixing the edges a priori, or selecting them greedily according to a single attribute, therefore tends to underperform. In fact, the value of the multigraph representation comes from choosing the paths to traverse jointly with the routing decisions [19]. Even when the node order is fixed, selecting the optimal edges often remains an $\mathcal{NP}$ -hard problem, known as the *Fixed Sequence Arc Selection Problem* (FSASP) [20]. Its hardness stems from global resource constraints, such as time windows or a route-duration limit, that couple the arc choices across legs and prevent them from being selected independently. Simple rules and heuristics offer no guarantee of good performance, which motivates learning the edge selection policy rather than hand-designing it.

Second, the action space of the routing problem expands substantially. In simple-graph autoregressive construction, a decoding step selects one of at most N candidate nodes. On a multigraph, the model must additionally select one of the M parallel edges, enlarging the per-step action space accordingly. When edge and node decisions are made jointly, as in edge-based construction, this enlargement compounds over the length of the tour and complicates both decoding and credit assignment during training.

Third, the multigraph geometry induces additional computational cost. Representing every edge explicitly is inherently quadratic in the number of nodes. A multigraph with N nodes and M parallel edges per pair requires $\mathcal{O}(MN^2)$ storage, in contrast to the $\mathcal{O}(N)$ footprint of node-based representations. Methods that maintain this dense representation throughout encoding and decoding therefore incur a large memory overhead, which constitutes a major obstacle to scaling learning-based multigraph routing.

Taken together, the difficulty of edge selection, the enlarged action space and the quadratic representation cost motivate the decomposition-based approaches examined in the next section.

5.2 Design Space and Methodology

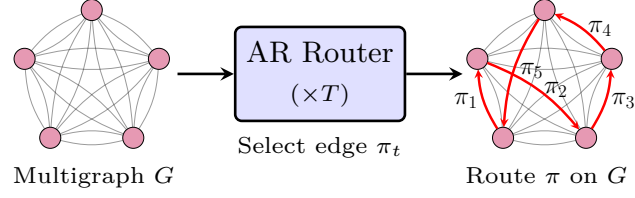
This section presents the three methods proposed in this thesis and situates them within a common design space. Paper A contributes two models, collectively referred to as GNN-based Multigraph Solver – GMS: Edge-Based GMS and Dual-Head GMS. Paper B contributes a third method, Node-Edge Policy Factorization – NEPF. All three share an autoregressive core for constructing routes, but differ in how they handle the parallel-edge dimension that distinguishes multigraphs from simple graphs. The resulting construction paradigms are illustrated in Figure 5.1.

The design space is organized around two blocks. AR construction builds a solution incrementally, conditioning each decision on the partial route. It is expressive and well suited to the sequential nature of routing, but its step-by-step decoding is comparatively slow. NAR construction instead produces its output in a single pass, which is fast but coarser, as it cannot condition on decisions made earlier in the same stage. Following GLOP [79], we exploit this complementarity, delegating to a NAR component the decisions for which a quick, global estimate suffices while reserving AR for the parts of the problem that require fine-grained, sequential reasoning. The three methods differ in how they place these components. GMS-EB performs edge-based construction end-to-end with a single AR model (Figure 5.1a). GMS-DH first prunes the multigraph with a NAR stage and then routes autoregressively on the resulting simple graph (Figure 5.1b). Finally, NEPF reverses this order, routing autoregressively over the nodes before selecting edges with a NAR stage (Figure 5.1c).

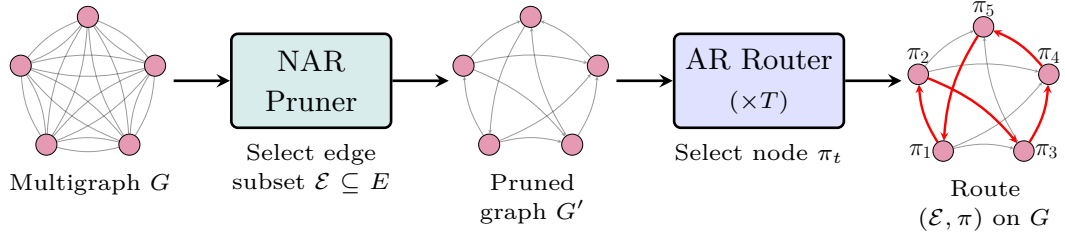
In all three methods the core routing model is autoregressive, in line with much recent work on, for example, foundation models for routing [15, 16] and neural architectures for constraint handling [22, 23, 113]. Retaining an AR routing backbone keeps our methods compatible with these developments. We do not consider fully end-to-end NAR construction or improvement-based methods (see Chapter 4) and leave their extension to the multigraph setting to future work.

Unlike the appended papers, whose notation differs in places, the three methods are presented here with common notation that follows the conventions used elsewhere in the thesis (mostly matching Paper B). Throughout, G denotes the multigraph instance and $\pi = (\pi_1, \dots, \pi_T)$ a sequence of visited nodes. The one exception concerns GMS-EB, whose route is a sequence π of *edges*.

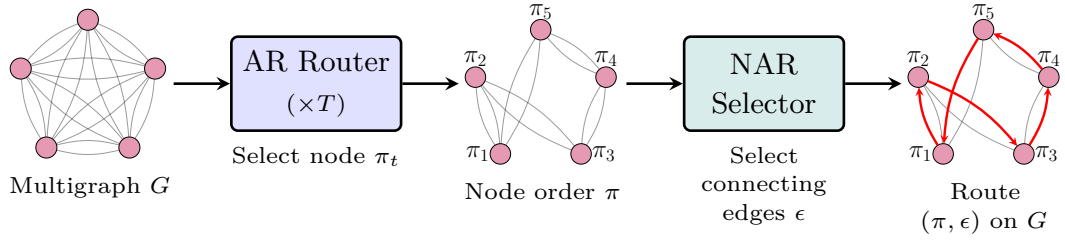
(a) GMS-EB



(b) GMS-DH



(c) NEPF



Autoregressive Non-autoregressive

Figure 5.1: The three construction paradigms in this thesis, represented by the three methods GMS-EB, GMS-DH and NEPF.

End-to-End Edge-Based AR Construction: GMS-EB

The first method, GMS-EB (Figure 5.1a), operates directly on the multigraph. Its encoder is a stack of GREAT layers [57], which are edge-based and therefore naturally accommodate parallel edges. The decoder is an edge-based variant of the multi-pointer decoder [66]. At construction step t , given the instance G and the partial route $\pi_{1:t-1}$, it selects the next edge π_t (and thereby implicitly the next node) from the edges leaving the current node. The resulting policy factorizes the tour distribution autoregressively as

$$p_\theta(\pi \mid G) = p_\theta(\pi_1 \mid G) \prod_{t=2}^T p_\theta(\pi_t \mid \pi_{1:t-1}, G). \quad (5.1)$$

In the multi-objective scenario, the preference λ enters only through the decoder, whose weights are generated by an MLP hypernetwork, such that $\theta(\lambda) = [\theta_{\text{enc}}, \theta_{\text{dec}}(\lambda)]$ with $\theta_{\text{dec}}(\lambda) = \text{MLP}(\lambda)$. This leaves the encoder preference-agnostic, ensuring it must only be run once per instance regardless of the number of evaluated preferences.

GMS-EB is trained with the REINFORCE scheme (3.15), which in the multi-objective setting is applied to the Chebyshev-scalarized reward $R_\lambda = -g_\lambda(C(\pi, G))$, with one preference λ sampled per batch. The baseline is the POMO shared baseline, which averages the rewards of the $\mathcal{O}(N)$ routes sampled for the same instance.

The main advantage of this edge-based design is its conceptual simplicity. A single model constructs the route in one stage and because decoding acts on the multigraph directly, each edge choice in (5.1) is conditioned on the full state given by the partial tour. Empirically, GMS-EB performs particularly well in the multi-objective setting. However, its main drawback is computational. A multigraph with N nodes and M parallel edges per pair has $\mathcal{O}(MN^2)$ edges, all of which must be embedded and stored. Since the decoder is then invoked $\mathcal{O}(T)$ times per tour and $\mathcal{O}(N)$ POMO rollouts [12] are performed, edge-based decoding scales as $\mathcal{O}(MN^2) \cdot \mathcal{O}(N) \cdot \mathcal{O}(T) = \mathcal{O}(MN^3T)$, against $\mathcal{O}(N^2T)$ for a node-based decoder. A thorough complexity analysis is given in Paper B (Appendix C). This cost is the central obstacle to scaling GMS-EB.

NAR Pruning then AR Routing: GMS-DH

GMS-DH (Figure 5.1b) retains the GREAT encoder but inserts a non-autoregressive selection head before the final encoder layer. Using the intermediate edge embeddings, this head assigns to each parallel edge $l \in E_{uv}$, where E_{uv} is the set of connecting edges between nodes $u, v \in V$, a probability $q_{\tilde{\theta}}(l \mid u, v, G)$ of being retained. One edge is then sampled per node pair in a single shot, non-autoregressively. Treating the pairs independently, the joint selection $\mathcal{E}$, which contains exactly one edge per pair, has probability

$$q_{\tilde{\theta}}(\mathcal{E} \mid G) = \prod_{l \in \mathcal{E}} q_{\tilde{\theta}}(l \mid \text{start}(l), \text{end}(l), G). \quad (5.2)$$

The retained edges are fed back and aggregated into node embeddings through the final GREAT layer and a small number of transformer layers, producing an effectively simple graph. A node-based MP routing decoder then constructs the tour autoregressively, now over nodes and conditioned on the selection $\mathcal{E}$:

$$p_{\theta}(\pi \mid G, \mathcal{E}) = p_{\theta}(\pi_1 \mid G, \mathcal{E}) \prod_{t=2}^T p_{\theta}(\pi_t \mid \pi_{1:t-1}, G, \mathcal{E}), \quad (5.3)$$

where π_t denotes a node.

The two heads are trained jointly with a hierarchical REINFORCE scheme in which the selection head is rewarded by the quality of the best tour its choice enables: $R^{(1)}(\mathcal{E}, G) = R(\mathcal{E}, \tilde{\pi}(\mathcal{E}), G)$ with $\tilde{\pi}(\mathcal{E})$ the optimal tour on the pruned graph, approximated in practice by the best of K sampled node sequences, $R^{(1)}(\mathcal{E}, G) \approx \max_{k \leq K} R(\mathcal{E}, \pi_k, G)$. The routing head is in turn trained on the reward of its own tours, in the same way as GMS-EB, and both heads use shared baselines that average the rewards over several edge selections per instance and over the K routes per edge selection, respectively. Again similarly to GMS-EB, the multi-objective case considers preference-based scalarization by generating decoder weights with MLP hypernetworks.

Because routing is carried out by a node-based decoder, GMS-DH is substantially more efficient than GMS-EB. GMS-DH also approximates GMS-EB well in terms of solution quality, even outperforming it in the single-objective setting (see Paper B). The decomposition, however, introduces new difficulties. The selection stage faces a form of the curse of dimensionality: it selects one edge for each of the $\mathcal{O}(N^2)$ node pairs simultaneously, yet most of these choices do not affect the final route, which complicates credit assignment and can

destabilize training. Pruning also still requires embedding the entire multigraph through the encoder, so its $\mathcal{O}(MN^2)$ memory footprint is not avoided. Finally, by construction, the selection (5.2) is non-autoregressive and precedes routing. Consequently, unlike in GMS-EB, the edge choice cannot depend on the partial tour and must be fixed before rollout begins.

AR Routing then NAR Edge Selection: NEPF

NEPF (Figure 5.1c) reverses the order of the two stages. A route is written as a node sequence $\pi = (\pi_1, \dots, \pi_T)$ together with a sequence of connecting edges $\epsilon = (\epsilon_1, \dots, \epsilon_{T-1})$, $\epsilon_t \in E_{\pi_t \pi_{t+1}}$, and the joint policy is factorized as

$$p_\theta(\pi, \epsilon \mid G) = p_{\theta_1}^{\text{node}}(\pi \mid G) p_{\theta_2}^{\text{edge}}(\epsilon \mid \pi, G). \quad (5.4)$$

The second factor solves the fixed sequence arc selection problem. That is, for a fixed π , it aims to choose the connecting edges

$$\epsilon^* = \arg \min_{\epsilon_t \in E_{\pi_t \pi_{t+1}}} C(\pi, \epsilon, G), \quad (5.5)$$

where $C(\pi, \epsilon, G)$ denotes the cost of (π, ϵ) .

To keep the node permutation stage cheap, a pre-encoding module summarizes each parallel-edge set E_{uv} into a compact latent vector through a permutation-invariant Deep Sets aggregation [70],

$$D_{uv} = \rho\left(\sum_{l \in E_{uv}} \phi(z_l)\right), \quad (u, v) \in V^2, \quad (5.6)$$

with z_l a linear embedding of the edge attributes and ρ, ϕ MLPs. Consequently, the encoder operates on the latent distance matrix $D \in \mathbb{R}^{N \times N \times d}$ rather than the explicit $\mathcal{O}(MN^2)$ multigraph. This encoder uses the same GREAT and transformer-based design as GMS-DH, followed by an MP decoder that produces π autoregressively. The edge stage then solves (5.5) non-autoregressively by pooling each edge set, mixing the sequence with a bidirectional LSTM, scoring the edges with a multi-pointer mechanism and sampling one edge per position in parallel.

The stages are trained jointly with a hierarchical REINFORCE scheme, mirroring GMS-DH with the roles of the two stages reversed, so that the node stage is now rewarded by the best of K sampled edge selections. Multi-objectivity is handled via preference-conditioned decoding and a Chebyshev-scalarized reward.

This stage ordering is efficient in both memory and time, since the parallel edges are collapsed initially and never need to be materialized in full during node sequence construction. The edge selection stage also only handles the $\mathcal{O}(MT)$ edges that lie along the chosen route, rather than the $\mathcal{O}(MN^2)$ edges of the whole multigraph. Moreover, because the interface to the node stage backbone is the learned distance matrix D , the framework is *backbone-agnostic* and integrates with a range of routing models. Empirically, NEPF tends to outperform both GMS-EB and GMS-DH in solution quality while being markedly faster to train and run (see Paper B). The price of the factorization (5.4) is that node construction must proceed without knowing which edges will ultimately be traversed, that is, with only partial state information. This matters for problems with time windows or resource limits, where the current time or resource level depends on the edges used. NEPF addresses this with an auxiliary RNN-based state estimator that reconstructs an estimate $\hat{s}_t$ of the state from the partial route $\pi_{1:t-1}$.

To summarize the three methods, whose main differences are collected in Table 5.1, GMS-EB establishes that learning-based routing on multigraphs is feasible and effective. However, it carries a computational cost that does not scale well. GMS-DH and NEPF recover efficiency by decomposing the problem into separate routing and edge selection stages. The two methods differ in the order in which the two stages are solved, with NEPF’s node-first factorization proving the most scalable and, in our experiments, the strongest overall. Jointly, the three methods address the research questions posed in Chapter 1. GMS-EB shows that neural solvers can route on the full multigraph (RQ1), GMS-DH and NEPF show that decomposition makes this scale (RQ2) and the comparison between them indicates how the two stages are best ordered (RQ3). The following chapter summarizes the two appended papers, before Chapter 7 concludes.

Table 5.1: Comparison of the three proposed methods. Decoding time complexities include the $\mathcal{O}(N)$ POMO rollouts and assume $T \approx N$ steps.

	GMS-EB	GMS-DH	NEPF
Paper	A	A	B
Stages	One joint stage over edges	Edge pruning + node routing	Node routing + edge selection
Construction	AR	NAR, then AR	AR, then NAR
Encoded input	Full multigraph, $\mathcal{O}(MN^2)$	Full multigraph, $\mathcal{O}(MN^2)$	Latent matrix D , $\mathcal{O}(N^2)$
Edges scored	$\mathcal{O}(MN^2)$, step by step	$\mathcal{O}(MN^2)$, in one shot	$\mathcal{O}(MN)$, in one shot
Routing decoding time complexity	$\mathcal{O}(MN^4)$	$\mathcal{O}(N^3)$	$\mathcal{O}(N^3)$
Edge choice context	Partial route	Instance only	Full node sequence
Node choice includes edge context	Yes	Yes, via pruning	No, estimated state
Backbone-agnostic	No, edge-based required	No, edge-based required	Yes

CHAPTER 6

Summary of Included Papers

This chapter provides a summary of the included papers.

6.1 Paper A

F. Rydin, A. Lischka, J. Wu, M. Haghiri Chehreghani, B. Kulcsár
Beyond Simple Graphs: Neural Multi-Objective Routing on Multigraphs
14th International Conference on Learning Representations (ICLR),
Rio de Janeiro, Brazil, Apr. 2026.
To appear in conference proceedings.
Available online: <https://arxiv.org/abs/2506.22095>.

This paper introduces two RL-based neural methods for multi-objective routing on multigraphs, collectively named GNN-based Multigraph Solver. Both are autoregressive, construction-based methods, but they differ in their action outputs, in how they decompose the multigraph routing problem and in their experimental runtime–quality trade-off.

First, the paper presents Edge-Based GMS, which performs routing in an end-to-end edge-based manner. GMS-EB encodes problem instances with the

GREAT architecture [57], configured to output edge embeddings, after which a multi-pointer-type decoder [66] autoregressively selects the next edge to traverse. Unlike the node-based construction of prior AR methods, this edge-based construction enables GMS-EB to tackle multigraph instances directly. However, since the total number of edges grows quadratically with the node count, edge-based decoding becomes a space- and time-complexity bottleneck and GMS-EB consequently exhibits high computational cost and poor scaling behavior.

Second, the paper introduces Dual-Head GMS, which aims to match the quality of GMS-EB at a substantially lower computational cost. GMS-DH first prunes the multigraph into a simple graph using a non-autoregressive learned policy, parameterized by one decoder head. It then re-encodes the simple graph with one final GREAT layer and a few transformer layers, after which the second decoder head constructs the node order autoregressively. This effectively splits the policy into two parts: a non-autoregressive edge component and an autoregressive node component.

Both methods use preference-based scalarization to decompose multi-objective VRPs into a set of single-objective subproblems. Concretely, GMS is trained with variants of the multi-objective REINFORCE algorithm [106], which samples preferences within each training batch and uses the negative Chebyshev cost as reward. At inference time, the preference grid is swept and a solution is obtained for each trade-off, yielding an approximation of the Pareto front. Architecturally, the preference vector is fed to hypernetworks [114] that generate the decoder weights of GMS-EB and GMS-DH. Since the encoder is preference-agnostic, it need only be run once per problem instance, improving computational efficiency.

Experimentally, the paper validates GMS on three multi-objective problems on multigraphs, namely the multi-objective TSP, CVRP and a variant of the TSP with time windows. Across all three, GMS obtains competitive hypervolume relative to classical single-objective heuristics such as LKH and HGS, multi-objective evolutionary algorithms and neural baselines not designed for multigraph inputs. GMS-DH stays within about 2% hypervolume of LKH on the multigraph TSP and within 5% of HGS on the multigraph CVRP, while solving 200 instances with 100 nodes in about a minute, compared to roughly 30 minutes for LKH and 75 hours for HGS. On the time-window variant, where it is less clear a priori which edges can be pruned, both variants

outperform all baselines, with the strongest baseline trailing by at least 10% in hypervolume. Comparing the two variants, GMS-EB tends to produce higher-quality solutions but at a prohibitively high computational cost on large graphs, whereas GMS-DH underperforms slightly while running at a much lower cost.

F. Rydin: idea conceptualization, literature review, implementation, validation and writing of the original draft.

A. Lischka: idea conceptualization, literature review, implementation and draft review.

J. Wu, M. Haghiri Chehreghani, B. Kulcsár: supervision, acquisition of funding and compute resources, idea conceptualization and draft review.

6.2 Paper B

F. Rydin, M. Haghiri Chehreghani, B. Kulcsár

Two-Stage Learned Decomposition for Scalable Routing on Multigraphs
Submitted for publication as conference paper, 2026.

Preprint available online: <https://arxiv.org/abs/2605.05389>.

This paper introduces Node-Edge Policy Factorization, a third neural method for routing on multigraphs. The central idea is to adopt a classical decomposition from operations research [20] within a modern neural routing model. Instead of constructing routes directly on the dense multigraph, NEPF factorizes the routing policy into a node permutation stage that fixes the visiting order and an edge selection stage that then chooses one connecting edge per pair of consecutive nodes on the route. The two stages are modeled by separate policies and trained jointly with hierarchical reinforcement learning. This factorization is motivated by the scalability issues of the prior multigraph methods, which maintain the full $\mathcal{O}(MN^2)$ multigraph representation throughout encoding and couple decoding tightly to the multigraph structure, preventing them from scaling to high node counts and many parallel edges.

The node permutation stage is designed to avoid processing the dense multigraph as much as possible. To this end, the paper proposes a pre-encoding aggregation mechanism that summarizes each set of parallel edges into a compact latent representation using Deep Sets [70], yielding a permutation-invariant latent distance matrix. This matrix is encoded with the GREAT

architecture [57] and a small number of transformer layers (the same base encoder as GMS-DH). An autoregressive multi-pointer decoder [66] then constructs the node order. Importantly, the stage is backbone-agnostic, requiring only a function that maps the latent distance matrix to a node permutation, which makes NEPF compatible with a wide range of existing routing architectures, unlike both variants of GMS.

The second stage solves the fixed sequence arc selection problem. Given the fixed node sequence, it selects one edge per connecting edge set. Edge features are pooled per set and mixed with a bidirectional LSTM to provide full route context. A multi-pointer mechanism then scores the edges within each set and edges are sampled non-autoregressively in a single shot, allowing multiple candidate edge selections to be drawn cheaply. Although shallow, this stage is shown to be robust across problems while adding little latency over independent heuristic edge selection.

Both stages are trained jointly with a hierarchical REINFORCE algorithm [73]. The edge policy optimizes local decisions conditioned on a fixed node sequence, while the node policy is trained on the best downstream reward achievable over multiple edge samples. In the multi-objective setting, following Lin et al. [106] and the approach in Paper A, MLP hypernetworks condition the decoding of both stages on a sampled preference vector, so that a single model approximates the Pareto front under Chebyshev scalarization without retraining or recomputing node encodings per preference.

The paper evaluates NEPF across six VRP variants, including two single-objective problems (a resource-constrained TSP and a multi-resource OP) and four multi-objective problems (multi-objective TSP, CVRP, TSPTW and a bi-objective OP), against classical baselines such as Gurobi, LKH, HGS and OR-tools, as well as GMS-EB and GMS-DH. Among the neural methods, NEPF performs best in 12 of 16 multi-objective and 5 of 8 single-objective configurations, while reducing inference time by one to two orders of magnitude relative to GMS-EB and by a factor of 2–3 compared to GMS-DH. With augmentation, NEPF also comes within 0.7–2.2% hypervolume of LKH and HGS on the MOTSP and MOCVRP, while running 4–5 times faster than LKH and about 700 times faster than HGS. On the single-objective problems, Gurobi retains better solution quality but requires 4–16 hours per 10 000 instances, compared to under a minute for NEPF. The paper further demonstrates that the framework retains its performance under several alternative node permuta-

tion backbones, scales more favorably with node count and parallel edges than GMS and generalizes to larger instances.

F. Rydin: idea conceptualization, literature review, implementation, validation and writing of the original draft.

M. Haghiri Chehreghani, B. Kulcsár: supervision, acquisition of funding and compute resources, idea conceptualization and draft review.

Concluding Remarks and Future Work

7.1 Conclusion

This thesis has developed, to our knowledge, the first learning-based methods designed specifically for vehicle routing problems on multigraphs. Across the two appended papers we introduced three neural construction solvers, GMS-EB, GMS-DH and NEPF, and evaluated them on a range of routing problems in both single- and multi-objective settings. Instead of removing parallel edges with a fixed rule before routing, for example by keeping the shortest edge per node pair, these methods learn which of the parallel edges to traverse as part of the routing policy. Overall, the thesis extends neural combinatorial optimization to a more realistic problem class and thereby takes a step towards narrowing the gap between the idealized settings on which the field has largely been developed and routing problems encountered in practice.

The three methods are best understood as points in a shared design space, in which the multigraph routing policy is decomposed into a node ordering component and an edge selection component. Each component is realized either autoregressively or non-autoregressively. GMS-EB establishes feasibility by constructing routes end-to-end on the explicit multigraph and attains

solution quality competitive with established operations-research heuristics and multi-objective evolutionary baselines, answering RQ1. Nevertheless, its explicit $\mathcal{O}(MN^2)$ treatment of the parallel edges scales poorly. This motivates the staged decompositions of the remaining two methods, which address RQ2 and RQ3. GMS-DH takes a first step in this direction by prefixing a NAR pruning stage to the AR router, while NEPF reverses the order, routing over nodes first and selecting edges non-autoregressively. This last design proves the most effective overall of the three. By outsourcing the multigraph to shallow pre- and post-processing components, NEPF avoids much of the scalability penalty inherent to explicit edge-based construction, scaling more favorably in both the node count N and the number of parallel edges M while retaining, and often improving upon, the solution quality of the direct approach.

7.2 Limitations and Future Work

A number of limitations remain, indicating promising directions for future work.

First, none of the proposed methods is strictly superior along all axes. GMS-EB routes directly on the multigraph and is conceptually the most faithful to the problem, but its explicit edge representation scales poorly and renders it impractical on larger instances. NEPF, conversely, scales considerably better and is the strongest method overall in our experiments, yet attains this by factorizing node ordering and edge selection into separate stages. It thereby loses the joint reasoning of the direct formulation. The three methods thus trade fidelity against scalability and the right method to use in a practical setting likely depends on the problem at hand.

Second, in common with most neural construction solvers our methods provide no optimality guarantees. They tend to produce high-quality solutions, but cannot quantify how far a given solution lies from the optimum, which limits their applicability in settings where guarantees are required. Arguably, however, this limitation can be mitigated. Because the methods construct feasible solutions quickly, they are naturally suited for warm-starting exact or improvement-based procedures, yielding strong initial solutions from which guarantee-providing methods can proceed. Coupling neural construction with such methods, to combine the speed of the former with the guarantees of the latter, is a compelling direction for future work.

Third, our benchmark suite consists mainly of comparatively simple problem settings. We do not consider combinations of constraints, nor stochastic or time-dependent costs and demands, all of which are prevalent in real logistics. Moreover, our instances are mostly synthetically generated rather than drawn from real road networks and consequently only approximate the structure of realistic multigraph routing problems. The extent to which our findings transfer to such settings therefore remains to be established. Doing so constitutes an important next step.

Fourth, this thesis studies multigraph routing in isolation, taking the multigraph as given. The thesis does not address the end-to-end pipeline required to construct multigraphs. In practice the parallel edges must first be computed, typically by solving a multi-objective shortest-path problem between each pair of locations to extract the relevant non-dominated connections. This construction step is often costly, since even the bi-objective shortest-path problem is intractable [45]. A full accounting of multigraph routing must therefore include the pre-processing cost and a natural extension of this thesis is to treat construction and routing jointly rather than in sequence, so that the two stages can inform one another.

Beyond the future work suggested by the limitations, another direction concerns the neural architecture. The results in this thesis indicate that shifting work onto non-autoregressive components is an effective route to scalability. It would be natural to push this further, towards fully non-autoregressive formulations of multigraph routing and possibly towards improvement-based methods. Combining such formulations with reinforcement learning, as in the construction methods studied here, is a particularly promising avenue.

To further improve scalability one could also integrate the proposed methods with partition-focused strategies, which first split a large instance into smaller subproblems, e.g., containing subsets of nodes, before solving each in turn [79]. This would enable extending neural multigraph routing to substantially larger instances than considered here.

Finally, and most ambitiously, a long-term goal of the field is to develop foundation-type models: single, broadly generalizable solvers that can handle a wide range of combinatorial optimization problems without per-problem retraining. Incorporating the multigraph representation into such models would be a valuable step towards general neural solvers that reflect the structure of real-world routing.

References

- [1] Swedish Post and Telecom Authority (PTS), “The Swedish Parcel Market 2024,” Swedish Post and Telecom Authority (PTS), Tech. Rep., 2024, <https://pts.se/>.
- [2] Capgemini Research Institute, “The Last-Mile Delivery Challenge,” Capgemini, Tech. Rep., 2019.
- [3] H. Ritchie, *Cars, Planes, Trains: Where Do CO₂ Emissions from Transport Come From?* <https://ourworldindata.org/co2-emissions-from-transport>, 2020.
- [4] B. Korte and J. Vygen, *Combinatorial Optimization*, 6th ed., Springer Berlin, Heidelberg, 2018.
- [5] G. B. Dantzig and J. H. Ramser, “The Truck Dispatching Problem,” *Management Science*, vol. 6, no. 1, pp. 80–91, 1959.
- [6] P. Toth and D. Vigo, *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, 2002.
- [7] A. Dosovitskiy, L. Beyer, A. Kolesnikov, et al., “An Image is Worth 16×16 Words: Transformers for Image Recognition at Scale,” in *International Conference on Learning Representations*, 2021.
- [8] T. B. Brown et al., “Language Models are Few-Shot Learners,” in *Advances in Neural Information Processing Systems*, 2020.
- [9] O. Vinyals, M. Fortunato, and N. Jaitly, “Pointer Networks,” in *Advances in Neural Information Processing Systems*, 2015.

- [10] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, “Neural Combinatorial Optimization with Reinforcement Learning,” in *International Conference on Learning Representations*, 2017.
- [11] W. Kool, H. van Hoof, and M. Welling, “Attention, Learn to Solve Routing Problems!” In *International Conference on Learning Representations*, 2019.
- [12] Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min, “POMO: Policy Optimization with Multiple Optima for Reinforcement Learning,” in *Advances in Neural Information Processing Systems*, 2020.
- [13] F. Zhou, A. Lischka, B. Kulcsár, J. Wu, M. Haghiri Chehreghani, and G. Laporte, “Learning for Routing: A Guided Review of Recent Developments and Future Directions,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 202, p. 104 278, 2025.
- [14] J. Zhou, Z. Cao, Y. Wu, W. Song, Y. Ma, J. Zhang, and C. Xu, “MVMoE: Multi-Task Vehicle Routing Solver with Mixture-of-Experts,” in *International Conference on Machine Learning*, 2024.
- [15] D. Drakulic, S. Michel, and J.-M. Andreoli, “GOAL: A Generalist Combinatorial Optimization Agent Learner,” in *International Conference on Learning Representations*, 2025.
- [16] C. Zhou, C. Yu, S. Yao, X. Lin, Z. Wang, Y. Zhou, and Q. Zhang, “URS: A Unified Neural Routing Solver for Cross-Problem Zero-Shot Generalization,” in *International Conference on Machine Learning*, 2026.
- [17] R. Bommasani et al., “On the Opportunities and Risks of Foundation Models,” *arXiv preprint*, 2021, arXiv:2108.07258.
- [18] B. Boyacı, T. H. Dang, and A. N. Letchford, “Vehicle Routing on Road Networks: How Good is Euclidean Approximation?” *Computers & Operations Research*, vol. 129, p. 105 197, 2021.
- [19] H. Ben Ticha, N. Absi, D. Feillet, and A. Quilliot, “Empirical Analysis for the VRPTW with a Multigraph Representation for the Road Network,” *Computers & Operations Research*, vol. 88, pp. 103–116, 2017.

-
- [20] T. Garaix, C. Artigues, D. Feillet, and D. Josselin, “Vehicle Routing Problems with Alternative Paths: An Application to On-Demand Transportation,” *European Journal of Operational Research*, vol. 204, no. 1, pp. 62–75, 2010.
 - [21] M. Behnke, T. Kirschstein, and C. Bierwirth, “A Column Generation Approach for an Emission-Oriented Vehicle Routing Problem on a Multigraph,” *European Journal of Operational Research*, vol. 288, no. 3, pp. 794–809, 2021.
 - [22] J. Bi, Y. Ma, J. Zhou, W. Song, Z. Cao, Y. Wu, and J. Zhang, “Learning to Handle Complex Constraints for Vehicle Routing Problems,” in *Advances in Neural Information Processing Systems*, 2024.
 - [23] J. Bi, Z. Cao, J. Zhou, W. Song, Y. Wu, J. Zhang, Y. Ma, and C. Wu, “Towards Efficient Constraint Handling in Neural Solvers for Routing Problems,” in *International Conference on Learning Representations*, 2026.
 - [24] Y.-D. Kwon, J. Choo, I. Yoon, M. Park, D. Park, and Y. Gwon, “Matrix Encoding Networks for Neural Combinatorial Optimization,” in *Advances in Neural Information Processing Systems*, 2021.
 - [25] H. Yi, Z. Huang, Y. Ma, and Z. Cao, “RADAR: Learning to Route with Asymmetry-aware Distance Representations,” in *International Conference on Learning Representations*, 2026.
 - [26] R. Yang and C. Fan, “Neural Combinatorial Optimization for Time Dependent Traveling Salesman Problem,” in *Advances in Neural Information Processing Systems*, 2025.
 - [27] J. Son, Z. Zhao, F. Berto, C. Hua, Z. Cao, C. Kwon, and J. Park, “Towards Real-World Routing with Neural Combinatorial Optimization,” in *International Conference on Learning Representations*, 2026.
 - [28] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th global ed., Pearson Education, 2022.
 - [29] S. A. Cook, “The Complexity of Theorem-Proving Procedures,” in *ACM Symposium on Theory of Computing*, 1971.
 - [30] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis II, “An Analysis of Several Heuristics for the Traveling Salesman Problem,” *SIAM Journal on Computing*, vol. 6, no. 3, pp. 563–581, 1977.

- [31] C. E. Miller, A. W. Tucker, and R. A. Zemlin, “Integer Programming Formulation of Traveling Salesman Problems,” *Journal of the ACM (JACM)*, vol. 7, no. 4, pp. 326–329, 1960.
- [32] Gurobi Optimization, LLC, <https://www.gurobi.com>, 2026.
- [33] IBM ILOG CPLEX Optimization Studio, <https://www.ibm.com/products/ilog-cplex-optimization-studio>, 2026.
- [34] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by Simulated Annealing,” *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
- [35] F. Glover, “Future Paths for Integer Programming and Links to Artificial Intelligence,” *Computers & Operations Research*, vol. 13, no. 5, pp. 533–549, 1986.
- [36] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
- [37] M. Dorigo, V. Maniezzo, and A. Colorni, “Ant System: Optimization by a Colony of Cooperating Agents,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 1, pp. 29–41, 1996.
- [38] P. Shaw, “Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems,” in *Principles and Practice of Constraint Programming — CP98*, M. Maher and J.-F. Puget, Eds., Springer Berlin, Heidelberg, 1998, pp. 417–431.
- [39] S. Ropke and D. Pisinger, “An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows,” *Transportation Science*, vol. 40, no. 4, pp. 455–472, 2006.
- [40] K. Helsgaun, “An Effective Implementation of the Lin–Kernighan Traveling Salesman Heuristic,” *European Journal of Operational Research*, vol. 126, no. 1, pp. 106–130, 2000.
- [41] T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei, “A Hybrid Genetic Algorithm for Multidepot and Periodic Vehicle Routing Problems,” *Operations Research*, vol. 60, no. 3, pp. 611–624, 2012.
- [42] T. Vidal, “Hybrid Genetic Search for the CVRP: Open-Source Implementation and SWAP* Neighborhood,” *Computers & Operations Research*, vol. 140, p. 105643, 2022.

-
- [43] M. Ehrgott and X. Gandibleux, “A Survey and Annotated Bibliography of Multiobjective Combinatorial Optimization,” *OR-Spektrum*, vol. 22, no. 4, pp. 425–460, 2000.
 - [44] S. Greco, M. Ehrgott, and J. R. Figueira, *Multiple Criteria Decision Analysis: State of the Art Surveys*, 2nd ed., Springer New York, NY, 2016.
 - [45] M. Ehrgott, *Multicriteria Optimization*. Springer Berlin, Heidelberg, 2005.
 - [46] A. Herzel, S. Ruzika, and C. Thielen, “Approximation Methods for Multiobjective Optimization Problems: A Survey,” *INFORMS Journal on Computing*, vol. 33, no. 4, pp. 1284–1299, 2021.
 - [47] Q. Zhang and H. Li, “MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition,” *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007.
 - [48] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.
 - [49] H. Ishibuchi and T. Murata, “A Multi-Objective Genetic Local Search Algorithm and its Application to Flowshop Scheduling,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 28, no. 3, pp. 392–403, 1998.
 - [50] M. Ehrgott, “A Discussion of Scalarization Techniques for Multiple Objective Integer Programming,” *Annals of Operations Research*, vol. 147, pp. 343–360, 2006.
 - [51] E. U. Choo and D. R. Atkins, “Proper Efficiency in Nonconvex Multicriteria Programming,” *Mathematics of Operations Research*, vol. 8, no. 3, pp. 467–470, 1983.
 - [52] C. Audet, J. Bignon, D. Cartier, S. Le Digabel, and L. Salomon, “Performance Indicators in Multiobjective Optimization,” *European Journal of Operational Research*, vol. 292, no. 2, pp. 397–422, 2021.
 - [53] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca, “Performance Assessment of Multiobjective Optimizers: an Analysis and Review,” *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 117–132, 2003.

- [54] L. M. Zintgraf, T. V. Kanter, D. M. Roijers, F. A. Oliehoek, and P. Beau, “Quality Assessment of MORL Algorithms: A Utility-Based Approach,” in *Proceedings of the 24th Annual Machine Learning Conference of Belgium and the Netherlands (Benelearn)*, 2015.
- [55] M. Fleischer, “The Measure of Pareto Optima: Applications to Multi-objective Metaheuristics,” in *International Conference on Evolutionary Multi-Criterion Optimization*, 2003.
- [56] Z. Sun and Y. Yang, “DIFUSCO: Graph-based Diffusion Solvers for Combinatorial Optimization,” in *Advances in Neural Information Processing Systems*, 2023.
- [57] A. Lischka, F. Rydin, J. Wu, M. H. Chehreghani, and B. Kulcsár, “A GREAT Architecture for Edge-Based Graph Problems Like TSP,” *arXiv preprint*, 2025, arXiv:2408.16717.
- [58] D. S. Lai, O. Caliskan Demirag, and J. M. Leung, “A Tabu Search Heuristic for the Heterogeneous Vehicle Routing Problem on a Multigraph,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 86, pp. 32–52, 2016.
- [59] H. Ben Ticha, N. Absi, D. Feillet, and A. Quilliot, “Multigraph Modeling and Adaptive Large Neighborhood Search for the Vehicle Routing Problem with Time Windows,” *Computers & Operations Research*, vol. 104, pp. 113–126, 2019.
- [60] H. Tikani, M. Setak, and E. Demir, “Multi-Objective Periodic Cash Transportation Problem with Path Dissimilarity and Arrival Time Variation,” *Expert Systems with Applications*, vol. 164, p. 114 015, 2021.
- [61] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
- [62] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., MIT Press, 2018.
- [63] K. Hornik, M. Stinchcombe, and H. White, “Multilayer Feedforward Networks are Universal Approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.
- [64] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is All you Need,” in *Advances in Neural Information Processing Systems*, 2017.

-
- [65] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer Normalization,” *arXiv preprint*, 2016, arXiv:1607.06450.
 - [66] 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 *AAAI Conference on Artificial Intelligence*, 2023.
 - [67] T. N. Kipf and M. Welling, “Semi-Supervised Classification with Graph Convolutional Networks,” in *International Conference on Learning Representations*, 2017.
 - [68] C. K. Joshi, T. Laurent, and X. Bresson, “An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem,” *arXiv preprint*, 2019, arXiv:1906.01227.
 - [69] C. K. Joshi, “Transformers are Graph Neural Networks,” *arXiv preprint*, 2025, arXiv:2506.22084.
 - [70] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Póczos, R. R. Salakhutdinov, and A. J. Smola, “Deep Sets,” in *Advances in Neural Information Processing Systems*, 2017.
 - [71] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
 - [72] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” in *International Conference on Learning Representations*, 2015.
 - [73] R. J. Williams, “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning,” *Machine Learning*, vol. 8, no. 3–4, pp. 229–256, 1992.
 - [74] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” *arXiv preprint*, 2017, arXiv:1707.06347.
 - [75] J. Ma, W. Pan, Y. Li, and J. Yan, “ML4CO-Bench-101: Benchmark Machine Learning for Classic Combinatorial Problems on Graphs,” in *Advances in Neural Information Processing Systems*, 2025.
 - [76] M. Kim, J. Park, and J. Park, “Sym-NCO: Leveraging Symmetricity for Neural Combinatorial Optimization,” in *Advances in Neural Information Processing Systems*, 2022.

- [77] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, “Learning Combinatorial Optimization Algorithms over Graphs,” in *Advances in Neural Information Processing Systems*, 2017.
- [78] Y. Xiao, D. Wang, B. Li, H. Chen, W. Pang, X. Wu, H. Li, D. Xu, Y. Liang, and Y. Zhou, “Reinforcement Learning-Based Nonautoregressive Solver for Traveling Salesman Problems,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 7, pp. 13 402–13 416, 2025.
- [79] 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 *AAAI Conference on Artificial Intelligence*, 2024.
- [80] X. Pan, C. Wang, C. Ying, Y. Xue, and T. Yu, “Beyond the Heatmap: A Rigorous Evaluation of Component Impact in MCTS-Based TSP Solvers,” in *International Conference on Learning Representations*, 2026.
- [81] W. Kool, H. van Hoof, J. Gromicho, and M. Welling, “Deep Policy Dynamic Programming for Vehicle Routing Problems,” in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 2022.
- [82] B. Hudson, Q. Li, M. Malencia, and A. Prorok, “Graph Neural Network Guided Local Search for the Traveling Salesperson Problem,” in *International Conference on Learning Representations*, 2022.
- [83] R. Qiu, Z. Sun, and Y. Yang, “DIMES: A Differentiable Meta Solver for Combinatorial Optimization Problems,” in *Advances in Neural Information Processing Systems*, 2022.
- [84] Y. Li, J. Guo, R. Wang, and J. Yan, “From Distribution Learning in Training to Gradient Search in Testing for Combinatorial Optimization,” in *Advances in Neural Information Processing Systems*, 2023.
- [85] D. Zhang, H. Dai, N. Malkin, A. Courville, Y. Bengio, and L. Pan, “Let the Flows Tell: Solving Graph Combinatorial Problems with GFlowNets,” in *Advances in Neural Information Processing Systems*, 2023.

-
- [86] X. Chen and Y. Tian, “Learning to Perform Local Rewriting for Combinatorial Optimization,” in *Advances in Neural Information Processing Systems*, 2019.
 - [87] Y. Wu, W. Song, Z. Cao, J. Zhang, and A. Lim, “Learning Improvement Heuristics for Solving Routing Problems,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 9, pp. 5057–5069, 2022.
 - [88] P. da Costa, J. Rhuggenaath, Y. Zhang, A. Akcay, and U. Kaymak, “Learning 2-Opt Heuristics for Routing Problems via Deep Reinforcement Learning,” *SN Computer Science*, vol. 2, no. 5, 2021.
 - [89] Y. Ma, J. Li, Z. Cao, W. Song, L. Zhang, Z. Chen, and J. Tang, “Learning to Iteratively Solve Routing Problems with Dual-Aspect Collaborative Transformer,” in *Advances in Neural Information Processing Systems*, 2021.
 - [90] A. Hottung and K. Tierney, “Neural Large Neighborhood Search for the Capacitated Vehicle Routing Problem,” in *European Conference on Artificial Intelligence*, 2020.
 - [91] M. Kim, J. Park, and J. Park, “Learning to CROSS Exchange to Solve Min-Max Vehicle Routing Problems,” in *International Conference on Learning Representations*, 2023.
 - [92] Y. Ma, Z. Cao, and Y. M. Chee, “Learning to Search Feasible and Infeasible Regions of Routing Problems with Flexible Neural K-Opt,” in *Advances in Neural Information Processing Systems*, 2023.
 - [93] B. Romera-Paredes, M. Barekatain, A. Novikov, et al., “Mathematical Discoveries from Program Search with Large Language Models,” *Nature*, vol. 625, pp. 468–475, 2024.
 - [94] H. Ye, J. Wang, Z. Cao, F. Berto, C. Hua, H. Kim, J. Park, and G. Song, “ReEvo: Large Language Models as Hyper-Heuristics with Reflective Evolution,” in *Advances in Neural Information Processing Systems*, 2024.
 - [95] S. Feng, W. Sun, S. Li, A. Talwalkar, and Y. Yang, “FrontierCO: Real-World and Large-Scale Evaluation of Machine Learning Solvers for Combinatorial Optimization,” in *International Conference on Learning Representations*, 2026.

- [96] 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 *Advances in Neural Information Processing Systems*, 2021.
- [97] H. Ye, J. Wang, Z. Cao, H. Liang, and Y. Li, “DeepACO: Neural-enhanced Ant Systems for Combinatorial Optimization,” in *Advances in Neural Information Processing Systems*, 2023.
- [98] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, “Exact Combinatorial Optimization with Graph Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems*, 2019.
- [99] F. Berto, C. Hua, J. Park, et al., “RL4CO: an Extensive Reinforcement Learning for Combinatorial Optimization Benchmark,” in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2025.
- [100] Z. Fang, D. Wang, J. Chen, J. Wang, and Z. Zhang, “UCPO: A Universal Constrained Combinatorial Optimization Method via Preference Optimization,” in *AAAI Conference on Artificial Intelligence*, 2026.
- [101] D. Drakulic, S. Michel, F. Mai, A. Sors, and J.-M. Andreoli, “BQ-NCO: Bisimulation Quotienting for Efficient Neural Combinatorial Optimization,” in *Advances in Neural Information Processing Systems*, 2023.
- [102] F. Luo, X. Lin, F. Liu, Q. Zhang, and Z. Wang, “Neural Combinatorial Optimization with Heavy Decoder: Toward Large Scale Generalization,” in *Advances in Neural Information Processing Systems*, 2023.
- [103] J. Zhou, Y. Wu, W. Song, Z. Cao, and J. Zhang, “Towards Omnigeneralizable Neural Methods for Vehicle Routing Problems,” in *International Conference on Machine Learning*, 2023.
- [104] 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 *Advances in Neural Information Processing Systems*, 2022.
- [105] R. Chen, C. Zhou, C. Yu, Y. Chen, Y. Zhou, Z. Chen, and Z. Wang, “SPACE: Unifying Symmetric and Asymmetric Routing Problems for Generalist Neural Solver,” *arXiv preprint*, 2026, arXiv:2605.24484.

-
- [106] X. Lin, Z. Yang, and Q. Zhang, “Pareto Set Learning for Neural Multi-Objective Combinatorial Optimization,” in *International Conference on Learning Representations*, 2022.
 - [107] J. Chen, Z. Zhang, Z. Cao, Y. Wu, Y. Ma, T. Ye, and J. Wang, “Neural Multi-Objective Combinatorial Optimization with Diversity Enhancement,” in *Advances in Neural Information Processing Systems*, 2023.
 - [108] M. Fan, Y. Wu, Z. Cao, W. Song, G. Sartoretti, H. Liu, and G. Wu, “Conditional Neural Heuristic for Multiobjective Vehicle Routing Problems,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 3, pp. 4677–4689, 2025.
 - [109] M. Fan, J. Zhou, Y. Zhang, Y. Wu, J. Chen, and G. Sartoretti, “Preference-Driven Multi-Objective Combinatorial Optimization with Conditional Computation,” in *Advances in Neural Information Processing Systems*, 2025.
 - [110] C. Zhou, X. Lin, Z. Wang, X. Tong, M. Yuan, and Q. Zhang, “Instance-Conditioned Adaptation for Large-scale Generalization of Neural Routing Solver,” *arXiv preprint*, 2024, arXiv:2405.01906.
 - [111] Y. Wang, Y. Li, J. Yan, and Y. Chang, “StruDiCO: Structured Denoising Diffusion with Gradient-free Inference-stage Boosting for Memory and Time Efficient Combinatorial Optimization,” in *Advances in Neural Information Processing Systems*, 2026.
 - [112] Y. Zhou, L. Kong, and H. Wang, “A Multiobjective Edge-Based Learning Algorithm for the Vehicle Routing Problem with Time Windows,” *Information Sciences*, vol. 715, p. 122 223, 2025.
 - [113] J. Chen, Z. Gong, L. Chen, M. Liu, J. Wang, Y. Yu, and W. Zhang, “Looking Ahead to Avoid Being Late: Solving Hard-Constrained Traveling Salesman Problem,” in *International Conference on Distributed Artificial Intelligence*, 2024.
 - [114] A. Navon, A. Shamsian, G. Chechik, and E. Fetaya, “Learning the Pareto Front with Hypernetworks,” in *International Conference on Learning Representations*, 2021.

