arXiv:2605.05389v1 [cs.LG] 6 May 2026

Two-Stage Learned Decomposition for Scalable Routing on Multigraphs

Filip Rydin[1], **Morteza Haghir Chehreghani**[1,2], **Balázs Kulcsár**[1]
[1]Chalmers University of Technology
[2]University of Gothenburg
{filipry, morteza.chehreghani, kulcsar}@chalmers.se

Abstract

Most neural methods for Vehicle Routing Problems (VRPs) are limited to Euclidean settings or simple graphs. In this work, we instead consider multigraphs, where parallel edges represent distinct travel options with varying trade-offs (e.g., distance vs time). Few methods are designed for such formulations and those that do exist face major scalability issues. We mitigate these scalability issues via a Node-Edge Policy Factorization (NEPF) approach, which splits the routing policy into a node permutation stage and an edge selection stage. To enable the decomposition, we introduce a pre-encoding edge aggregation scheme and a non-autoregressive architecture for the edge stage, as well as a hierarchical reinforcement learning method to train the stages jointly. Our experiments across six VRP variants demonstrate that NEPF matches or outperforms the state-of-the-art in terms of solution quality, while being significantly faster in training and inference[1].

1 Introduction

Learning-based methods for Vehicle Routing Problems (VRPs) are emerging as efficient alternatives to classical methods, such as exact methods and meta-heuristics (Zhou et al., 2025b). Their overall aim is to obtain high-quality solutions in real time with minimal engineering effort to adapt to new problem variants. Recently, several works have even begun exploring cross-problem generalization, paving the way for future VRP foundation models (Drakulic et al., 2025; Zhou et al., 2025a).

However, flexibility in input representation remains a key limitation. Most neural methods operate either on Euclidean instances (Kool et al., 2018; Kwon et al., 2020) or on graphs with a single precomputed edge between each node pair (Kwon et al., 2021; Drakulic et al., 2023). Real-world transportation networks often present multiple alternative paths between locations with differing attributes (e.g., distance, travel time and reliability), naturally leading to a multigraph formulation. Jointly optimizing path selection and visit order often reduces costs significantly compared to fixing paths in advance (Ben Ticha et al., 2017). Despite this, neural approaches for VRPs on multigraphs remain in their early stages.

Only one recent work, Rydin et al. (2026), has explored this setting, but their methods face two problems. First, they maintain the full $\mathcal{O}(MN^2)$ multigraph representation (where M is parallel edges per node pair) throughout encoding, greatly increasing memory footprint compared to node-based $\mathcal{O}(N)$ representations. Second, their decoding operates directly on this dense edge-level representation and is tightly coupled to the multigraph structure. Taken together, these issues prevent scaling current approaches to high node counts with many parallel edges. They also hinder wider adoption of the multigraph framework, for instance in foundation models, where compatibility with standard node-based architectures is desirable.

[1]Our code will be made publicly available upon paper acceptance.

Preprint.

We show that a classical decomposition strategy from operations research (Garaix et al., 2010) can be effectively integrated into modern neural routing models, and that it provides a powerful principle for scalable learning on multigraphs. Concretely, we factorize the routing solution into a node permutation and a sequence of edge choices, and model these components with separate policies trained jointly with hierarchical reinforcement learning. We ensure efficiency in the node permutation stage by proposing a pre-encoding aggregation mechanism that summarizes each set of parallel edges into a compact representation, avoiding the explicit construction of the dense multigraph during encoding. To tackle the edge selection stage, we introduce a light but effective non-autoregressive architecture that operates on the sequence of edge sets. Our contributions are:

- Conceptually, we propose a Node-Edge Policy Factorization (NEPF), a new formulation for learning-based routing on multigraphs that decouples high-level node decisions from edge-level choices, enabling scalability without maintaining the full $\mathcal{O}(MN^2)$ graph structure.
- Methodologically, to enable the two-stage decomposition, we introduce a pre-encoding edge aggregation scheme that compresses parallel edges into a latent representation, significantly reducing memory and computational cost. We also design a lightweight non-autoregressive architecture for the edge selection stage and train both stages jointly using a hierarchical reinforcement learning framework.
- Experimentally, we demonstrate that NEPF matches or outperforms state-of-the-art approaches across six single-objective and multi-objective problems, while being orders of magnitude faster in both training and inference. We also show that the proposed framework is compatible with a wide range of node-based backbones, making it a promising building block in more general routing models.

2 Related Work

Learning for routing on richer graph representations While early work (Vinyals et al., 2015; Bello et al., 2017; Kool et al., 2018; Kwon et al., 2020) focused almost exclusively on Euclidean VRP instances, more recent approaches consider richer and more realistic representations. These include asymmetric graphs (Kwon et al., 2021; Drakulic et al., 2023; Yi et al., 2026), graphs with dynamic edge costs (Yang & Fan, 2025) and real-world instances (Lischka et al., 2025; Son et al., 2026).

To the best of our knowledge, the only prior work on multigraph VRPs is Rydin et al. (2026), which introduced two methods: GMS-EB and GMS-DH. Both methods require the full $\mathcal{O}(MN^2)$ multigraph representation throughout encoding, causing major scalability issues. Moreover, GMS-EB features edge-based autoregressive decoding, incurring $\mathcal{O}(MN^4)$ complexity, while GMS-DH requires pre-pruning the entire graph before constructing routes, degrading performance when parallel edges are numerous or edge selection is difficult.

We propose an alternative factorization that selects nodes first, then edges. This avoids maintaining the dense multigraph during most processing, yielding significantly faster training and inference while ensuring strong performance across diverse problem settings. Our node-centric approach also ensures compatibility with many neural backbones, enabling easier integration with, for example, architectures for constraint handling (Bi et al., 2024; Chen et al., 2024; Bi et al., 2026) or cross-problem generalization (Drakulic et al., 2025; Zhou et al., 2025a).

Classical methods for multigraph VRPs. In the operations research literature, multigraphs are widely used to capture the presence of multiple distinct paths between locations in road networks (Lai et al., 2016; Ben Ticha et al., 2017, 2019; Tikani et al., 2021). Garaix et al. (2010) introduced the two-step decomposition we build on, namely to first fix the node permutation and then solve the edge selection problem, which is known as the *Fixed Sequence Arc Selection Problem* (FSASP). While they solve the FSASP with dynamic programming, we replace both stages with end-to-end learning, enabling GPU acceleration, cross-problem adaptation, and handling of cases (e.g., negative resource consumption) that complicate classical methods. See Appendix A for extended discussion.

3 Problem Formulation, Background and Notation

The goal in a multigraph routing problem is to find the feasible route $r \in \mathcal{R}$ in a multigraph G that minimizes a cost $C(r)$. We assume that G is directed, with nodes V and edges E. It has edge

attributes e_l, $l \in E$ (e.g., distance or travel time) and, depending on the VRP at hand, node attributes n_u, $u \in V$ (e.g., node demand). Denote the connecting edge set between nodes $(u, v) \in V^2$ as E_{uv}.

A route r is fully described by a sequence of edges, but we consider an alternative formulation. In our case, a solution is described by a node sequence $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_T)$, together with a sequence $\boldsymbol{\epsilon} = (\epsilon_1, \ldots, \epsilon_{T-1})$ of connecting edges, such that $\epsilon_t \in E_{\pi_t \pi_{t+1}}$. Formally, the fixed sequence arc selection problem is to determine the optimal $\boldsymbol{\epsilon}$ for a fixed $\boldsymbol{\pi}$. This problem is NP-hard in many cases (Garaix et al., 2010).

Multi-objective optimization. Multigraphs are highly relevant in the Multi-Objective (MO) setting, as each objective typically corresponds to a different edge attribute (consider for instance the m-objective TSP with m edge distances). As such, following Rydin et al. (2026), we heavily emphasize MO problems. The goal is to find the *Pareto set* of routes. Redefine $C : \mathcal{R} \to \mathbb{R}^m$ as a multi-objective route cost. Formally, the Pareto set is the set

$$\text{PS} := \{r \in \mathcal{R} \mid \nexists r' \in \mathcal{R} : (C_i(r') \leq C_i(r)\ \forall i) \wedge (\exists i :\ C_i(r') < C_i(r))\}\,. \tag{1}$$

That is, for a solution in the Pareto set no alternative solution exists that is better in every sense. The *Pareto front* is the corresponding set in the objective space. To solve MO routing problems, we utilize Chebyshev scalarization, which transforms the multi-objective problem into several single-objective problems. Define a preference vector $\lambda \in \mathbb{R}^m$ such that $\lambda_i \geq 0\ \forall i$ and $\sum_i \lambda_i = 1$. Given an ideal value z_i^* for objective i, the Chebyshev cost is defined as

$$C_\lambda(r) := \max_i \{\lambda_i | C_i(r) - z_i^* |\}. \tag{2}$$

For a fixed λ, the Chebyshev subproblem is to solve $\min_r C_\lambda(r)$. Given a solution r in the Pareto set, there always exists a corresponding preference for which the solution r solves the Chebyshev subproblem (Choo & Atkins, 1983). That is, this scalarization can recover the entire Pareto front. In contrast, naive linear scalarization, i.e., using the λ-weighted sum of objectives, might not recover the entire Pareto front (Ehrgott, 2005).

4 Node-Edge Policy Factorization (NEPF) for Multigraph VRPs

Following the classical decomposition by Garaix et al. (2010), our framework is designed to first output $\boldsymbol{\pi}$ and then $\boldsymbol{\epsilon}$, yielding the complete route $r = (\boldsymbol{\pi}, \boldsymbol{\epsilon})$. Denote the joint policy $p_\theta(r \mid G)$, which we factorize as $p_\theta(r \mid G) = p^{\text{node}}_{\theta_1}(\boldsymbol{\pi} \mid G)\, p^{\text{edge}}_{\theta_2}(\boldsymbol{\epsilon} \mid \boldsymbol{\pi}, G)$. Note that the factorization does not restrict the expressivity of the joint policy, as any distribution over routes can be represented in this form. Our first stage, referred to below as the node permutation stage, samples $\boldsymbol{\pi} \sim p^{\text{node}}_{\theta_1}$, while the second stage solves the FSASP to obtain $\boldsymbol{\epsilon} \sim p^{\text{edge}}_{\theta_2}$. We couple the stages through joint hierarchical training. The full setup is visualized in Figure 1.

4.1 Node Permutation Stage

Pre-encoding aggregation. In our framework, rather than processing the entire dense multigraph in the encoding stage like Rydin et al. (2026), we first compute a latent d-dimensional distance matrix $D \in \mathbb{R}^{N \times N \times d}$ using a small pre-encoding module. This minimizes the memory footprint of the full $\mathcal{O}(MN^2)$ multigraph representation. After linear embedding of the edge attributes: $g_l = W_g\, e_l$, we obtain a latent distance vector per node-pair with Deep Sets (Zaheer et al., 2017) according to

$$D_{uv} = \rho\Big(\sum_{l \in E_{uv}} \phi(g_l)\Big), \quad (u, v) \in V^2, \tag{3}$$

where ρ and ϕ are MLPs. We note that this aggregation is permutation-invariant with respect to the edge ordering. The latent distance matrix captures the metric relationship for each node-pair, without explicitly retaining each edge as a unique entity.

Encoding. The encoder maps the latent distance matrix D to node encodings $h_u \in \mathbb{R}^d$, $u \in V$. As architecture, we utilize the Graph Edge Attention Network (GREAT) of Lischka et al. (2025) together with a small number of transformer layers (Vaswani et al., 2017) to obtain expressive node embeddings. This is the same base encoder design as GMS-DH. Unlike many alternative architectures,

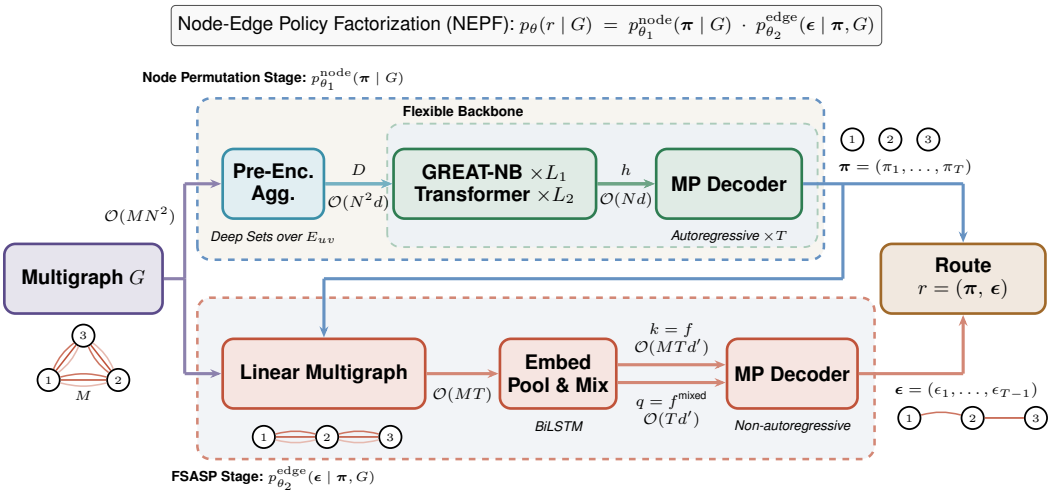

Figure 1: Overview of the proposed node-edge policy factorization for multigraph VRPs. The model first aggregates parallel edges into a compact representation, then generates a node permutation and finally selects edges conditioned on the node sequence. This avoids processing the full $\mathcal{O}(MN^2d)$ embedded multigraph as much as possible, while retaining flexibility. The backbone can be exchanged for any architecture that processes distance matrices and outputs node permutations.

it is capable of producing expressive enough embeddings for the multi-objective asymmetric graph scenario (Rydin et al., 2026).

More specifically, the first L_1 layers of the encoder are Node-Based (NB) GREAT layers. A single such layer is organized as a transformer layer and includes an attention sublayer, a feedforward layer, skip connections and normalization (batchnorm in the original implementation, which we replace with layernorm), all applied to the edge embeddings. The attention sublayer propagates edge embeddings $D_{uv} \mapsto D'_{uv}$ by forming temporary node features x_u, according to

$$x_u = \sum_{v \in \mathcal{N}_u} \left(\alpha'_{uv} W' D_{uv} \| \alpha''_{uv} W'' D_{vu} \right), \tag{4}$$

$$D'_{uv} = W'''(x_u \| x_v), \tag{5}$$

where $\|$ denotes concatenation, $\mathcal{N}_u$ is the neighborhood of node u and α', α'' are learnable attention scores computed via a gating-mechanism. In the last GREAT layer, we only apply (4), refraining from projecting back the node features to edges with (5). These are then passed through L_2 transformer layers, yielding node encodings h_u, $u \in V$. Note that GREAT layers are completely edge-based. Thus, if the original graph has node attributes, we concatenate them with edge attributes before (3).

Decoding. The decoder autoregressively calculates probabilities $p^{\text{node}}_{\theta_1}(\pi_t \mid \pi_{1:t-1}, G)$ for the next node, given context from the partial route $\pi_{1:t-1}$ and encoded nodes h_u, $u \in V$. To parameterize this policy, we utilize a Multi-Pointer (MP) decoder (Jin et al., 2023). We generally refer to the original paper for details. However, we make three modifications to fit this decoder into our setting.

Firstly, we adapt the $-\text{cost}(u, v)$ term in the score calculation (which penalizes expensive node transitions $u \to v$, improving convergence) to the multigraph setting. We set $\text{cost}(u, v)$ slightly differently depending on the VRP (and single-objective vs. multi-objective setting), but in general we use the cost of the cheapest edge between u and v.

Secondly, we weigh this term with a learnable scalar β. This yields superior performance compared to the default choice $\beta = 1$, see Appendix C.3 for an ablation study.

Finally, we solve the problem of partial observability that is introduced by selecting edge traversal after node permutation. To clarify, with our decomposition we deprive the model of full state information for certain problems. For instance, consider time-dependent VRPs, where current time is determined by traversed edges. Often, it is beneficial performance-wise to include such state information in the decoder query. Consequently, to remedy this problem, we propose an auxiliary

RNN-based state estimator that restores accurate state estimates $\hat{s}_t$ given the partial route $\pi_{1:t-1}$. See Appendix C.2 for full details.

Flexible backbone. While we utilize a GREAT-based encoder and autoregressive multi-pointer decoder for the node permutation stage, we emphasize that our framework is backbone-agnostic. In fact, we only require it to be a function $f_{\text{NP}} : \mathbb{R}^{N\times N\times d} \times \mathbb{R}^{N\times d} \to \Pi$, mapping the learned distance matrix D as well as optional embedded node features $z_u \in \mathbb{R}^d$, $u \in V$ to a node permutation $\boldsymbol{\pi} \in \Pi$. Consequently, our framework is highly compatible with existing routing models and in Section 5.2, we show promising results with various backbones in the single-objective setting.

4.2 Fixed Sequence Arc Selection Problem Stage

In the second stage, the input is the sequence $(E_1, E_2, \ldots, E_{T-1})$ of edge sets $E_t = E_{\pi_t \pi_{t+1}}$ and the task is to select one edge per set. To start, for each edge l, we concatenate attributes e_l with (potential) node attributes n_u of the end-point node u, obtaining augmented features e'_l, which are mapped to $\mathbb{R}^{d'}$ with a linear layer: $f_l = W_f\, e'_l$. We then form set-level embeddings with mean pooling:

$$f_t^{\text{pooled}} = \frac{1}{|E_t|} \sum_{l \in E_t} f_l, \quad t = 1, \ldots, T-1. \tag{6}$$

To ensure edge selections are informed by full route context, we mix the sequence of aggregated embeddings with a bi-directional RNN (thereby avoiding attention, which would scale quadratically with T). Mixed embeddings are formed with a BiLSTM and a skip connection according to

$$f_t^{\text{mixed}} = \text{BiLSTM}(f_1^{\text{pooled}}, \ldots, f_{T-1}^{\text{pooled}})_t + f_t^{\text{pooled}}. \tag{7}$$

To calculate scores we set keys to the embeddings $k_l = f_l$ and queries for each position to the mixed embeddings $q_t = f_t^{\text{mixed}}$. We form the score α_l for edge $l \in E_t$ with a multi-pointer mechanism:

$$\alpha_l = \frac{1}{H} \sum_{h=1}^{H} \frac{1}{\sqrt{d'}} (W_h^q q_t)^T (W_h^k k_l) - \widetilde{\beta}\, \text{cost}(l), \quad l \in E_t, \tag{8}$$

where H is the number of heads, $\widetilde{\beta}$ is a learnable parameter and cost is a problem-specific cost function that penalizes expensive edges. To obtain per-step normalized edge probabilities, we apply tanh-clipping and softmax within each edge set E_t, $t = 1, \ldots, T-1$. Edges in subsequent steps are then sampled independently in parallel. This non-autoregressive one-shot sampling allows us to efficiently obtain multiple candidates $\boldsymbol{\epsilon} \sim p_{\theta_2}^{\text{edge}}(\cdot \mid \boldsymbol{\pi}, G)$ for evaluation.

While the FSASP stage is rather shallow, it is robust enough for our purpose. It consistently performs well across problems and the lightweight architecture results in little latency overhead compared to heuristically selecting edges independently per position.

4.3 Hierarchical Training Setup

We propose a hierarchical training algorithm to train the node permutation stage and the FSASP stage jointly. The overall goal is to maximize

$$J(\theta) = \mathbb{E}_{r \sim p_\theta,\, G \sim \mathcal{G}}\big[R(r)\big] = \mathbb{E}_{\boldsymbol{\epsilon} \sim p_{\theta_2}^{\text{edge}},\, \boldsymbol{\pi} \sim p_{\theta_1}^{\text{node}},\, G \sim \mathcal{G}}\big[R(\boldsymbol{\pi}, \boldsymbol{\epsilon})\big], \tag{9}$$

where $R(\boldsymbol{\pi}, \boldsymbol{\epsilon}) = -C(\boldsymbol{\pi}, \boldsymbol{\epsilon})$ is the negative cost of the route $r = (\boldsymbol{\pi}, \boldsymbol{\epsilon})$. We estimate the gradient of this expectation and use it in gradient ascent. With our probability factorization and the REINFORCE estimator (Williams, 1992), we obtain

$$\nabla_\theta J(\theta) = \mathbb{E}_{\boldsymbol{\epsilon} \sim p_{\theta_2}^{\text{edge}},\, \boldsymbol{\pi} \sim p_{\theta_1}^{\text{node}},\, G \sim \mathcal{G}}\Big[R(\boldsymbol{\pi}, \boldsymbol{\epsilon}) \left(\nabla_{\theta_1} \log p_{\theta_1}^{\text{node}}(\boldsymbol{\pi} \mid G) + \nabla_{\theta_2} \log p_{\theta_2}^{\text{edge}}(\boldsymbol{\epsilon} \mid \boldsymbol{\pi}, G)\right)\Big]. \tag{10}$$

To compute a Monte–Carlo approximation, we generate B multigraph instances per batch and K_1 POMO node permutations (Kwon et al., 2020) with differing starting nodes. Additionally, for each node permutation and instance, we sample K_2 edge selections. As already argued, this has negligible runtime effect as long as K_2 remains moderate. Setting $R^*(\boldsymbol{\pi}_{ij}) = \max_{k=1,\ldots,K_2} R(\boldsymbol{\pi}_{ij}, \boldsymbol{\epsilon}_{ijk})$ as the reward for a node-permutation, we define the baselines

$$b^{\text{node}}(G_i) = \frac{1}{K_1} \sum_{j=1}^{K_1} R^*(\boldsymbol{\pi}_{ij}), \quad b^{\text{edge}}(\boldsymbol{\pi}_{ij}, G_i) = \frac{1}{K_2} \sum_{k=1}^{K_2} R(\boldsymbol{\pi}_{ij}, \boldsymbol{\epsilon}_{ijk}). \tag{11}$$

We then estimate the gradient (10) according to

$$\nabla_\theta J(\theta) \approx \frac{1}{BK_1} \sum_{i=1}^{B} \sum_{j=1}^{K_1} \big(R^*(\boldsymbol{\pi}_{ij}) - b^{\text{node}}(G_i)\big) \nabla_{\theta_1} \log p^{\text{node}}_{\theta_1}(\boldsymbol{\pi}_{ij} \mid G_i) + \\ \frac{1}{BK_1K_2} \sum_{i=1}^{B} \sum_{j=1}^{K_1} \sum_{k=1}^{K_2} \big(R(\boldsymbol{\pi}_{ij}, \boldsymbol{\epsilon}_{ijk}) - b^{\text{edge}}(\boldsymbol{\pi}_{ij}, G_i)\big) \nabla_{\theta_2} \log p^{\text{edge}}_{\theta_2}(\boldsymbol{\epsilon}_{ijk} \mid \boldsymbol{\pi}_{ij}, G_i). \tag{12}$$

We refer to this as hierarchical RL because the edge policy p^{edge} optimizes local decisions conditioned on a fixed node sequence, while the node policy p^{node} is trained on the best achievable downstream reward R^* over multiple edge samples. This induces a two-level optimization scheme, where the lower-level edge policy informs the training signal of the higher-level node policy.

4.4 Multi-Objective Preference Conditioning

In the multi-objective setting, following Lin et al. (2022), we use MLP hypernetworks to condition decoding on a preference vector. This allows a single model to recover the full Pareto front without retraining and avoids recomputing node encodings per preference. We use separate MLPs for the two stages, mapping the preference λ to the multi-pointer scoring weights. We also adopt the Chebyshev reward $R_\lambda(r) = -C_\lambda(r)$ for training and sample one preference per batch when estimating (12).

5 Experimental Results

We evaluate NEPF across six VRP variants, demonstrating robust performance and significant efficiency gains over state-of-the-art multigraph methods.

Problems. In the single-objective setting, we benchmark on a Resource Constrained TSP (RCTSP) and multi-resource Orienteering Problem (OP). In the multi-objective setting, we benchmark on the Multi-Objective TSP (MOTSP), CVRP (MOCVRP) and TSP with Time-Windows (MOTSPTW) featured in Rydin et al. (2026), as well as a bi-objective OP (MOOP). All problems have multiple edge features, hence the multigraph framework is appropriate. See Appendix D for further details.

Regarding instance generation, we utilize the graph distributions FLEXx and FIXx from Rydin et al. (2026). The former distribution has variable number of edges between each node pair (max x) and the edge features are uncorrelated. The latter distribution has fixed x edges and negatively correlated features. We benchmark with $x = 2, 5$ for both distributions and 100 nodes. We also complement our analysis with results on more realistic instances in Appendix E.

Baselines. In terms of classical baselines, for the single-objective problems, we utilize Gurobi applied to mixed-integer linear program formulations. In the multi-objective setting, where applicable, we instead utilize state-of-the-art single-objective heuristics together with scalarization and pre-pruning. These include LKH for the MOTSP (Helsgaun, 2017), HGS for the MOCVRP Vidal et al. (2012) and Google OR-tools Perron & Furnon (2019) for both MOTSP and MOCVRP. To supplement the near-optimal baselines, we also apply some simpler problem-specific heuristics for each problem. We refer to Appendix D for details on these. In terms of learning-based methods, we benchmark against optimized re-implementations of GMS-EB and GMS-DH (Rydin et al., 2026). GMS was originally designed for the multi-objective setting, but we adapt them to the single-objective setting by removing the preference-conditioned decoding.

Metrics. For the two single-objective problems, we report the objective value averaged over 10 000 random instances. Similarly, we report the Hypervolume (HV) (Zitzler et al., 2003) for the four multi-objective problems, averaged over 200 instances. In both cases, we also report the gap compared to the best method (in terms of objective value and HV) and the total inference time.

Model settings. We train all neural models for 200 epochs, with 100 000 random problem instances per epoch. Appendix C.1 contains all hyperparameters for NEPF and we refer to the original paper for the hyperparameters for GMS (Rydin et al., 2026). Besides standard inference, we test the

Table 1: Multi-objective problems. Note that HV is "higher-is-better" and normalized to the range $[0, 1]$. The best neural method is in **bold** and the second best underlined.

		FLEX2-100			FLEX5-100			FIX2-100			FIX5-100		
		HV	Gap	Time	HV	Gap	Time	HV	Gap	Time	HV	Gap	Time
MOTSP	LKH	0.94	0.00%	(7.0m)	0.97	0.00%	(5.9m)	0.95	0.00%	(5.9m)	0.91	0.00%	(5.6m)
	OR-tools	0.91	3.46%	(4.8m)	0.96	1.60%	(4.3m)	0.93	2.25%	(4.3m)	0.90	1.78%	(4.4m)
	Nearest Neighbor	0.88	6.71%	(25s)	0.94	3.33%	(25s)	0.91	4.23%	(25s)	0.88	3.11%	(25s)
	GMS-EB	0.93	1.52%	(3.2m)	0.96	0.94%	(7.7m)	0.94	1.12%	(3.2m)	0.90	0.97%	(7.7m)
	GMS-EB (aug)	0.93	1.27%	(25m)	0.97	0.77%	(1.0h)	0.94	0.93%	(25m)	0.90	0.91%	(1.0h)
	GMS-DH	0.92	2.22%	(34s)	0.95	1.98%	(33s)	0.93	2.44%	(31s)	0.89	2.43%	(48s)
	GMS-DH (aug)	0.93	1.84%	(4.0m)	0.96	1.50%	(4.2m)	0.93	2.21%	(4.0m)	0.89	2.33%	(6.2m)
	NEPF	0.93	1.63%	(11s)	0.96	0.95%	(11s)	0.94	1.21%	(11s)	0.90	0.96%	(11s)
	NEPF (aug)	**0.93**	**1.14%**	(1.4m)	**0.97**	**0.67%**	(1.4m)	**0.94**	**0.92%**	(1.4m)	**0.91**	**0.66%**	(1.4m)
MOCVRP	HGS	0.89	0.00%	(22h)	0.95	0.00%	(22h)	0.91	0.00%	(22h)	0.87	0.00%	(22h)
	OR-tools	0.84	5.96%	(32m)	0.92	2.79%	(31m)	0.88	3.80%	(33m)	0.84	3.12%	(31m)
	Nearest Neighbor	0.73	17.49%	(1.0m)	0.86	9.18%	(1.0m)	0.81	10.70%	(1.0m)	0.81	7.29%	(1.0m)
	GMS-EB	0.86	3.26%	(4.3m)	0.93	1.67%	(10m)	0.89	2.19%	(4.4m)	0.84	3.63%	(12m)
	GMS-EB (aug)	0.87	2.63%	(34m)	0.93	1.38%	(1.8h)	0.89	1.87%	(35m)	0.84	3.51%	(2.1h)
	GMS-DH	0.84	5.97%	(36s)	0.89	6.24%	(41s)	0.87	5.10%	(39s)	0.84	3.88%	(56s)
	GMS-DH (aug)	0.84	5.08%	(4.6m)	0.91	4.37%	(5.2m)	0.87	4.57%	(4.9m)	0.84	3.98%	(7.9m)
	NEPF	0.86	2.90%	(16s)	0.93	1.38%	(16s)	0.89	2.10%	(15s)	0.86	1.65%	(16s)
	NEPF (aug)	**0.87**	**2.24%**	(1.8m)	**0.94**	**1.06%**	(1.9m)	**0.90**	**1.72%**	(1.8m)	**0.86**	**1.40%**	(1.8m)
MOTSPTW	Insertion	0.92	1.67%	(20m)	0.96	0.00%	(1.8h)	0.92	2.92%	(25m)	0.86	4.09%	(1.8h)
	GMS-EB	0.63	32.41%	(3.2m)	0.57	40.72%	(6.6m)	0.64	32.40%	(3.2m)	0.88	2.31%	(7.7m)
	GMS-EB (aug)	0.67	28.20%	(26m)	0.60	38.19%	(1.0h)	0.69	27.42%	(26m)	0.90	0.36%	(1.0h)
	GMS-DH	0.69	25.66%	(32s)	0.56	41.76%	(36s)	0.95	0.63%	(34s)	0.86	4.71%	(51s)
	GMS-DH (aug)	0.76	18.52%	(4.6m)	0.63	35.10%	(5.2m)	**0.95**	**0.00%**	(4.9m)	0.89	0.83%	(7.2m)
	NEPF	0.93	0.70%	(17s)	0.94	2.13%	(20s)	0.94	0.87%	(17s)	0.89	0.98%	(20s)
	NEPF (aug)	**0.93**	**0.00%**	(2.2m)	**0.95**	**1.22%**	(2.7m)	0.95	0.30%	(2.2m)	**0.90**	**0.00%**	(2.7m)
MOOP	Greedy	0.38	60.90%	(52s)	0.51	47.98%	(2.2m)	0.45	52.12%	(52s)	0.49	46.17%	(3.0m)
	GMS-EB	0.95	0.98%	(3.7m)	0.98	0.26%	(8.8m)	0.92	1.13%	(3.5m)	0.91	0.03%	(7.5m)
	GMS-EB (aug)	0.96	0.55%	(29m)	**0.98**	**0.00%**	(1.2h)	0.92	1.35%	(28m)	**0.91**	**0.00%**	(1.0h)
	GMS-DH	0.96	0.53%	(44s)	0.98	0.36%	(50s)	0.93	0.10%	(44s)	0.89	1.89%	(56s)
	GMS-DH (aug)	0.96	0.26%	(5.1m)	0.98	0.19%	(5.9m)	**0.93**	**0.00%**	(5.2m)	0.89	1.58%	(6.9m)
	NEPF	0.96	0.32%	(21s)	0.98	0.21%	(25s)	0.91	2.46%	(18s)	0.85	6.72%	(18s)
	NEPF (aug)	**0.96**	**0.00%**	(2.7m)	0.98	0.01%	(3.2m)	0.92	1.16%	(2.4m)	0.86	4.99%	(2.6m)

learning-based methods with $\times 8$ instance augmentation, by scaling the edge attributes with varying factors (Lischka et al., 2025). During testing we also apply quantization and perform inference in half-precision, a straightforward approach to substantially reduce runtime.

Hardware. We train and evaluate learning-based methods with an NVIDIA Tesla A40 GPU with $48\,\mathrm{GB}$ of VRAM. We evaluate classical methods using an Intel Xeon Gold 6338 CPU, with parallelization across 32 processes for a more fair comparison with neural models.

5.1 Performance Evaluation

We present results on multi-objective problems in Table 1 and single-objective problems in Table 2. Overall, NEPF demonstrates competitive solution quality while achieving substantial speedups over existing neural methods. We analyze the results along three dimensions: solution quality, runtime efficiency, and distribution-dependent behavior.

Multi-objective problems. NEPF achieves the best neural performance in 12 out of 16 benchmark configurations (Table 1). Compared to GMS-EB, NEPF matches or improves solution quality on 14/16 cases while reducing inference time by 1-2 orders of magnitude. Against GMS-DH, NEPF provides better hypervolume on 13/16 cases with 2-3$\times$ faster inference. The improvements are particularly pronounced on MOTSPTW, where GMS fails to perform robustly.

Table 2: Single-objective problems. Note that the RCTSP is a minimization problem, while the OP is a maximization problem.

		FLEX2-100			FLEX5-100			FIX2-100			FIX5-100		
		Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
	Gurobi	3.66	0.00%	(4.9h)	1.49	0.00%	(9.2h)	9.41	0.00%	(4.3h)	14.4	0.00%	(7.8h)
	Beam Search	6.85	87.10%	(1.1h)	3.74	151.50%	(2.7h)	14.3	52.23%	(1.2h)	18.8	30.55%	(3.0h)
RCTSP	GMS-EB	5.44	48.52%	(2.5m)	2.24	50.60%	(5.4m)	15.0	59.06%	(2.8m)	27.4	90.14%	(7.2m)
RCTSP	GMS-EB (aug)	5.02	36.92%	(20m)	2.07	39.01%	(42m)	14.0	48.49%	(23m)	26.4	83.07%	(59m)
RCTSP	GMS-DH	4.34	18.41%	(1.2m)	2.43	63.15%	(1.8m)	11.3	19.87%	(1.5m)	16.1	11.65%	(3.9m)
RCTSP	GMS-DH (aug)	4.20	14.69%	(9.3m)	2.19	47.23%	(14m)	11.0	17.34%	(12m)	**15.8**	**9.52%**	(31m)
	NEPF	4.23	15.40%	(39s)	1.84	23.41%	(43s)	11.5	22.45%	(40s)	22.0	52.96%	(51s)
	NEPF (aug)	**4.06**	**10.76%**	(5.2m)	**1.74**	**16.58%**	(5.9m)	**11.0**	**16.95%**	(5.3m)	21.6	49.81%	(6.8m)
	Gurobi	46.9	0.00%	(7.8h)	50.0	0.00%	(5.9h)	38.3	0.00%	(5.1h)	39.3	0.00%	(16h)
	Greedy	36.9	21.22%	(29s)	46.2	7.70%	(1.4m)	31.1	19.00%	(24s)	34.2	12.80%	(1.2m)
OP	GMS-EB	44.0	6.15%	(2.40m)	44.6	10.83%	(5.6m)	36.2	5.45%	(2.3m)	37.2	5.16%	(6.3m)
OP	GMS-EB (aug)	44.0	6.15%	(19m)	47.3	5.40%	(48m)	36.2	5.45%	(19m)	37.2	5.16%	(48m)
OP	GMS-DH	44.6	4.83%	(1.2m)	50.0	0.08%	(1.8m)	36.4	4.96%	(1.5m)	37.8	3.78%	(4.0m)
OP	GMS-DH (aug)	44.6	4.83%	(9.4m)	**50.0**	**0.00%**	(13m)	**36.4**	**4.96%**	(12m)	**37.8**	**3.78%**	(29m)
	NEPF	44.7	4.64%	(40s)	50.0	0.06%	(46s)	35.3	7.80%	(38s)	33.2	15.34%	(46s)
	NEPF (aug)	**44.7**	**4.59%**	(5.5m)	**50.0**	**0.00%**	(6.2m)	35.4	7.75%	(5.2m)	33.3	15.30%	(6.2m)

Table 3: NEPF together with various backbone architectures on the single-objective problems.

	RCTSP				**OP**			
	FLEX2-100		FIX2-100		FLEX2-100		FIX2-100	
	Obj.	Time	Obj.	Time	Obj.	Time	Obj.	Time
NEPF	4.23	(39s)	11.5	(40s)	44.7	(40s)	35.3	(38s)
NEPF + MatNet	5.13	(38s)	12.8	(39s)	43.8	(38s)	34.6	(36s)
NEPF + GOAL	6.94	(25s)	15.8	(26s)	41.3	(24s)	32.7	(22s)
NEPF + ICAM	5.48	(1.3m)	13.2	(1.3m)	43.0	(1.1m)	34.1	(51s)

Single-objective problems. On single-objective problems (Table 2), NEPF outperforms both baselines in 5/8 configurations. For the RCTSP, NEPF achieves the best neural performance on the FLEX distributions while maintaining significantly faster inference than GMS. On the OP, NEPF matches the strong performance of GMS-DH on the FLEX distributions while being 2-3$\times$ faster.

FIX5 distribution. Note that NEPF underperforms on FIX5 across three problems. We attribute this to extreme correlation structure and homogeneity among node pairs. Since all node pairs have exactly 5 edges with strongly negatively correlated attributes, pre-encoding aggregation might lose information critical for distinguishing edges. However, in Appendix E we show that NEPF achieves the best neural performance on more realistic road networks with up to 22 edges per node pair, suggesting the FIX5 underperformance reflects limitations of that particular synthetic stress test rather than scalability issues with parallel edges.

Comparison to classical methods. On multi-objective problems with strong classical baselines (MOTSP, MOCVRP), NEPF achieves 1-3% HV gaps to the best classical method while being 5-20$\times$ faster. For single-objective problems, Gurobi retains better solution quality, but requires 4-16 hours per 10 000 instances compared to our 40-60 seconds, highlighting the inference advantage of NEPF.

5.2 Cross-Backbone Results

We now show results when replacing the GREAT-based node permutation backbone with alternatives based on MatNet (Kwon et al., 2021), GOAL (Drakulic et al., 2025) and ICAM (Zhou et al., 2024). The former two utilize mixed-score attention encoders, that incorporate the latent distance matrix into attention calculations, while the latter utilizes an Adaptation Attention Free Module (AAFM), with superior complexity to full attention. See Appendix C.4 for details on integration with NEPF.

Table 3 shows experimental results in the single-objective case. While changing the backbone yields slightly worse performance, overall solution quality is maintained for both problems and distributions.

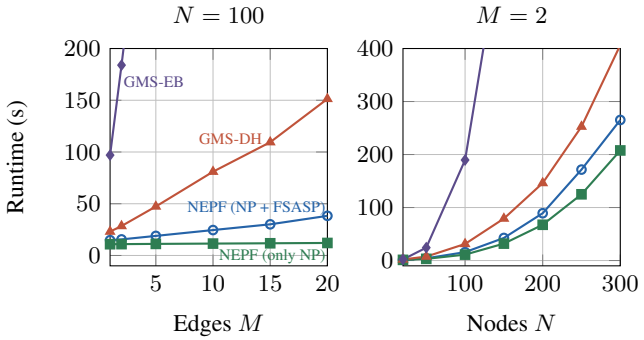

Figure 2: Inference time (total for 200 instances) as a function of parallel edges M and nodes N for the MOTSP.

Table 4: MOTSP zero-shot performance on larger instances.

	FLEX2-200		FLEX2-300	
LKH	0.97	0.00%	0.98	0.00%
GMS-EB	0.95	1.63%	0.96	1.74%
GMS-DH	0.95	2.37%	0.95	2.67%
NEPF	0.95	1.70%	0.96	1.77%
	FIX2-200		FIX2-300	
LKH	0.97	0.00%	0.98	0.00%
GMS-EB	0.96	1.20%	0.96	1.24%
GMS-DH	0.94	2.68%	0.95	2.81%
NEPF	0.96	1.24%	0.96	1.27%

Table 5: MOTSP train time per epoch, $N = 100$.

	$M = 2$	$M = 5$
NEPF (NP + FSASP)	37m	49m
NEPF (only NP)	35m	39m
GMS-DH	1.4h	3.2h
GMS-EB	12h	30h

Table 6: Ablation study (MOTSPTW FLEX2-100)

	HV	Time
NEPF	**0.9282**	(17s)
NEPF w.o. FSASP stage	0.7242	(11s)
NEPF w.o. FSASP mixing	0.9169	(15s)
NEPF w.o. pre-enc. agg.	0.9251	(17s)

We argue that this shows that NEPF can be integrated with foundations models like GOAL and URS (Zhou et al., 2025a) (which is largely based on the AAFM architecture) to extend them to multigraph routing. In the multi-objective setting, performance instead degrades substantially when changing the backbone, which we attribute to the inability of the various encoders to produce expressive enough embeddings for even simple graph asymmetric settings (see Appendix H.2 of Rydin et al. (2026)).

5.3 Scalability Evaluation

In Figure 2, we directly show the advantage in empirical runtime scaling of our method compared to the state-of-the-art. Note that the quadratic scaling with node count is inherent for autoregressive construction-based neural methods, but NEPF grows slower than the alternatives. Taken together with strong zero-shot performance on larger instances in Table 4, these results highlight the strengths of NEPF. Table 5 also shows the significantly reduced training time compared to the baselines. We motivate the scalability further in Appendix B, showing the advantageous theoretical complexity of NEPF compared to the baselines.

5.4 Ablations

Finally, we ablate some of the key components in our framework. We focus on the MOTSPTW, as that problem requires a context-dependent non-trivial edge selection. The results in Table 6 demonstrate the effectiveness of all included components. In particular, removing the FSASP stage entirely and replacing it with a simple heuristic that optimizes each edge independently (based on scalarized edge cost) leads to considerably worse performance.

6 Conclusion

We introduce a node-edge policy factorization for multigraph VRPs, showing that routing decisions can be decomposed into high-level node permutations and low-level edge selections, while retaining joint optimization. This decoupling eliminates the need to process dense multigraph representations throughout the model, addressing a key scalability bottleneck in prior neural approaches and enabling significant improvements in efficiency. From a broader perspective, our results suggest that separating structural and local decisions via factorization can be a powerful principle for extending neural combinatorial optimization to richer, more realistic problem settings

In future work, we will apply our approach to more challenging VRPs with hard constraints and stochastic travel times. We will also investigate combining NEPF with learning-to-partition paradigms, to develop multiscale methods for massive-scale multigraph routing.

Acknowledgments and Disclosure of Funding

The authors would like to thank Attila Lischka, Valter Schütz, Jiaming Wu and Hannes Nilsson for valuable comments throughout the research process. This work was performed as a part of the research project "LEAR: Robust LEArning methods for electric vehicle Route selection" funded by the Swedish Electromobility Centre (SEC). 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.

References

Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural Combinatorial Optimization with Reinforcement Learning. In *International Conference on Learning Representations*, 2017.

Hamza Ben Ticha, Nabil Absi, Dominique Feillet, and Alain Quilliot. Empirical Analysis for the VRPTW with a Multigraph Representation for the Road Network. *Computers & Operations Research*, 88:103–116, 2017.

Hamza Ben Ticha, Nabil Absi, Dominique Feillet, and Alain Quilliot. Multigraph Modeling and Adaptive Large Neighborhood Search for the Vehicle Routing Problem with Time Windows. *Computers & Operations Research*, 104:113–126, 2019.

Jieyi Bi, Yining Ma, Jianan Zhou, Wen Song, Zhiguang Cao, Yaoxin Wu, and Jie Zhang. Learning to Handle Complex Constraints for Vehicle Routing Problems. In *Advances in Neural Information Processing Systems*, 2024.

Jieyi Bi, Zhiguang Cao, Jianan Zhou, Wen Song, Yaoxin Wu, Jie Zhang, Yining Ma, and Cathy Wu. Towards Efficient Constraint Handling in Neural Solvers for Routing Problems. In *International Conference on Learning Representations*, 2026.

Jingxiao Chen, Ziqin Gong, Lvda Chen, Minghuan Liu, Jun Wang, Yong Yu, and Weinan Zhang. Looking Ahead to Avoid Being Late: Solving Hard-Constrained Traveling Salesman Problem. In *International Conference on Distributed Artificial Intelligences*, 2024.

E. U. Choo and D. R. Atkins. Proper Efficiency in Nonconvex Multicriteria Programming. *Mathematics of Operations Research*, 8(3):467–470, 1983.

Darko Drakulic, Sofia Michel, Florian Mai, Arnaud Sors, and Jean-Marc Andreoli. BQ-NCO: Bisimulation Quotienting for Efficient Neural Combinatorial Optimization. In *Advances in Neural Information Processing Systems*, 2023.

Darko Drakulic, Sofia Michel, and Jean-Marc Andreoli. GOAL: A Generalist Combinatorial Optimization Agent Learner. In *International Conference on Learning Representations*, 2025.

Matthias Ehrgott. *Multicriteria Optimization*. Springer Berlin, Heidelberg, 2005.

Thierry Garaix, Christian Artigues, Dominique Feillet, and Didier Josselin. Vehicle Routing Problems with Alternative Paths: An Application to On-Demand Transportation. *European Journal of Operational Research*, 204(1):62–75, 2010.

Keld Helsgaun. An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems. Technical report, Roskilde University, 2017.

Yan Jin, Yuandong Ding, Xuanhao Pan, Kun He, Li Zhao, Tao Qin, Lei Song, and Jiang Bian. Pointerformer: Deep Reinforced Multi-Pointer Transformer for the Traveling Salesman Problem. In *AAAI Conference on Artificial Intelligence*, 2023.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, Learn to Solve Routing Problems! In *International Conference on Learning Representations*, 2018.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. POMO: Policy Optimization with Multiple Optima for Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2020.

Yeong-Dae Kwon, Jinho Choo, Iljoo Yoon, Minah Park, Duwon Park, and Youngjune Gwon. Matrix Encoding Networks for Neural Combinatorial Optimization. In *Advances in Neural Information Processing Systems*, 2021.

David S.W. Lai, Ozgun Caliskan Demirag, and Janny M.Y. Leung. A Tabu Search Heuristic for the Heterogeneous Vehicle Routing Problem on a Multigraph. *Transportation Research Part E: Logistics and Transportation Review*, 86:32–52, 2016.

Adam N. Letchford, Saeideh D. Nasiri, and Amar Oukil. Pricing routines for vehicle routing with time windows on road networks. *Computers & Operations Research*, 51:331–337, 2014.

Xi Lin, Zhiyuan Yang, and Qingfu Zhang. Pareto Set Learning for Neural Multi-Objective Combinatorial Optimization. In *International Conference on Learning Representations*, 2022.

Attila Lischka, Filip Rydin, Jiaming Wu, Morteza Haghir Chehreghani, and Balázs Kulcsár. A GREAT Architecture for Edge-Based Graph Problems Like TSP, 2025. arXiv:2408.16717.

J. F. Pekny and D. L. Miller. An Exact Parallel Algorithm for the Resource Constrained Traveling Salesman Problem with Application to Scheduling with an Aggregate Deadline. In *ACM Annual Conference on Cooperation*, 1990.

Laurent Perron and Vincent Furnon. Or-tools, 2019. `https://developers.google.com/optimization/`.

Filip Rydin, Attila Lischka, Jiaming Wu, Morteza Haghir Chehreghani, and Balázs Kulcsár. Beyond Simple Graphs: Neural Multi-Objective Routing on Multigraphs. In *International Conference on Learning Representations*, 2026.

Jiwoo Son, Zhikai Zhao, Federico Berto, Chuanbo Hua, Zhiguang Cao, Changhyun Kwon, and Jinkyoo Park. Towards Real-World Routing with Neural Combinatorial Optimization. In *International Conference on Learning Representations*, 2026.

Hamid Tikani, Mostafa Setak, and Emrah Demir. Multi-Objective Periodic Cash Transportation Problem with Path Dissimilarity and Arrival Time Variation. *Expert Systems with Applications*, 164:114015, 2021.

Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. The Orienteering Problem: A Survey. *European Journal of Operational Research*, 209(1):1–10, 2011.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, 2017.

Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, Nadia Lahrichi, and Walter Rei. A Hybrid Genetic Algorithm for Multidepot and Periodic Vehicle Routing Problems. *Operations Research*, 60(3):611–624, 2012.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer Networks. In *Advances in Neural Information Processing Systems*, 2015.

Ronald J. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8(3–4):229–256, 1992.

Ruixiao Yang and Chuchu Fan. Neural Combinatorial Optimization for Time Dependent Traveling Salesman Problem. In *Advances in Neural Information Processing Systems*, 2025.

Hang Yi, Ziwei Huang, Yining Ma, and Zhiguang Cao. RADAR: Learning to Route with Asymmetry-aware Distance Representations. In *International Conference on Learning Representations*, 2026.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J. Smola. Deep Sets. In *Advances in Neural Information Processing Systems*, 2017.

Changliang Zhou, Xi Lin, Zhenkun Wang, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. Instance-Conditioned Adaptation for Large-scale Generalization of Neural Routing Solver, 2024. arXiv:2405.01906.

Changliang Zhou, Canhong Yu, Shunyu Yao, Xi Lin, Zhenkun Wang, Yu Zhou, and Qingfu Zhang. URS: A Unified Neural Routing Solver for Cross-Problem Zero-Shot Generalization, 2025a. arXiv:2509.23413.

Fangting Zhou, Attila Lischka, Balázs Kulcsár, Jiaming Wu, Morteza Haghir Chehreghani, and Gilbert Laporte. Learning for Routing: A Guided Review of Recent Developments and Future Directions. *Transportation Research Part E: Logistics and Transportation Review*, 202:104278, 2025b.

Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert da Fonseca. Performance Assessment of Multiobjective Optimizers: an Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.

A Extended Method Motivation

The original FSASP formulated by Garaix et al. (2010) is exactly the multidimensional multiple choice knapsack problem, which is NP-hard. Nevertheless, under some assumptions on resources, the problem can be solved in pseudo-polynomial time with dynamic programming by viewing it as a shortest path problem with resource constraints. The question is then, why do we need learning for the second stage?

Firstly, the original dynamic programming algorithm formulated by Garaix et al. (2010) has been noted to be time-consuming under repeated evaluations across a large number of node permutations (Ben Ticha et al., 2019). We observe that our framework requires a substantial number of FSASPs to be solved for each batch of instances. In fact, we require one solution per instance, POMO node permutation and preference in the multi-objective scenario ($64 \cdot 100 \cdot 101$ total solutions per batch in the bi-objective case with 100 nodes). While subsequent work have proposed improved heuristic and exact methods (Lai et al., 2016; Ben Ticha et al., 2019), our FSASP architecture naturally admits full GPU-acceleration and parallelization. In contrast, dynamic programming approaches are often sequential by nature.

Secondly, the end-to-end learnability of our framework allows it to be adapted across problems without much modification. In contrast, exact methods and hand-crafted heuristics often need considerable problem-specific tuning to perform well.

Thirdly, our method works under assumptions that break many classical methods. One example is negative resource consumption along edges (for instance under electric vehicle battery regeneration), which usually complicates shortest path reformulations by breaking monotonicity assumptions.

Note, however, that when efficient and effective heuristics are available, they can be easily integrated into our framework by replacing the learned FSASP stage. We employ such strategies in our experiments on the MOTSP and MOCVRP, see Appendix C.1.

B Time and Space Complexity

We analyze the time and space complexity of each component of NEPF in turn, using the following notation: N nodes, M parallel edges per node pair, d node permutation embedding dimension, d' FSASP embedding dimension, H pointer heads, $T \approx N$ tour length, $K_1 \approx N$ POMO samples, K_2 FSASP samples.

Pre-encoding aggregation. The aggregation module is the only component that processes the full multigraph. For each of the $\mathcal{O}(N^2)$ node pairs, the MLP ϕ is applied to each of the M parallel edge embeddings before summing and passing through ρ, yielding a time complexity of $\mathcal{O}(MN^2d^2)$. The space complexity is $\mathcal{O}(MN^2d)$ to hold the full edge set, which is immediately compressed to the latent distance matrix D of size $\mathcal{O}(N^2d)$.

Encoder. Each of the L_1 GREAT-NB layers aggregates over the latent distance matrix and projects back to edge embeddings, incurring $\mathcal{O}(N^2d^2)$ time and $\mathcal{O}(N^2d)$ space. The subsequent L_2 transformer layers perform self-attention over N node embeddings at time complexity $\mathcal{O}(N^2d + Nd^2)$ and space complexity $\mathcal{O}(N^2d)$.

Node permutation decoder. The MP decoder proceeds autoregressively over T steps, computing attention scores between a single query and N keys at each step. The time complexity is $\mathcal{O}(NdH)$ per step, yielding $\mathcal{O}(TNdH) = \mathcal{O}(N^2dH)$ total time and $\mathcal{O}(Nd)$ space. As we roll out K_1 parallel POMO samples, the final complexity is $\mathcal{O}(K_1N^2dH) = \mathcal{O}(N^3dH)$ in time and $\mathcal{O}(K_1Nd) = \mathcal{O}(N^2d)$ in space.

FSASP stage. Given K_1 permutations $\boldsymbol{\pi}$, the stage processes $T-1$ edge sets per permutation, each of size M. The linear embedding and mean pooling over each edge set costs $\mathcal{O}(K_1NMd')$ time and space. The BiLSTM mixing pass over the $T-1$ pooled embeddings adds $\mathcal{O}(K_1Nd'^2)$ time and $\mathcal{O}(K_1Nd')$ space. Score computation requires $\mathcal{O}(K_1NMd'H)$ time and $\mathcal{O}(K_1NMd')$ space. Final sampling of K_2 selections requires $\mathcal{O}(K_1K_2NM)$ time and space. The total FSASP complexity is

therefore $\mathcal{O}(K_1 NMd'H + K_1 K_2 NM + K_1 Nd'^2) = \mathcal{O}(N^2 Md'H + K_2 N^2 M + N^2 d'^2)$ in time and $\mathcal{O}(N^2 Md' + K_2 N^2 M)$ in space.

Summary. Table 7 summarizes the complexity of each component alongside the two baselines.

The dominant memory cost of NEPF arises from the pre-encoding aggregation. However, this cost is incurred only once and can be effectively mitigated via batching strategies (see Section B.1). Although the FSASP stage has complexity $\mathcal{O}(MN^2)$ in time and memory, it uses a shallow architecture and small constants in practice, resulting in a limited empirical footprint.

Compared to the baselines, NEPF avoids propagating the full $\mathcal{O}(MN^2)$ multigraph representation through deep encoding and decoding layers. In particular, NEPF eliminates the $\mathcal{O}(MN^4 d)$ decoding cost of GMS-EB, leading to substantially improved scalability in N. Moreover, while GMS-EB and GMS-DH maintain $\mathcal{O}(MN^2 d)$ representations throughout encoding (both), decoding (GMS-EB), and pruning (GMS-DH), NEPF reduces the dependence on M to shallow pre- and post-processing stages. This reduces memory usage during the main network forward pass, which enables larger batch sizes and lower latency for a fixed number of instances.

Table 7: Time and space complexity per component.

Component	Time	Space
NEPF (ours)		
Pre-encoding aggregation	$\mathcal{O}(MN^2 d^2)$	$\mathcal{O}(MN^2 d)$
Encoding	$\mathcal{O}(N^2 d^2)$	$\mathcal{O}(N^2 d)$
MP decoder	$\mathcal{O}(N^3 dH)$	$\mathcal{O}(N^2 d)$
FSASP stage	$\mathcal{O}(MN^2 d'H + K_2 MN^2 + N^2 d'^2)$	$\mathcal{O}(MN^2 d' + K_2 MN^2)$
GMS-EB		
Encoding	$\mathcal{O}(MN^2 d^2)$	$\mathcal{O}(MN^2 d)$
Decoding	$\mathcal{O}(MN^4 dH)$	$\mathcal{O}(MN^3 d)$
GMS-DH		
Encoding	$\mathcal{O}(MN^2 d^2)$	$\mathcal{O}(MN^2 d)$
Pruning	$\mathcal{O}(MN^2 d^2 + MN^2 dH)$	$\mathcal{O}(MN^2 d)$
Decoding	$\mathcal{O}(N^3 dH)$	$\mathcal{O}(N^2 d)$

B.1 Stage-wise dynamic batching.

As the space complexity decreases substantially for NEPF after the pre-encoding aggregation layer, a smart inference strategy under memory constraints is to run this module sequentially with small batch sizes. After initial aggregation, the batches can be concatenated and the rest of the model run with a larger batch size. As this strategy reduces latency significantly compared to utilizing a small batch size throughout the entire model, we employ it in our experiments.

C Additional Method Details

C.1 Model Settings

Here we specify additional hyperparameters for NEPF. Remark that the training parameters below apply to GMS too.

Training setup. We set the batch size to $B = 64$. As optimizer, we use ADAM (Kingma & Ba, 2015) with learning rate $\eta = 10^{-4}$ and weight decay 10^{-6}. In the multi-objective setting, we sample one preference $\lambda = (\lambda_1, \lambda_2)$ per batch with $\lambda_1 \sim \text{Unif}[0, 1]$, $\lambda_2 = 1 - \lambda_1$. We also apply the curriculum learning of Rydin et al. (2026) to further speed up training, where NEPF utilizes the same curriculum as GMS-DH.

Model hyperparameters. We set the embedding dimension to $d = 128$, feed forward hidden dimension to 512, number of encoder layers to $L_1 = 5$ and $L_2 = 2$ and utilize 8 attention heads in

Table 8: Ablation of the auxiliary state estimation module on the OP and FLEX2-100. In the third row, we show results when simply using $\hat{s}_t = t - 1$. The third column is the feasibility rate across the 10 000 test instances.

	Obj.	Feas.	Time
NEPF	**44.6791**	100%	(40s)
NEPF w.o. state est.	43.9264	99.99%	(38s)
NEPF w. simple state est.	43.7936	100%	(38s)

the encoder and MP decoder. In the FSASP stage we set a slightly smaller embedding dimension $d' = 64$, but retain 8 attention heads. We sample $K_2 = 20$ edge selections per node permutation during training and $K_2 = 50$ during evaluation, with $K_1 = N$ as POMO size. Finally, regarding the clipping constants in the decoding, we set $c = 50$ for the node permutation stage and $\widetilde{c} = 1$ for the FSASP stage. The latter choice promotes sample diversity among the K_2 parallel edge selections.

Replacing FSASP stage with simple heuristic. For the MOTSP and MOCVRP, edge selection given a node permutation and a scalarizing preference λ can be approximated cheaply with greedy selection per node pair. This is because local edge selection does not impact downstream costs, compared to for instance the MOTSPTW, where selecting time-consuming edges early may lead to time-window violations later in the route. Consequently, for these two problems we do not employ a learned FSASP stage in our experiments. Instead, we greedily select the edge between each node pair that minimizes linearly scalarized cost. We analyze this approximation more directly in Appendix C.5.

C.2 Auxiliary State Estimator

In the node permutation decoder, we form the query using

$$q_t = W_1 h_{\pi_1} + W_2 h_{\pi_{t-1}} + W_3 \bar{h}_{\text{graph}} + W_4 \bar{h}_{\text{visited}} + W_5 s_t, \tag{13}$$

where s_t represents current state information at step t, such as the time or the vehicle load. However, as we select edges after nodes, the exact state s_t is not always available. As such, we propose an auxiliary module that explicitly estimates the state. The effect of this module on the OP can be seen in Table 8. Similarly, we utilize this module in our experiments for the RCTSP and MOOP. We do not utilize it for the MOTSP, MOCVRP or MOTSPTW.

Given the partial node permutation, $(\pi_1, \dots, \pi_{t-1})$, we form the state estimate recursively using the node encodings according to

$$\hat{s}_t = W^{\text{state}} h_t^{\text{state}}, \tag{14}$$

$$h_t^{\text{state}} = \text{LSTM}(h_{\pi_1}, \dots, h_{\pi_{t-1}})_{t-1}. \tag{15}$$

That is, the state estimator shares node representations with the decoder. Note that we predict the raw state. An alternative would be to predict an increment $\Delta\hat{s}_t$ and set $\hat{s}_t = \Delta\hat{s}_t + \hat{s}_{t-1}$. However, we find this to yield worse results empirically.

We train this module using the ground-truth state s_t, available after edge selection. Since we sample multiple edge selections $\boldsymbol{\epsilon}_k,\ k = 1, \dots, K_2$ per instance and node permutation, we select the state of the best such selection as the ground truth. The loss is then formed via

$$\mathcal{L}_{\text{est}} = \sum_{t=1}^{T} (\hat{s}_t - s_t)^2, \tag{16}$$

and added to the node permutation loss and edge selection loss in (12).

C.3 MP Decoder Cost Scaling Parameter

In the multi-pointer score calculation, we introduce a learnable parameter β to weigh the problem-specific cost functions. Empirically, we observe that a high β is preferable to the base choice $\beta = 1$. This is shown for the MOTSP in Table 9. Thus, instead of heuristically selecting a specific value, we propose to learn this scalar, which seems to work comparably.

Table 9: Hypervolume for different β values for the MOTSP and FLEX2-100. In the learnable case, the final value is $\beta \approx 6.44$.

	HV
NEPF β learnable	0.9292
NEPF $\beta = 1$	0.9254
NEPF $\beta = 2$	0.9283
NEPF $\beta = 5$	0.9294

C.4 Integration with MatNet, GOAL and ICAM

We integrate NEPF with alternative backbones by replacing the GREAT + transformer encoder with MatNet, GOAL and ICAM encoders. On the other hand, we keep the multi-pointer decoder for all setups as well as the POMO rollout and training. This is not strictly necessary, but simplifies the experimental setup. It is also possible to use the decoding and training setups from the original papers (e.g., multi-head attention decoder for MatNet and BQ-NCO (Drakulic et al., 2023) training for GOAL), but we leave this for future work. Further integration details are provided below for each architecture.

MatNet. We let the the pre-encoding aggregation layer output a latent distance matrix of dimension $N \times N \times 64$ (compared to vanilla $N \times N \times 128$). This distance matrix is inserted into the attention calculations using the idea from Appendix A.1 in the original paper. That is, an MLP (hidden dimension size 32) mixes internally computed attention scores with the external information provided by the distance matrix. Note that MatNet supports node features (unlike GREAT). Hence, for problems where node features are present, we insert their embeddings as initial "row" embeddings.

GOAL. GOAL mixed-score attention layers are in principle similar to MatNet ones, but the attention mixing is performed with a different mechanism. We let the pre-encoding aggregation output a matrix with the same embedding dimension as the encoder ($d = 128$) and use this matrix directly as encoder input together with potential embedded node features. For problems without node features, we use vectors of all zeros as initial node embeddings.

ICAM. This architecture uses a scalar pairwise adaptation bias inserted into the core mechanism to take into account pairwise distances between nodes. The bias is computed as $f(N, d_{ij}) = -\alpha \cdot \log_2 N \cdot d_{ij}$, where d_{ij} is pairwise distance and α is a learnable parameter. In our case, we first apply our initial aggregation layer to form a latent distance matrix of size $N \times N \times 32$. Then, for each ICAM layer, we introduce a learnable linear projection that compresses the vector of size 32 for each node pair into a scalar, which is used instead of d_{ij}. The linear projection weights are different for each ICAM layer to enable more informative processing through the encoder stack. Embedded node features, if they are present, are used as "row" features.

C.5 Replacing FSASP Stage with Simple Heuristic

As mentioned in Appendix C.1, for the MOTSP and MOCVRP, we replace the learned FSASP stage with a simple function, which selects the edge with lowest linearly scalarized cost between each node pair. Note that this leads to empirically sound performance in Table 1. In this section, however, we want to directly quantify the effect of this heuristic solution method.

Specifically, consider a fixed node permutation π for the MOTSP under a preference λ. We consider the edge selection $\boldsymbol{\epsilon}_{\text{lin}}$, which greedily minimizes linearly scalarized cost $C_\lambda(r) = \lambda_1 C_1(r) + \lambda_2 C_2(r)$, and want to quantify the difference in Chebyshev cost compared to the optimal FSASP solution $\boldsymbol{\epsilon}_{\text{Chb}}$. We perform a direct comparison empirically.

Let π be the node permutation for the optimal route with respect to the linearly scalarized cost. We obtain this with Gurobi and use it since it is impractical to obtain the optimal permutation with respect to the Chebyshev cost (this is a nonlinear cost). We then select $\boldsymbol{\epsilon}_{\text{lin}}$ with the heuristic and the optimal $\boldsymbol{\epsilon}_{\text{Chb}}$ with dynamic programming. Then, we compare the Chebyshev cost of both selections across λ, as well as the difference in hypervolume between both strategies. We perform this experiments for FLEX2-50 and FIX2-50 across 200 random instances.

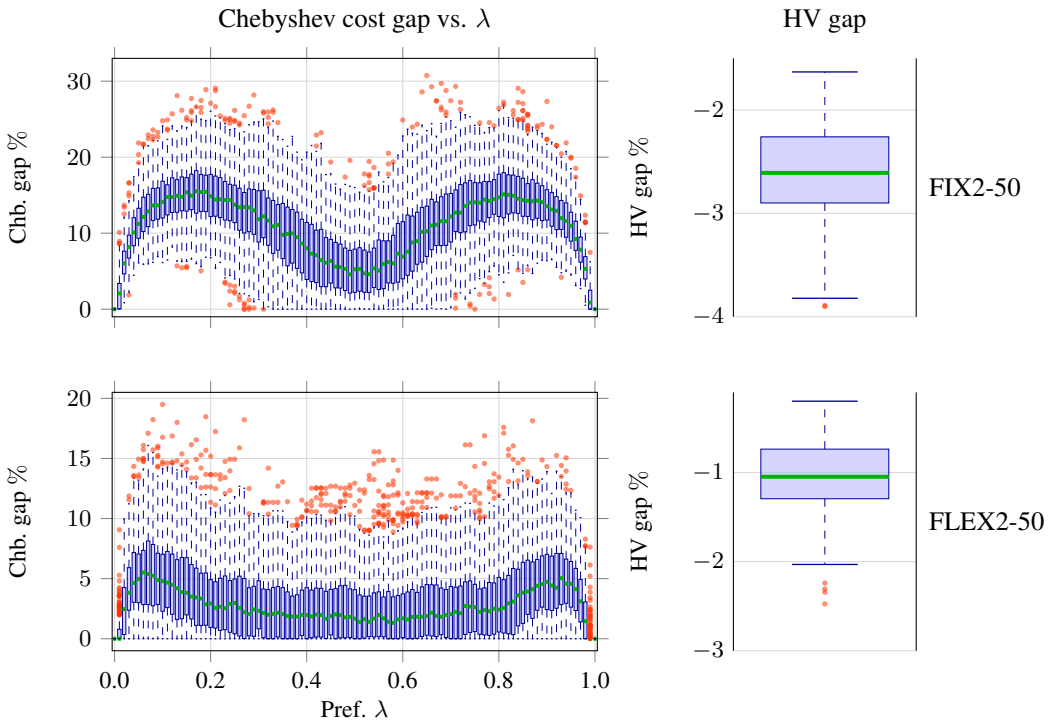

Figure 3: Boxplots over optimality gap between the greedy FSASP solver minimizing linearly scalarized cost and the optimal solution minimizing Chebyshev cost. The left column shows gap in Chebyshev cost as a function of multi-objective preference λ, while the right column shows gap in overall hypervolume.

The boxplots over optimality gaps are visualized in Figure 3. For both distributions, the greedy heuristic never exceeds $\approx 30\%$ gap compared to the optimal solution in terms of Chebyshev cost. Moreover, the average gap is significantly smaller and for many instances and preferences, there is no difference at all between the two solutions. In terms of hypervolume, the linear heuristic always performs slightly better, which might be explained by the underlying node permutation being generated with respect to linear cost.

D Problems and Baselines

D.1 RCTSP

In the resource constrained TSP (Pekny & Miller, 1990), each edge is associated with a cost and a resource consumption. The total cost and resource consumption are given by the sum over the traversed edges. The goal is to minimize the cost of a tour visiting all nodes, while ensuring the total resource consumption stays under a threshold R.

Data generation. For a fixed distribution, edge count and node count, the resource constraint R is constant across instances. Let R_{cost} and R_{resource} be the expected resource consumption for a cost-optimal and resource-optimal tour respectively. We set

$$R = (R_{\text{cost}} + R_{\text{resource}})/4. \tag{17}$$

This constraint tightness level ensures the problem does not reduce into a standard TSP, while it is generally not difficult to find feasible solutions.

Model details. Edge features are the cost and resource consumption and there are no node features. In the decoding for all models, we augment the context with the current resource usage r_t in (13) by

setting $s_t = r_t$. To ensure feasibility, we subtract the constraint violation multiplied with a constant factor from the reward during training. This leads to no recorded infeasible instances for any model during evaluation in our experiments.

Baseline heuristic. Besides Gurobi, we design a beam search heuristic as baseline. It incrementally constructs tours using a Lagrangian score (cost + λ·resource). At each step, partial tours are pruned via feasibility checks and Pareto dominance, retaining a fixed number of beam states. Completed tours are repaired if needed by selecting alternative parallel edges with lower resource consumption. The trade-off parameter λ is adaptively updated across a fixed number of outer iterations to balance cost and resource feasibility.

D.2 OP

Our second single-objective problem is a multi-constraint variant of the orienteering problem. Each edge is associated with two costs and the sum of both costs over traversed edges must not exceed fixed thresholds T_1 and T_2. The goal is to maximize the sum of collected prizes over the visited nodes. As in standard orienteering formulations, not all nodes need to be visited, and the route must terminate at the depot (see Vansteenwegen et al. (2011)).

Data generation. Prizes are sampled uniformly in $[0, 1]$ independently for each node. Similarly to the RCTSP, we set thresholds using the average incurred costs of optimal paths. Let $C_i^{(j)}$ be expected cost i for a route that is optimal with respect to cost j. Since our graph distributions are symmetric with respect to the costs, we note that $C_1^{(2)} = C_2^{(1)}$ and $C_1^{(1)} = C_2^{(2)}$. The thresholds are thus set to

$$T_1 = T_2 = (C_2^{(1)} + C_2^{(2)})/8 = (C_1^{(1)} + C_1^{(2)})/8, \tag{18}$$

where the denominator ensures that many nodes can be visited, but not all.

Model details. Importantly, since neither graph distribution satisfies the triangle inequality, it is non-trivial to mask out infeasible nodes in the decoding. As such, following Lischka et al. (2025), we let the models learn when to return to the depot. We do this by penalizing constraint violation during training as with the RCTSP.

Besides the two-dimensional edge costs, we concatenate the node prize to each incident edge to form edge features in the first stage. In the FSASP stage, we only utilize the original edge costs as features, as prizes are not needed. In the node permutation MP decoder, we augment the context with the current cost level $c_t \in \mathbb{R}^2$ by setting $s_t = c_t$.

Baseline heuristic. We design a greedy constructive heuristic that iteratively builds a path starting from the depot. At each step, it selects the next node by maximizing prize-to-cost ratio, where edge costs are weighted by remaining resource until threshold is reached. A candidate move is only considered if it admits a direct feasible return path to the depot. The process terminates when no feasible extension exists.

D.3 MOTSP

As the simplest multi-objective problem, we consider the multi-objective TSP. Here, edge costs are two-dimensional, and the goal is to find the Pareto set of routes minimizing the two-dimensional sum over all edges.

Data generation. This problem has no additional features beyond the edge costs, which we sample as usual from FLEXx and FIXx.

Model details. For this problem, we do not utilize the FSASP stage. Instead, we replace it by always selecting the parallel edge with the lowest linearly scalarized cost. This reduces runtime while maintaining performance.

Baseline heuristics. We consider a nearest-neighbor heuristic as well as LKH and Google OR-tools. These utilize linear scalarization to transform the multi-objective problem into several single-objective problems. For each scalarized problem, the multigraph is pruned by removing sub-optimal edges with respect to the scalarized cost.

Reference for HV calculation. We use $(60, 60)$ and $(100, 100)$ as reference for the FLEXx-100 and FIXx-100 distributions respectively, regardless of x. In the zero-shot experiments in Table 4, we use $(120, 120)$ and $(180, 180)$ for FLEX2-200 and FLEX2-300 while we use (N, N) for FIX2-N.

D.4 MOCVRP

Similarly to the MOTSP, the multi-objective capacitated vehicle routing problem features two edge costs, which are summarized over the route to yield the objectives. In this case, however, the vehicle must respect a capacity c, while satisfying customer demands $d_i,\ i \in V$. It starts at a depot, and must return occasionally to drop off its load. The route finishes when all customers have been visited.

Data generation. Similarly to Kool et al. (2018), we set the vehicle capacity to $c = 50$ and sample demands uniformly and independently from the set $\{1, \dots, 9\}$.

Model details. Edge features are the two edge costs as well as demand of the end-point node. As in the MOTSP, we do not utilize the FSASP stage for this problem, instead always selecting the edge that minimizes linearly scalarized cost. Finally, we augment the context with the current vehicle load l_t by setting $s_t = l_t$ in the MP decoder.

Baseline heuristics. Similarly to the MOTSP, we utilize a nearest neighbor heuristic, HGS and Google OR-tools as baselines.

Reference for HV calculation. We utilize the same reference points as for the MOTSP.

D.5 MOTSPTW

Following Rydin et al. (2026), we also benchmark on the multi-objective TSP with time-windows. In this problem, one objective is the number of violated time windows, while the other is the traveled distance. Each edge is associated with a distance as well as a travel time. The vehicle starts at a depot and must visit all customers. It must immediately leave a customer upon arrival. That is, it cannot wait until a time window starts.

As this problem features edge attributes interacting with node attributes and early edge selections impact downstream penalties, solving the FSASP is crucial for satisfactory performance.

Data generation. We sample time-windows according to the "medium" distribution of Chen et al. (2024) and Bi et al. (2024).

Model details. Edge features in both stages are the distance, travel time and time-window of the end-point node.

Baseline heuristics. We use an insertion heuristic for the linearly scalarized problem. Nodes are iteratively inserted at the position that minimizes the preference weighted sum of incremental distance and time-window violation. For each insertion, all parallel edge options are considered, and downstream arrival times are updated accordingly.

Reference for HV calculation. For FLEXx, we utilize $(105, 60)$ as reference, where the first objective is the time-window violation. For FIXx, we utilize $(105, 100)$.

D.6 MOOP

Our last problem is a bi-objective version of the orienteering problem. In this setting, each edge is associated with a cost and a resource. One objective is the sum of costs of traversed edges (to be

minimized), whereas the other objective is the sum of collected prizes across visited nodes (to be maximized). The resource consumption must not exceed a threshold R, which ensures the vehicle might not be able to visit all nodes. Note that there is an inherent trade-off between choosing low-cost edges (that minimize the cost objective) and low-resource edges (which ensure the vehicle can visit more nodes).

As we prefer a pure minimization problem for compatibility with our other problems, we change the implementation slightly. Instead of maximizing collected prizes our implementation minimizes prizes not collected.

Data generation. We sample prizes as in the OP (see Section D.2) and set the resource limit as in the RCTSP (see (17)).

Model details. Edge features in the node permutation stage are cost, resource consumption and end-point node prize. In the FSASP stage, edge features are only cost and resource consumption. Additionally, we augment the context with current resource consumption r_t. As in the OP, the model needs to learn when to return to the depot. Thus we set the collected prize to zero when the resource constraint is violated.

Baseline heuristic. Similarly to the OP, we use a greedy heuristic which maximizes prize-to-cost ratio of the next edge traversal. We run this heuristic for each multi-objective preference vector $\lambda = (\lambda_1, \lambda_2)$. For a single run, the prize in the numerator is weighted with the preference of the prize objective λ_1 and the cost in the denominator is weighted by λ_2.

Reference for HV calculation. For FLEXx, we utilize $(50, 25)$ as reference (first objective is prize not collected). For FIXx, we utilize $(50, 30)$.

E Additional Experiments

In this section, we show results when applying NEPF to more realistic instances zero-shot. The purpose is both to show that our method performs well out-of-distribution without any fine-tuning and to show that performance is maintained on distributions where the triangle inequality is satisfied.

We generate instances by first sampling points uniformly in the unit square $[0, 1]^2$ and calculating Euclidean distances. Then, following Letchford et al. (2014) and Ben Ticha et al. (2017), we transform these Euclidean instances into multigraphs with the following procedure. First, for each node pair (u, v) with Euclidean distance $d^{(1)}_{uv}$, we compute a second distance $d^{(2)}_{uv} = \nu \cdot d^{(1)}_{uv} + \mu \cdot \gamma_{uv} \cdot \hat{d}^{(1)}$, where:

- $\hat{d}^{(1)} = \max_{(u,v)\in V^2} d^{(1)}_{uv}$ is the maximum Euclidean distance.
- $\gamma_{uv} \sim \text{Unif}[0, 1]$ is a random variable.
- μ and ν are parameters controlling the correlation between the distances $d^{(1)}$ and $d^{(2)}$.

For the latter, we define three cases: Strong Correlation (SC): $(\nu, \mu) = (0.9, 0.1)$, Weak Correlation (WC): $(\nu, \mu) = (0.5, 0.5)$ and No Correlation (NC): $(\nu, \mu) = (0.1, 0.9)$. Finally, in the resulting simple graph with two edge distances, we solve a multi-objective shortest path problem for each node pair using dynamic programming. The output is a multigraph, where both edge distances satisfy the triangle inequality and where the number of edges between node pairs varies. Table 10 summarizes the statistics of instances generated with this method for 100 nodes. It shows the maximum number of edges per node pair, mean edges per node pair and mean edges per graph. In particular, remark that there exists node pairs with relatively large edge counts.

Table 11 shows results on these instances for one single-objective problem (RCTSP) and one multi-objective problem (MOTSP). We use the same baselines as in the main results in Section 5. All neural methods are applied zero-shot without training on similar instances (we apply the versions trained on FLEX5-100). Also note that we set a time limit for Gurobi to 6 minutes per instance.

To summarize these results, NEPF consistently shows strong zero-shot generalization. Among the neural methods, it performs best in 5/6 cases and matches GMS-DH closely in the remaining case.

Table 10: Statistics for the distributions in this section.

	NC 100	WC 100	SC 100
Max M per node pair	22	17	12
Mean M per node pair	4.70	2.80	1.80
Mean #edges per graph	47k	28k	18k

Table 11: Results across 100 instances induced by multi-objective shortest path calculation. The objective value and gap are calculated over feasible instances only.

		NC 100				WC 100				SC 100			
		HV/Obj.	Gap	Feas.	Time	HV/Obj.	Gap	Feas.	Time	HV/Obj.	Gap	Feas.	Time
MOTSP	LKH	0.74	0.00%	-	(5.3m)	0.67	0.00%	-	(5.7m)	0.70	0.00%	-	(8.2m)
	OR-tools	0.71	3.95%	-	(2.8m)	0.64	4.16%	-	(2.6m)	0.69	2.03%	-	(2.0m)
	NN	0.67	9.31%	-	(13s)	0.60	10.65%	-	(13s)	0.65	8.31%	-	(13s)
	GMS-EB	0.45	39.63%	-	(17m)	0.62	7.24%	-	(13m)	0.62	12.38%	-	(9.1m)
	GMS-EB (aug)	0.71	4.29%	-	(2.2h)	0.64	4.14%	-	(1.7h)	0.67	4.50%	-	(1.2h)
	GMS-DH	0.64	13.15%	-	(34s)	0.59	12.28%	-	(29s)	0.65	7.91%	-	(26s)
	GMS-DH (aug)	0.70	5.63%	-	(4.8m)	0.63	5.10%	-	(4.0m)	0.66	6.09%	-	(3.6m)
	NEPF	0.70	5.17%	-	(8.2s)	0.63	5.89%	-	(7.9s)	0.66	5.77%	-	(7.6s)
	NEPF (aug)	**0.71**	**3.69%**	-	(50s)	**0.64**	**4.04%**	-	(48s)	**0.67**	**4.19%**	-	(47s)
RCTSP	Gurobi	11.9	0.00%	46%	(20m)	9.97	0.00%	22%	(22m)	7.77	0.00%	79%	(16m)
	Beam Search	17.5	46.89%	100%	(13m)	15.0	50.37%	97%	(10m)	10.6	35.93%	100%	(5.5m)
	GMS-EB	29.5	147.20%	94%	(12s)	18.4	84.01%	48%	(17s)	11.4	46.41%	99%	(6.4s)
	GMS-EB (aug)	24.6	105.96%	100%	(1.6m)	18.2	82.90%	69%	(2.2m)	9.92	27.71%	100%	(48s)
	GMS-DH	18.5	54.78%	100%	(3.5s)	14.6	45.97%	37%	(2.5s)	9.94	27.90%	100%	(1.8s)
	GMS-DH (aug)	**17.1**	**43.36%**	100%	(30s)	15.0	50.31%	89%	(18s)	9.60	23.57%	100%	(13s)
	NEPF	18.3	53.83%	100%	(1.0s)	14.0	40.11%	98%	(0.9s)	9.54	22.73%	100%	(0.7s)
	NEPF (aug)	17.4	46.03%	100%	(5.4s)	**13.4**	**34.79%**	100%	(4.4s)	**9.24**	**18.86%**	100%	(3.9s)

Remark especially how NEPF maintains feasibility for the RCTSP, which the other neural methods struggle with. Gurobi also achieves a relatively low feasibility rate, which could likely be mitigated with a warm-start approach or a more stable mixed-integer linear program formulation.