mathematics

MDPI

Article

GRAWA: A Deep Reinforcement Learning Approach for the Multi-Objective Traveling Salesman Problem

Zhengtao Ji [1,2,3], Xiaoxiong Zhang [2,3,*], Xixi Zhu [2,3,4] and Shengye Yan [5]

[1] School of Automation, Nanjing University of Information Science and Technology, Nanjing 210044, China; 202412491646@nuist.edu.cn
[2] The Sixty-Third Research Institute, National University of Defense Technology, Nanjing 210007, China; zhuxixi14@nudt.edu.cn
[3] Laboratory for Big Data and Decision, National University of Defense Technology, Changsha 410073, China
[4] National Key Laboratory of Information Systems Engineering, National University of Defense Technology, Changsha 410073, China
[5] School of Artificial Intelligence, Nanjing University of Information Science and Technology, Nanjing 210044, China; shengye.yan@gmail.com
* Correspondence: xiaoxiongzhang@nudt.edu.cn; Tel.: +86-188-5108-3243

Abstract

The multi-objective traveling salesman problem (MOTSP) is a typical challenging multi-objective combinatorial optimization problem. Numerous advanced deep reinforcement learning algorithms have been developed to solve it. Existing learning-based methods can share information across subproblems through mechanisms such as parameter transfer. This paper investigates explicit graph-based modeling of preference relationships within a shared policy and proposes a graph-relation-driven deep reinforcement learning algorithm for MOTSP. The proposed algorithm trains a model to map problem instances to approximate Pareto sets under a collection of preference vectors. Specifically, the graph weight encoder combines global and objective-specific city representations with preference features learned on a fixed graph whose edges are defined by distances between weight vectors. A graph co-guided fusion module then combines global and objective-specific city embeddings with the learned preference representations to generate preference-conditioned city embeddings. The resulting problem embeddings and contextual embeddings are supplied to the decoder to generate candidate tours in parallel, providing an approximation to the Pareto set. Furthermore, a Top-p training baseline is introduced to adaptively select reference candidates according to their scalarized costs. Finally, extensive experiments are conducted on various MOTSP instances. Comparisons are carried out between the proposed algorithm and heuristic and learning-based algorithms. Experimental results demonstrate the solution quality of the proposed algorithm on the evaluated multi-objective traveling salesman problems. Its computational cost is reported under the specified hardware, decoding budgets, and evaluation settings.

Keywords: multi-objective traveling salesman problem; combinatorial optimization; attention model; deep reinforcement learning; weight distribution graph; adaptive Top-p baseline

MSC: 90C29; 90C27

Check for updates

Academic Editor: Carlos Conceicao Antonio

Received: 17 September 2026
Revised: 23 October 2026
Accepted: 27 October 2026
Published: 7 November 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

As an important extension of the classical NP-hard traveling salesman problem (TSP), the multi-objective traveling salesman problem (MOTSP) is a typical multi-objective combinatorial optimization problem widely found in practical decision-making scenarios such as

logistics distribution [1,2], green transportation [3], and emergency response [4,5]. Traditional TSP solely aims to find a single shortest path that visits all cities and returns to the starting point; in contrast, MOTSP requires the simultaneous optimization of multiple conflicting objective functions, such as total travel distance and overall cost, within a discrete solution space [6]. The core challenge of this class of problems stems from the superposition of dual complexities: "combinatorial (discrete)" and "multi-objective." On the one hand, the discrete nature of decision variables results in an extremely complex solution space structure; on the other hand, the conflict between objectives means that improving one objective often comes at the expense of another. Therefore, the fundamental goal of solving MOTSP is no longer to seek a single absolute optimal solution, but to make multidimensional trade-offs within a complex discrete space, ultimately generating a set of Pareto-optimal solutions that reflect the compromises among various objectives. Computing an exact Pareto set for MOTSPs is computationally challenging. Practical methods therefore seek approximate Pareto sets rather than guaranteeing Pareto optimality over the entire feasible space. These approaches include heuristic and learning-based algorithms.

Heuristic algorithms usually leverage Pareto dominance relations and search strategies to approximate high-quality solutions to the problem. Commonly used heuristic approaches include multi-objective evolutionary algorithms [7–10], Pareto local search algorithms [11–14], and hybrid algorithms that combine multi-objective evolution with Pareto local search [15–17]. The computational cost of these methods depends on the instance size, search operators, and optimization budget. Applying a search procedure to a different problem setting may also require adapting its representation or operators. These considerations motivate investigation of learned construction policies alongside established search-based methods.

Learning-based algorithms adopt decomposition strategies to transform MOTSPs into a set of single-objective subproblems, which are then solved by well-established network models. For instance, the DRL-MOA model [18] uses a set of uniformly distributed weight vectors to scalarize MOTSPs into a series of single-objective combinatorial optimization subproblems. DRL-MOA trains a pointer network [19] for each subproblem and uses a neighborhood-based parameter-transfer strategy to share learned information across subproblems [18]. Each trained model can directly generate approximate optimal solutions for its corresponding subproblem, so as to collectively approximate the Pareto front of MOTSPs. The MODRL/D-AM [20] replaces the pointer network with the attention model (AM) [21], while keeping the remaining components consistent with DRL-MOA. The MODRL/D-EL model [22] introduces an evolutionary search strategy into MODRL/D-AM to boost model training performance. Zhang et al. [23] proposed using a pre-trained attention model as the meta-model for subproblems and fine-tuned this meta-model to obtain a dedicated model for each subproblem. The P-MOCO model [24] trains a hypernetwork to generate decoder parameters for various single-objective subproblems. Wang et al. [25] presented the MORAM model, in which a routing encoder is designed to aggregate global and objective embeddings and improve contextual embeddings for solving multi-objective combinatorial optimization problems. Gao et al. [26] put forward the single-model approach MOPN. This method concatenates problem instances and weight vectors as the input to the pointer network. These methods provide different mechanisms for transferring information or conditioning a policy on preferences. GRAWA investigates a particular architectural design: a fixed graph over preference vectors supports feature propagation, and the resulting preference representations are fused with city features within a shared policy. This design is motivated by relationships between scalarized costs, without assuming that neighboring preferences necessarily yield nearby optimal tours or objective vectors.

To address the limitations of the above-mentioned methods, this paper proposes a Graph-Relation Adaptive-Weight Attention model, termed GRAWA, for solving MOTSPs. First, the graph weight encoder combines global and objective-specific city-encoding branches with a graph weight routing module. The routing module learns representations of the neighborhood relationships among fixed scalarization preferences and supplies these preference features to the subsequent fusion module. Second, a graph co-guided fusion module is constructed. It generates improved problem embeddings via graph neural networks, cross-attention and gating mechanisms to realize node-level adaptive fusion and improve the quality of the Pareto front. Finally, an adaptive Top-p baseline is proposed for model training. It determines the number of reference candidates used to construct the baseline according to their scalarized-cost distribution. The main contributions of this paper are summarized as follows:

(1) Design of a graph weight routing module for MOTSP: a fixed graph represents neighborhood relationships among scalarization preference vectors. Graph propagation, score-gated pooling, and index-based restoration produce learned preference representations, which are subsequently combined with city embeddings to condition a shared tour-construction policy. The contribution lies in this preference-graph encoding and its integration into the MOTSP architecture, rather than in a new graph-convolution or pooling operator.
(2) Construction of a graph co-guided fusion module: cross-attention and two independently learned gates combine preference representations with city-level features. The comparison with weighted aggregation in Section 4.5 evaluates the contribution of this fusion design to mean solution quality under the reported settings.
(3) Design of a Top-p training baseline with an adaptive candidate-set size: a fixed cumulative probability threshold determines a variable number of reference candidates from their scalarized-cost distribution. The threshold is selected empirically for each problem family and remains fixed during training. Under the descriptive common-HV-target criterion in Section 4.6, Top-p reaches the target earlier on average than Top-k in three of the four evaluated scenarios, whereas Top-k reaches it earlier in the tri-objective mixed-distance scenario. These results characterize observed learning performance and do not establish reduced gradient-estimation variance or improved stability across training seeds. The potential bias of the Top-p update and its performance relative to a leave-one-out variant are examined in Appendix A.

2. Related Work and Research Motivations

This section first introduces the basic concepts of the multi-objective traveling salesman problem. Then, it presents decomposition strategies, attention models and their training methods as the research foundation of this paper. On this basis, the limitations of existing learning-based algorithms are analyzed, and the research motivations of this work are clarified.

2.1. Multi-Objective Traveling Salesman Problem

The multi-objective traveling salesman problem (MOTSP) involves n cities and M objectives to be minimized. Throughout this paper, M denotes the number of objectives, whereas $m \in \{1,\ldots,M\}$ indexes an individual objective. In the Euclidean setting, the instance is represented by M coordinate matrices $\mathbf{p}^1,\ldots,\mathbf{p}^M \in \mathbb{R}^{n\times 2}$. The travel cost between cities i and j for objective m is their Euclidean distance in the corresponding coordinate system:

$$c_{ij}^m = \|\mathbf{p}_i^m - \mathbf{p}_j^m\|_2. \tag{1}$$

The fundamental goal of solving this problem is to find a Hamiltonian cycle $\mathbf{x} = (x_1,\ldots,x_n)$ from the entire discrete search space $\mathcal{X}$. This cycle visits every node exactly once and connects all nodes. The tour is evaluated by M objective functions, $f_m(\mathbf{x})$ for $m \in \{1,\ldots,M\}$, each of which is to be minimized. The tour cost for objective m is defined as follows:

$$f_m(\mathbf{x}) = \sum_{i=1}^{n-1} c^m_{x_i,x_{i+1}} + c^m_{x_n,x_1}, \qquad m = 1,\ldots,M. \tag{2}$$

where $c^m_{x_i,x_{i+1}}$ denotes the cost of traveling from node x_i to its next adjacent node x_{i+1} in the path under the m-th objective; $c^m_{x_n,x_1}$ represents the loop-closing cost of returning from the last node x_n of the cycle to the starting node x_1 under the m-th objective.

Pareto dominance, Pareto set, and Pareto front. Let $\mathcal{X}$ be the feasible tour set and $\mathbf{f}(x) = (f_1(x),\ldots,f_M(x))^T$ the objective vector, with all M objectives minimized. For $\mathbf{u},\mathbf{v} \in \mathbb{R}^M$, $\mathbf{u}$ Pareto-dominates $\mathbf{v}$, denoted by $\mathbf{u} \prec \mathbf{v}$, if and only if both conditions below hold:

$$\mathbf{u} \prec \mathbf{v} \Leftrightarrow \begin{cases} u_m \le v_m, & \forall m \in \{1,\ldots,M\}, \\ u_\ell < v_\ell, & \exists \ell \in \{1,\ldots,M\}. \end{cases} \tag{3}$$

A feasible tour y dominates another feasible tour x when $f(y) \prec f(x)$. A tour is Pareto-optimal if no feasible tour dominates it. The Pareto set (PS) consists of all feasible tours whose objective vectors are not dominated by those of any other feasible tour:

$$\mathrm{PS} = \{x \in \mathcal{X} : \nexists y \in \mathcal{X}, \mathbf{f}(y) \prec \mathbf{f}(x)\}. \tag{4}$$

The Pareto front (PF) is the image of the Pareto set in objective space:

$$\mathrm{PF} = \{\mathbf{f}(x) : x \in \mathrm{PS}\} \subseteq \mathbb{R}^M. \tag{5}$$

Thus, the PS consists of feasible tours in the decision space, whereas the PF consists of their nondominated objective vectors. Distinct tours may have the same objective vector, so the mapping from the PS to the PF need not be one-to-one. In the experiments, nondominated solution sets returned by the algorithms are treated as approximations; nondominance within a generated set does not establish membership in the true PS or PF.

2.2. Decomposition Strategy

In practice, the aim is to obtain a diverse set of feasible tours whose objective vectors approximate the Pareto front defined in Section 2.1. Decomposition-based methods address this task by formulating multiple scalarized subproblems with different objective preferences. Two widely used decomposition approaches are the weighted-sum and weighted Chebyshev methods [8]. Their general formulations for M objectives are given below, while Figure 1 illustrates the special case $M = 2$. Let $\mathcal{X}$ denote the feasible tour set and $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}),\ldots,f_M(\mathbf{x}))^T$ the objective vector, with all objectives minimized. For a weight vector $\boldsymbol{\lambda} = (\lambda_1,\ldots,\lambda_M)^T$, the two scalarizing functions are defined as follows:

$$g_{\mathrm{ws}}(\mathbf{x} \mid \boldsymbol{\lambda}) = \sum_{m=1}^{M} \lambda_m f_m(\mathbf{x}). \tag{6}$$

$$g_{\mathrm{tch}}(\mathbf{x} \mid \boldsymbol{\lambda}, \mathbf{z}^*) = \max_{1\le m\le M}\{\lambda_m |f_m(\mathbf{x}) - z^*_m|\}. \tag{7}$$

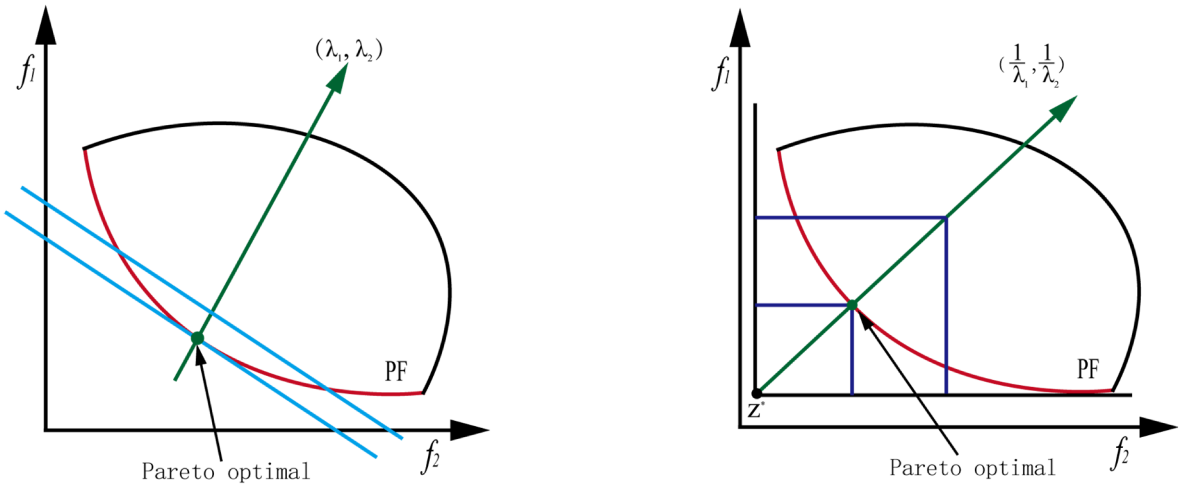
The weights satisfy $\lambda_m \ge 0$ for every $m \in \{1,\ldots,M\}$ and $\sum_{m=1}^{M} \lambda_m = 1$. Here, M denotes the number of objectives and m is the objective index. For this minimization problem, the ideal point $\mathbf{z}^{\mathrm{ideal}}$ consists of the best attainable value of each objective considered separately:

$$z^{\mathrm{ideal}}_m = \min_{\mathbf{x}\in\mathcal{X}} f_m(\mathbf{x}), \qquad m = 1,\ldots,M. \tag{8}$$

These componentwise minima need not be attained by the same feasible tour. Therefore, the ideal point need not itself be a feasible objective vector. A utopian point $\mathbf{z}^*$ is strictly better than the ideal point in every objective and can be written as

$$z_m^* = z_m^{\text{ideal}} - \varepsilon_m, \qquad \varepsilon_m > 0, \quad m = 1, \ldots, M. \tag{9}$$

Thus, $z_m^* < z_m^{\text{ideal}}$ for every objective m, and the utopian point cannot be attained by a feasible tour. Equation (7) uses $\mathbf{z}^*$ as a utopian reference point. The positive offsets ε_m express this strict componentwise improvement; they are introduced here to define the point, not as additional hyperparameters of GRAWA's weighted-sum training objective. For each fixed weight vector, the corresponding single-objective subproblem minimizes either $g_{\text{ws}}(\mathbf{x}|\boldsymbol{\lambda})$ or $g_{\text{tch}}(\mathbf{x}|\boldsymbol{\lambda},\mathbf{z}^*)$ over $\mathbf{x} \in \mathcal{X}$.



(**a**) Weighted-sum method (**b**) Chebyshev method

Figure 1. Schematic illustration of decomposition in the bi-objective case ($M = 2$). PF denotes the Pareto front. (**a**) Weighted-sum scalarization. (**b**) Weighted Chebyshev scalarization with a utopian point $\mathbf{z}^*$, which lies strictly below the ideal point in both objective components. The general M-objective formulations are given in Equations (6) and (7). The red curves represent the Pareto fronts. The cyan lines in (**a**) and blue contours in (**b**) illustrate level sets of the respective scalarizing functions. The green arrows indicate the directions labeled by the weight components in each panel.

Varying the weight vector yields a family of scalarized subproblems whose solutions can be used to approximate the Pareto front. With strictly positive weights, exact weighted-sum minimization yields supported Pareto-optimal solutions. However, varying the weights does not generally recover unsupported points on a non-convex Pareto front, which can occur in discrete MOTSPs. When some weights are zero, exact minimization guarantees only weak Pareto optimality in general.

The preference vectors w^j used in the subsequent GRAWA formulation play the same objective-weighting role as λ here. GRAWA uses the weighted-sum tour cost for policy training; the Chebyshev formulation is introduced as background on decomposition methods. GRAWA's finite preference set and learned approximate solutions provide no guarantee of recovering the entire Pareto front.

2.3. Attention Model and Its Training Method

As shown in Figure 2, the attention model (AM) consists of an encoder–decoder architecture. The encoder is responsible for extracting latent features from input nodes, while the decoder progressively selects city nodes based on the encoder's output and finally generates a complete feasible-solution sequence.

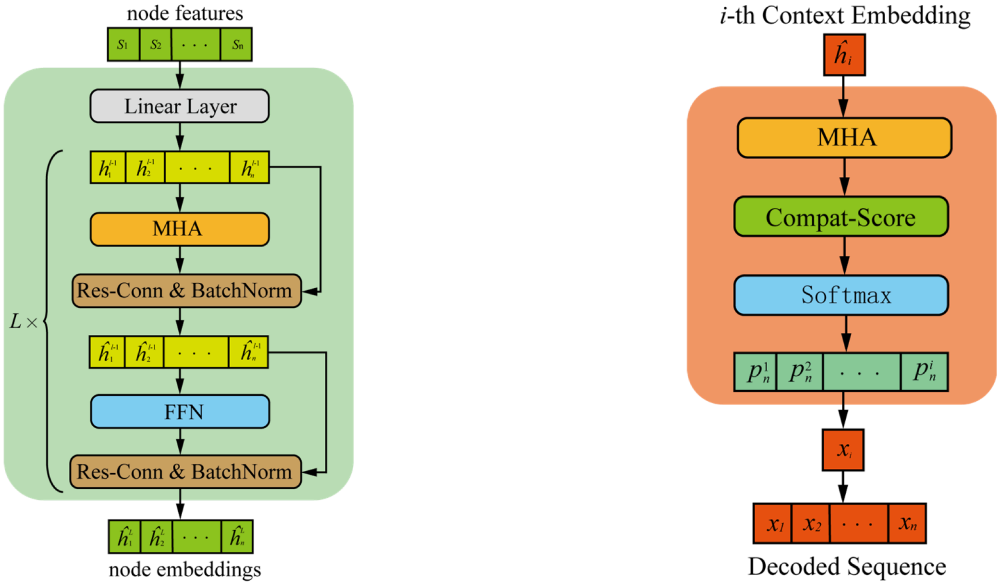


Figure 2. Illustration of the attention model framework. (**a**) Encoder, which maps the raw node features into high-dimensional embeddings through a linear layer followed by L stacked MHA–FFN blocks with residual connections and batch normalization; (**b**) Decoder, which computes the compatibility score of each unvisited node from the current context embedding and constructs the solution sequence autoregressively according to the softmax-normalized selection probabilities.

(1). Encoder: The encoder maps raw node features into high-dimensional node embeddings and models global dependencies among nodes via the attention mechanism. Let $\mathbf{H}^{\ell-1}$ denote the input of the $\ell-$ th encoder layer. The update process of this layer can be formulated as:

$$\mathbf{Z}^{\ell} = \mathrm{BN}(\mathbf{H}^{\ell-1} + \mathrm{MHA}_{\ell}(\mathbf{H}^{\ell-1})). \tag{10}$$

$$\mathbf{H}^{\ell} = \mathrm{BN}(\mathbf{Z}^{\ell} + \mathrm{FFN}_{\ell}(\mathbf{Z}^{\ell})). \tag{11}$$

where $MHA(\cdot)$ denotes the multi-head self-attention module, and $FFN(\cdot)$ represents the feed-forward neural network. After L layers of encoding, the final node embeddings are obtained.

(2). Decoder: The decoder constructs a tour autoregressively by selecting one city node at each step based on the encoder outputs. It computes a selection score for each candidate node using the current context embedding. Let $\mathcal{U}_t$ denote the set of unvisited node indices at step t. The score and selection probability for node i are given in Equations (12) and (13), respectively:

$$u_{t,i} = \begin{cases} \alpha \tanh\left(\frac{(\mathbf{W}_q \mathbf{c}_t)^{\top}(\mathbf{W}_k \mathbf{h}_i)}{\sqrt{d}}\right), & i \in \mathcal{U}_t, \\ -\infty, & i \notin \mathcal{U}_t, \end{cases} \tag{12}$$

$$p_{\theta}(x_t = i \mid S, x_{<t}) = \frac{\exp(u_{t,i})}{\sum_{j=1}^{n} \exp(u_{t,j})}. \tag{13}$$

where $\mathbf{W}_q$ and $\mathbf{W}_k$ are learnable parameters used to generate query vectors and key vectors, respectively; α is the score scaling factor; d denotes the dimension of latent vectors. For visited nodes, the selection scores are set to $-\infty$, yielding zero probability after softmax normalization. This mask prevents a previously selected city from being selected again during tour construction. After all cities have been selected, the closing edge returns to the first city to complete the tour.

(3). Training Method: Since optimal-solution labels for combinatorial optimization problems are generally difficult to obtain on a large-scale basis, deep reinforcement learning is commonly adopted to train the model. The model is treated as a parameterized stochastic policy $p_\theta(x|S)$, which samples and generates feasible solutions x given an input instance S. Let $g(\mathbf{x}|S)$ denote the tour cost to be minimized for solution $\mathbf{x}$ on instance S. The training objective is its expectation under the policy. Following Reference [25], a cost-based REINFORCE estimator [27] is used to update the model by gradient descent, as follows:

$$\nabla_\theta \mathcal{L}(\theta) \approx \frac{1}{B}\sum_{j=1}^{B}\left[g(\mathbf{x}_j|S_j) - b(S_j)\right]\nabla_\theta \log p_\theta(\mathbf{x}_j|S_j). \tag{14}$$

where B denotes the batch size, S_j represents the j-th training instance, x_j is the solution sampled under the current policy, and $b(S_j)$ is a baseline function. An appropriately chosen baseline that is independent of the sampled action conditional on the instance can reduce the variance of the score-function estimator without changing its expectation. This general motivation does not establish variance reduction for the sample-dependent Top-p baseline proposed here; its statistical properties are discussed separately in Section 3.3 and Appendix A.1.

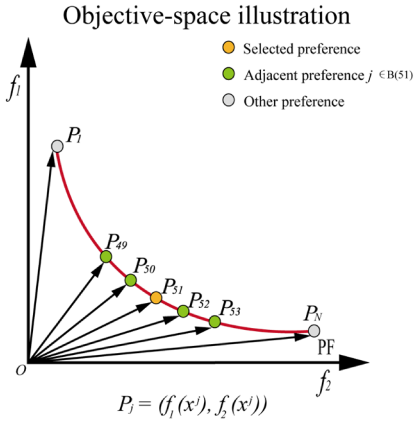
2.4. Research Motivation

Decomposition-based learning methods address MOTSP through scalarized subproblems associated with different preference vectors. DRL-MOA [18] trains a separate model for each subproblem and uses neighborhood-based parameter transfer to share learned information. MORAM [25] and MOPN [26] instead use a shared model conditioned on preference vectors. These approaches demonstrate that information sharing across subproblems is already part of the existing literature. Our motivation is to investigate an explicit preference-graph representation within a shared policy. GRAWA propagates features over a fixed graph of preference vectors and combines the learned preference representations with instance-specific city embeddings. The motivation for this graph is the sensitivity of scalarized costs to changes in preference weights, rather than a guarantee that neighboring preferences produce structurally similar tours or nearby Pareto-front points. This distinguishes the proposed representation and fusion design from the broader idea of sharing information across subproblems. Let w^i denote the preference vector of subproblem i. In Figure 3a, P_i denotes the objective vector of a representative feasible tour x^i associated with that subproblem. For the illustrated bi-objective case, its coordinates are defined as

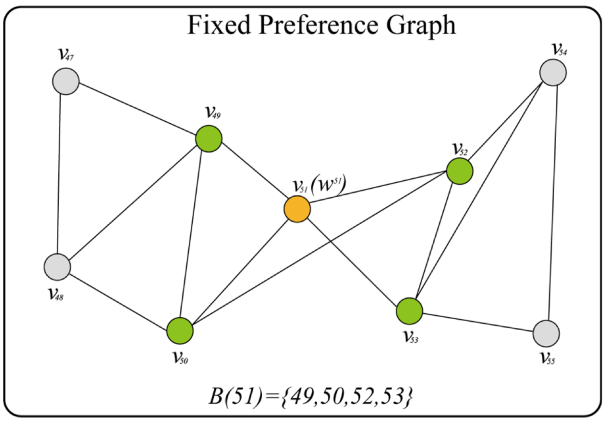
$$P_i = (f_1(x^i), f_2(x^i)), \qquad i = 1,\ldots,N. \tag{15}$$

where N is the number of preference vectors and $f_1(x^i)$ and $f_2(x^i)$ are the two objective costs of tour x^i. The points $P_1,\ldots,P_N$ are indexed by their associated preference vectors; their coordinates are objective values, not city coordinates or preference-vector components. The smooth Pareto-front sketch is illustrative and does not imply that the discrete MOTSP front is continuous, that different preferences produce distinct solutions, or that nearby preferences necessarily yield neighboring optimal objective vectors. Instead, preference proximity motivates information exchange through a cost-based relationship. Appendix D proves that, for a fixed MOTSP instance, a tour that is optimal or near-optimal under one preference has a bounded scalarized-cost gap under a nearby preference, with the bound depending on their Euclidean distance and the objective-space diameter. A strict optimality gap additionally ensures local constancy of the minimizing objective vector. These results motivate preference-based feature sharing without guaranteeing that optimal tours or Pareto-front points are close. The graph in Figure 3b therefore has one node v_i for each preference vector w^i and is constructed directly from these vectors, without using the points

P_i or a known Pareto front. The neighborhood $B(i)$ denotes the indices of nodes adjacent to v_i in the final undirected graph, as formally defined in Section 3.1.2.



(**a**) Preference-associated objective vectors



(**b**) Fixed preference graph

Figure 3. From preference-based subproblems to a fixed preference graph. (**a**) Schematic objective vectors $P_i = (f_1(x^i), f_2(x^i))$ of representative tours associated with the preferences w^i. The axes show objective costs; the sketch does not assert a continuous Pareto front or a one-to-one correspondence between preferences and distinct optimal solutions. (**b**) Each node v_i represents a preference vector w^i. Edges are constructed from Euclidean distances in preference space by taking the union of four-nearest-neighbor relations. The resulting graph is undirected and unweighted, with symmetric binary adjacency. $B(i)$ denotes the set of node indices adjacent to i after symmetrization. Node positions in panel (**b**) are schematic layout positions and have no numerical role in the graph construction. PF denotes the Pareto front. Formal definitions are given in Section 3.1.2, and the graph-construction procedure is provided in Algorithm A1 in Appendix C.

To model relationships between preferences and support their interaction with city representations, we introduce a graph weight encoder containing a graph weight routing module, together with a graph co-guided fusion module. First, the fixed preference graph in Figure 3b is constructed using Euclidean distances between preference vectors. For each node, four nearest distinct preference nodes are selected, with distance ties resolved by the fixed node index. An undirected edge is then added whenever either node selects the other as a nearest neighbor. This union rule yields an unweighted graph with a symmetric binary adjacency matrix. The graph weight routing module learns representations on this fixed graph to support information exchange between related subproblems. The complete mathematical definition, including the distinction between the four-neighbor selection set and the final graph neighborhood $B(i)$, is provided in Section 3.1.2, and an explicit construction procedure is given in Algorithm A1 in Appendix C. Second, the graph co-guided fusion module combines cross-attention with learned preference representations. Two gated branches integrate the context features and projected query features at the city level. This design is intended to support preference-conditioned tour construction; its observed performance relative to weighted aggregation is evaluated in Section 4.5.

Furthermore, an adaptive Top-p baseline is introduced for the reinforcement learning training of GRAWA. Unlike a Top-k baseline that retains a fixed number of reference candidates, Top-p uses a fixed cumulative probability threshold and lets the retained candidate count depend on the current scalarized-cost distribution. The adaptivity is in the count, not in the threshold value. This construction reuses the tours generated in parallel for the same instance. Its empirical learning behavior is evaluated against Top-k in Section 4.6, while its statistical properties and its performance relative to a leave-one-out variant are examined in Appendix A.

3. Graph-Relation Adaptive-Weight Attention Model

As illustrated in Figure 4, the proposed GRAWA model builds and improves upon the standard encoder–decoder architecture of the attention model. The architectural enhancements are primarily concentrated within the encoder, whereas the decoder remains identical to the original attention model. Specifically, the graph weight routing module encodes neighborhood relationships among fixed scalarization preference vectors. It produces learned preference features through graph propagation, score-gated pooling, and index-based restoration. These features are trained through the downstream policy computation and are combined with instance-specific city embeddings in the fusion module; the routing branch does not receive solution structures or Pareto-front points as inputs. Concurrently, to ensure the effective integration of graph-modal and feature-modal information, a graph co-guided fusion module is developed, yielding more discriminative problem embeddings. Finally, the decoder constructs candidate tours for the specified preference vectors. These tours are used to approximate the Pareto set, and their objective vectors provide a Pareto front approximation. No guarantee of Pareto optimality over the full feasible tour set is claimed.

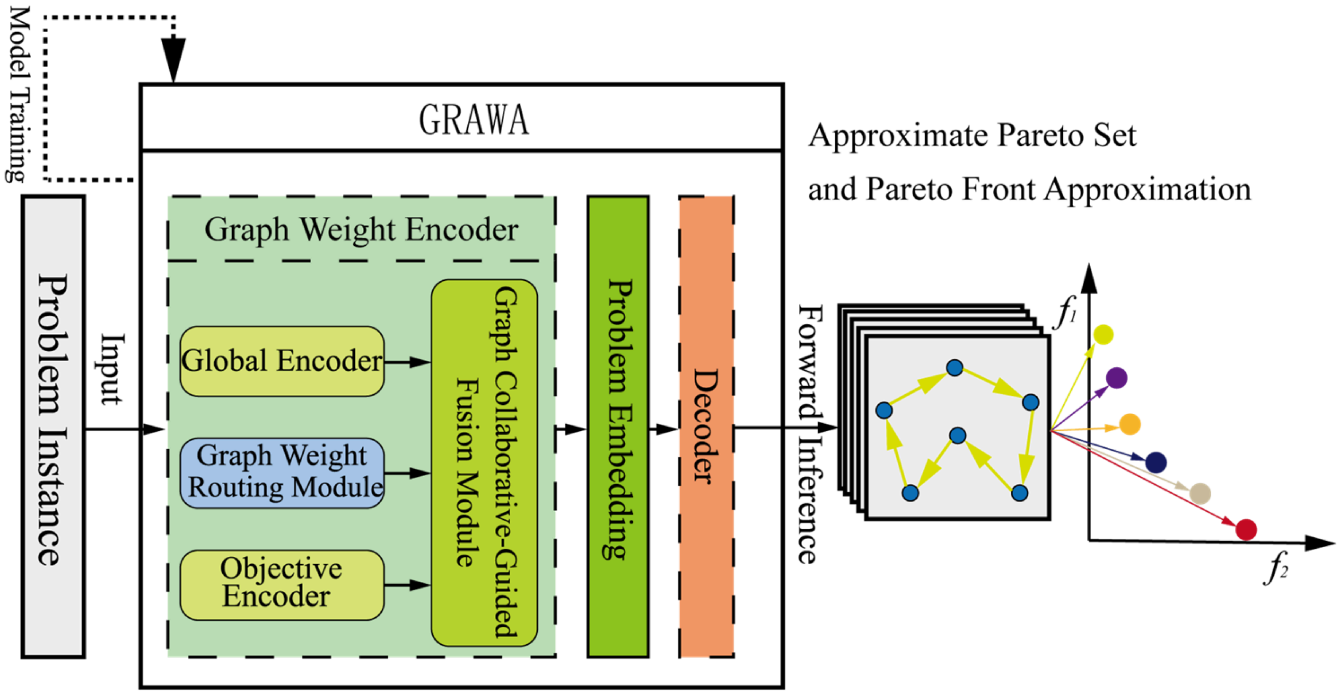


Figure 4. Framework of GRAWA for generating an approximate Pareto set and its corresponding Pareto front approximation. The colored circles in the objective space represent the objective vectors of tours generated for different preference vectors, with colors distinguishing the preferences.

The GRAWA model is trained using the REINFORCE algorithm [27]. First, the model parameters θ are initialized as specified in Section 4.2. A fixed set of N preference vectors is constructed on the unit simplex using the lattice rule in Section 3.1.2. We use $N = 101$ for bi-objective problems and $N = 91$ for tri-objective problems. The preference vectors and their base adjacency are shared across training instances and remain fixed throughout training. The explicit graph-construction procedure is provided in Appendix C. Its overall training procedure is outlined in Algorithm 1.

Algorithm 1 GRAWA Model Training

Input: Training distribution φ, number of epochs E, batches per epoch T, batch size B, objective count M, Top-p threshold p, initial learning rate η_0, and decay factor λ, fixed inverse temperature $\beta = 1.0$.
Output: Trained model parameters θ^*.

Initialize θ using the initialization rules in Section 4.2;
Generate the fixed preference set $W = \{w^j\}_{j=1}^{N}$ using Section 3.1.2;
Construct the fixed preference-graph adjacency A_w using Algorithm A1;
Initialize Adam using the configuration in Section 4.2; set $\eta \leftarrow \eta_0$;
for $e = 1$ **to** E **do**
  for $\ell = 1$ **to** T **do**
    // Sample a batch of problem instances
    $\{S_i\}_{i=1}^{B} \sim \text{SampleBatch}(\varphi, B)$;
    // Generate one tour per instance and preference
    for each $i \in \{1,\ldots,B\}, j \in \{1,\ldots,N\}$ **do**
      $x_i^j \sim p_\theta(\cdot \mid S_i, w^j; W, A_w)$;
      $q_i^j \leftarrow \log p_\theta(x_i^j \mid S_i, w^j; W, A_w)$;
      Compute $C_i^j \leftarrow g(x_i^j \mid w^j)$, including the closing edge;
    end for
    // Compute the Top-p baseline using Algorithm 2
    for $i = 1$ **to** B **do**
      $\mathcal{X}_i \leftarrow \{x_i^k\}_{k=1}^{N}$;
      for $j = 1$ **to** N **do**
        $b_i^j \leftarrow \text{Top-pBaseline}(\mathcal{X}_i, w^j, p, \beta)$;
      end for
    end for
    // Compute the cost-based update and optimize the model
    Clear the accumulated parameter gradients;
    $\hat{g}_\theta \leftarrow \frac{1}{BN} \sum_{i=1}^{B} \sum_{j=1}^{N} \text{stopgrad}(C_i^j - b_i^j) \nabla_\theta q_i^j$;
    Clip the global L2 norm of $\hat{g}_\theta$ to 1.0;
    $\theta \leftarrow \text{AdamStep}(\theta, \hat{g}_\theta, \eta)$;
  end for
  // Decay the learning rate after each epoch
  $\eta \leftarrow \lambda \eta$;
end for
return $\theta^* \leftarrow \theta$;

We use $E = 100$, $B = 50$, and $T = 200$, corresponding to 10,000 training instances per epoch and one optimizer update per batch. The initial learning rate is $\eta_0 = 5 \times 10^{-4}$ and the decay factor is $\lambda = 0.99$. The index e denotes an epoch, ℓ denotes a batch within an epoch, i indexes problem instances, and j indexes the current preference. The candidate-tour index k is distinct from j. The fixed preference set W and adjacency A_w are included explicitly in the policy notation here and are suppressed elsewhere for brevity. The operator stopgrad treats the cost-minus-baseline coefficient as constant during differentiation, while gradients propagate through the policy log-probability. AdamStep denotes an optimizer descent step with its moment estimates retained across batches.

Subsequently, the model is trained for E epochs, with T batches of B problem instances sampled from φ per epoch. One Adam update is performed per batch, and the learning rate is multiplied by λ after each epoch. Then, the problem instances $\{S_1,\ldots,S_B\}$ and the weight vectors $\{w^1,\ldots,w^N\}$ are input into the GRAWA model to generate a solution set $\{x_i^1, \ldots, x_i^N\}$. For an instance S_i and a preference vector w^j, tour construction is formulated

as a finite-horizon Markov decision process (MDP). The instance and preference remain fixed throughout an episode. After t selections, the state is defined as

$$s_t = (S_i, w^j, \pi_t, \mathcal{U}_t), \qquad \pi_t = (\pi_1, \ldots, \pi_t), \qquad \mathcal{U}_t = \{1, \ldots, n\} \setminus \{\pi_1, \ldots, \pi_t\} \tag{16}$$

where π_t is the ordered partial tour, $\mathcal{U}_t$ is the set of unvisited cities, and n is the number of cities. The initial state has an empty partial tour and all cities unvisited. At step t, the action $a_t \in \mathcal{U}_t$ selects the next city. The deterministic transition appends a_t to the partial tour and removes it from $\mathcal{U}_t$. The first city is selected by the policy from all n cities, and subsequent actions are restricted to the remaining unvisited cities. An episode terminates after n selections; the return edge from the last city to the first city is then included to form a closed tour.

For the completed tour $x = (\pi_1, \ldots, \pi_n)$, the scalarized cost is

$$C_i^j(x) = g(x \mid w^j) = \sum_{m=1}^{M} w_m^j f_m(x), \qquad f_m(x) = \sum_{t=1}^{n} c^{(m)}_{\pi_t, \pi_{t+1}}, \qquad \pi_{n+1} = \pi_1. \tag{17}$$

where $c_{uv}^{(m)}$ is the travel cost between cities u and v for objective m. The training objective is to minimize the expected scalarized tour cost. Equivalently, it can be expressed using an undiscounted terminal reward:

$$r_{t+1} = \begin{cases} 0, & 0 \le t < n-1, \\ -C_i^j(x), & t = n-1, \end{cases} \qquad \gamma = 1. \tag{18}$$

where r_{t+1} is the reward associated with action a_t and γ is the discount factor. Thus, the episodic return equals the negative total tour cost, including the closing edge. During training, actions are sampled from the masked policy distribution. The cost-based update below uses the completed-tour cost and its Top-p baseline; it is not an update based on HV. The stochastic update over B problem instances and N preference vectors is defined as follows:

$$\nabla_\theta \mathrm{G}(\theta) \approx \frac{1}{BN} \sum_{i=1}^{B} \sum_{j=1}^{N} \left[\left(g(\mathbf{x}_i^j | \mathbf{w}^j) - b(S_i, \mathbf{w}^j) \right) \nabla_\theta \log p_\theta(\mathbf{x}_i^j | S_i, \mathbf{w}^j) \right] \tag{19}$$

where $g(\mathbf{x}_i^j | \mathbf{w}^j)$ is the scalarized objective function of the subproblem; $b(S_i, \mathbf{w}^j)$ is the adaptive Top-p baseline; $\nabla_\theta \log p_\theta(\mathbf{x}_i^j | S_i, \mathbf{w}^j)$ represents the logarithmic policy-gradient term, where $p_\theta(\mathbf{x}_i^j | S_i, \mathbf{w}^j)$ is the conditional probability of the parameterized policy model generating a feasible solution $\mathbf{x}_i^j$ given S_i and $\mathbf{w}^j$.

In our implementation, the Top-p baseline is constructed from the candidate tours generated for the same instance, including the current tour used in the corresponding score-function term. Although the candidate costs are detached from the computational graph, this does not remove the statistical dependence between the baseline and the current tour. Consequently, the update in Equation (19) is not guaranteed to be an unbiased estimator of the gradient of the original expected scalarized cost. We therefore treat Top-p as a heuristic training baseline. Appendix A.1 provides the expectation decomposition and bias analysis, while Appendix A.2 evaluates the practical effect of including the current tour through a performance comparison with a leave-one-out variant. We retain the original formulation as the default to preserve the training procedure used in the main experiments, while treating leave-one-out as an alternative baseline whose theoretical merits and empirical comparison are discussed in Appendix A.

3.1. Graph Weight Encoder

As illustrated in Figure 5, the graph weight encoder contains three branches: a global encoder, objective-specific encoders, and a graph weight routing module. In this paper,

"graph weight encoder" denotes this complete three-branch encoder, whereas "graph weight routing module" denotes its preference-graph branch. The global encoder extracts city representations from the concatenated objective features, whereas the objective-specific encoders preserve the information associated with individual objectives. The graph weight routing module operates on a separate preference graph, whose nodes represent weight vectors rather than cities. It learns preference representations through graph propagation, score-gated pooling, and unpooling with a skip connection. The resulting preference representations are combined with the global and objective-specific city embeddings by the graph co-guided fusion module.

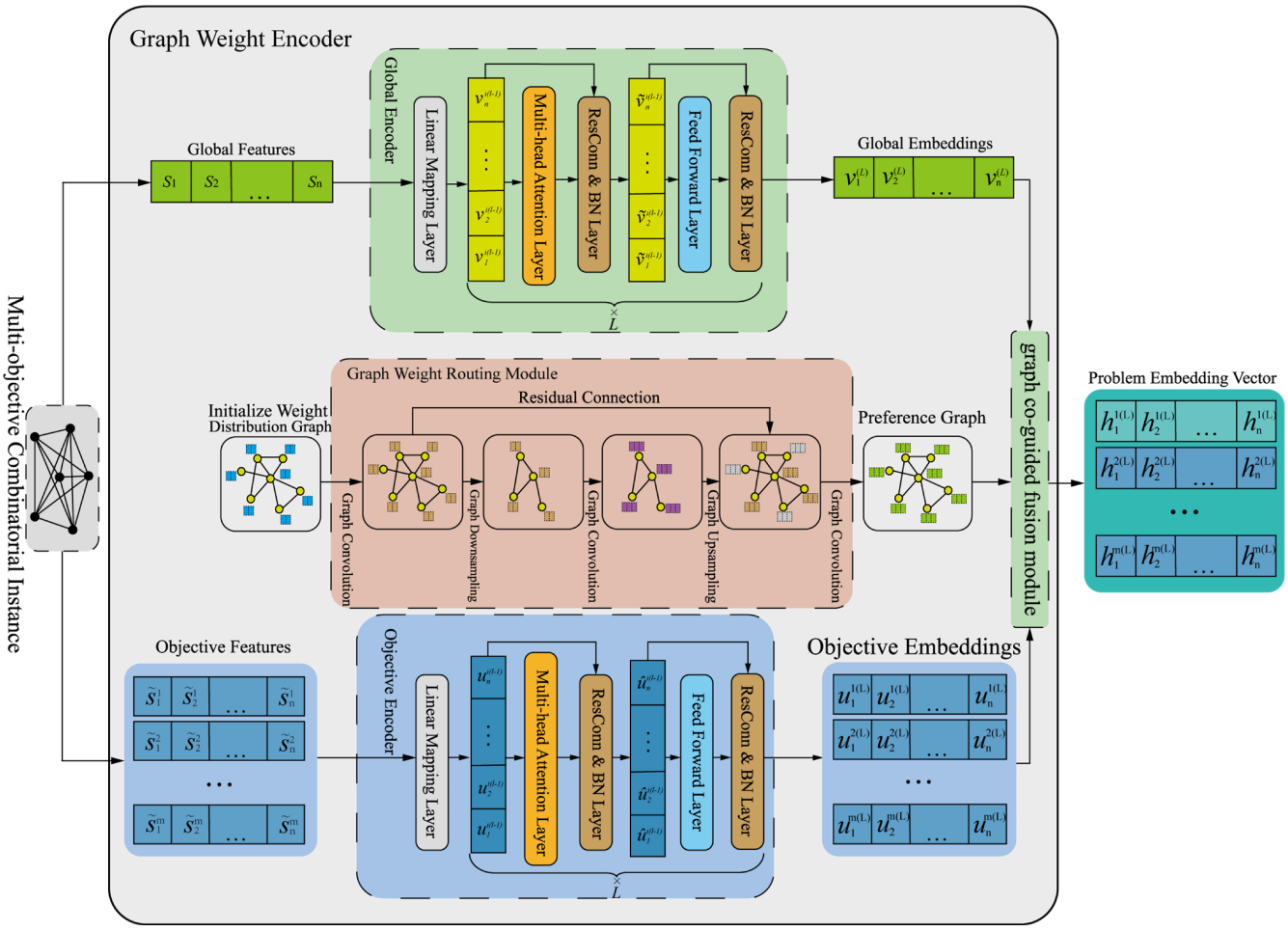


Figure 5. The framework of the graph weight encoder. The global and objective-specific branches produce city embeddings, whereas the routing branch produces preference-node embeddings. In the routing branch, selected node features are multiplied by their sigmoid scores before low-resolution propagation. Saved indices restore the original node positions, and a skip connection supplies features to all preference nodes.

The base preference graph is fixed once the weight vectors are generated. Its node representations and selected pooled subgraph are learned through the training process. Thus, the term adaptive refers here to feature learning and score-dependent node selection, rather than to the continuous reconstruction of the base adjacency matrix.

Meaning of the preference representation. The routing input consists of the preference matrix $W \in \mathbb{R}^{N\times M}$ and the fixed adjacency $A_w \in \{0,1\}^{N\times N}$. Its initial node features and output can be summarized as

$$Z^{(0)} = WE_{\text{in}} + \mathbf{1}_N b_{\text{in}}^{\top}, \qquad Z_w = \mathcal{R}_\psi(A_w, Z^{(0)}) \in \mathbb{R}^{N\times d}. \tag{20}$$

where $E_{\text{in}} \in \mathbb{R}^{M\times d}$ and $b_{\text{in}} \in \mathbb{R}^{d}$ are the learnable input-projection parameters, $\mathbf{1}_N$ is an all-ones column vector, and $\mathcal{R}_\psi$ denotes the sequence of graph propagation, score-gated pooling, index-based restoration, and skip-connected propagation defined in Section 3.1.2. The rows of Z_w are learned representations of scalarization preferences in the context of the specified preference graph. They are not externally supplied semantic annotations, objective-value vectors, or representations extracted directly from a Pareto front.

The parameters are trained through the downstream tour-construction policy. However, the routing branch has no instance-specific city input: for fixed parameters and a fixed preference set, its deterministic output is shared by instances using that set. Instance-specific conditioning occurs when the preference representation is fused with the city features in Section 3.2. We therefore use the term "learned preference representations" instead of "direction-guided key semantic information."

3.1.1. Global Encoder and Objective-Specific Encoder

For clarity, the following formulation describes one problem instance; the batch dimension is omitted. Let n denote the number of cities, M the number of objectives, N the number of preference vectors, and d the embedding dimension. For objective $m \in \{1,\ldots,M\}$, the raw city-feature matrix is denoted by X_m, with two input coordinates for a Euclidean objective and one input coordinate for a one-dimensional mixed-distance objective. The global input is obtained by concatenating these features along the feature dimension.

$$X_g = [X_1 \| \cdots \| X_M] \in \mathbb{R}^{n\times q}, \qquad q = \sum_{m=1}^{M} q_m. \tag{21}$$

where X_m has shape n by q_m, q_m is the input feature dimension of objective m, and the concatenation preserves the common city ordering across objectives.

The global encoder applies an input projection and L attention–feed-forward blocks of the form given in Equations (10) and (11). Each objective-specific encoder uses the same block structure with its own input projection and parameters. Their outputs are

$$V = \text{Enc}_g(X_g) \in \mathbb{R}^{n\times d}, \qquad U_m = \text{Enc}_m(X_m) \in \mathbb{R}^{n\times d}. \tag{22}$$

where V contains global city embeddings and U_m contains city embeddings for objective m. We use $L = 3$ and $d = 128$, and each attention layer uses eight heads. The attention–feed-forward blocks retain the residual connections and batch normalization defined in Equations (10) and (11). The graph and fusion operations described below use the same output dimension d.

3.1.2. Graph Weight Routing Module

Preference graph construction. Let W contain the N preference vectors as rows. The preference vectors belong to the unit simplex:

$$W = [(w^1)^\top; \ldots; (w^N)^\top] \in \mathbb{R}^{N\times M}, \qquad w_m^j \geq 0, \quad \sum_{m=1}^{M} w_m^j = 1. \tag{23}$$

where w_m^j is the relative preference assigned to objective m by the j-th weight vector. These components are neither objective costs nor city coordinates. For bi-objective problems, we use the 101 vectors $(t/100, 1 - t/100)$, with $t = 0,\ldots,100$. For tri-objective problems, we use all vectors $(a/12, b/12, c/12)$, where a,b,c are nonnegative integers satisfying $a + b + c = 12$, giving 91 vectors. The vectors are stored in a fixed lexicographic order, and node v_j corresponds to preference vector w^j, whose transpose is the j-th row of W.

We construct an undirected, unweighted preference graph $G_w = (\mathcal{V}_w, \mathcal{E}_w, A_w)$, where $\mathcal{V}_w = \{v_1,\ldots,v_N\}$ contains one node for each preference vector and A_w is the base adjacency

matrix. The graph is constructed directly from the preference vectors, without using the objective-space points P_i in Figure 3 or knowledge of the true Pareto front. For nodes j and r, their distance is defined in preference space as

$$d_w(j,r) = \|w^j - w^r\|_2 = \sqrt{\sum_{m=1}^{M} \left(w_m^j - w_m^r\right)^2}, \qquad j \neq r. \tag{24}$$

where j,$r \in \{1,\ldots,N\}$ index the preference nodes, N is the number of preference vectors, M is the number of objectives, and $\|\cdot\|_2$ denotes the Euclidean norm. The distance is computed from the M preference components of w^j and w^r, not from the objective values P_j and P_r or the schematic node positions in Figure 3b. Appendix D provides a sensitivity bound using this same Euclidean preference distance. The bound concerns scalarized-cost transfer between subproblems and does not identify graph edges with neighboring points on the Pareto front. For each node j, we sort all other node indices $r \neq j$ in ascending order by $(d_w(j,r),r)$, using distance as the primary key and the fixed node index to break distance ties. The first $k_w = 4$ indices form the nearest-neighbor selection set $\mathcal{N}_w(j)$. The neighborhood-selection size k_w is a predefined graph-construction hyperparameter fixed at four in the reported experiments; it is not learned during training or determined by the number of cities. The selection relation need not be symmetric. We construct the final undirected graph by taking the union of the directed nearest-neighbor relations:

$$(A_w)_{jr} = \begin{cases} 1, & j \neq r \text{ and } [r \in \mathcal{N}_w(j) \text{ or } j \in \mathcal{N}_w(r)], \\ 0, & \text{otherwise}, \end{cases} \qquad A_w \in \{0,1\}^{N\times N}. \tag{25}$$

where $(A_w)_{jr}$ is the adjacency entry between nodes j and r. An entry of one indicates that at least one node selects the other as a nearest neighbor, whereas an entry of zero indicates no edge. The edge set is $\mathcal{E}_w = \left\{\left\{v_j, v_r\right\} : j < r, (A_w)_{jr} = 1\right\}$, with each unordered pair representing one undirected edge. The union rule gives $A_w = A_w^\top$, and the condition $j \neq r$ gives $(A_w)_{jj} = 0$. Thus, the base graph is undirected and unweighted, with symmetric binary adjacency and no self-loops. The term "weight" refers to the node-associated preference vectors, not to real-valued edge weights. All adjacency entries are non-trainable. Self-loops are added separately within graph convolution, and the base graph remains fixed for all problem instances using the same preference set.

The graph neighborhood $B(i)$ used in Figure 3 is defined after symmetrization as

$$B(i) = \left\{r \in \{1,\ldots,N\} \setminus \{i\} : (A_w)_{ir} = 1\right\} = \mathcal{N}_w(i) \cup \{r \neq i : i \in \mathcal{N}_w(r)\}. \tag{26}$$

where i denotes any preference-node index, $\mathcal{N}_w(i)$ is the set of four indices selected before symmetrization, and $B(i)$ is the set of all node indices adjacent to i in the final graph. Consequently, $r \in B(i)$ if and only if $i \in B(r)$, and $|B(i)|$ may exceed four. The set $B(i)$ is a graph-neighborhood index set, not a continuous region on the Pareto front. Appendix C provides an explicit construction algorithm consistent with these definitions.

The initial preference-node features are obtained by a learnable affine projection:

$$Z^{(0)} = WE_{\text{in}} + \mathbf{1}_N b_{\text{in}}^\top \in \mathbb{R}^{N\times d}. \tag{27}$$

where $E_{\text{in}} \in \mathbb{R}^{M\times d}$ and $b_{\text{in}} \in \mathbb{R}^d$ are trainable parameters, and $\mathbf{1}_N$ is the all-ones vector. These features encode the preference vectors and do not directly contain city coordinates or sampled tours.

Graph propagation. For a graph with N_ℓ nodes, let $A \in \mathbb{R}^{N_\ell\times N_\ell}$ denote its adjacency and $Z \in \mathbb{R}^{N_\ell\times d}$ its node features. We define

$$\hat{A} = A + I_{N_\ell}, \qquad \hat{D} = \operatorname{diag}(\hat{A}\mathbf{1}_{N_\ell}). \tag{28}$$

$$\mathrm{GCN}_{\Theta}(A, Z) = \mathrm{ReLU}\big(\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}Z\Theta\big), \qquad \Theta \in \mathbb{R}^{d\times d}. \tag{29}$$

where I_{N_ℓ} is the identity matrix, $\mathbf{1}_{N_\ell}$ is the all-ones vector, and $\mathrm{diag}(\cdot)$ constructs a diagonal matrix from its vector argument. The matrix $\hat{A}$ denotes the adjacency with self-loops, and $\hat{D}$ is its corresponding degree matrix. The matrix $\Theta \in \mathbb{R}^{d\times d}$ contains trainable parameters, and $\mathrm{ReLU}(\cdot)$ denotes the element-wise rectified linear activation. The resulting node-feature matrix has shape $N_\ell \times d$. The degree matrix is computed after adding self-loops, and the normalization is recomputed for each graph resolution. We adopt the standard graph-convolution propagation rule of Kipf and Welling [28], with self-loop addition, symmetric degree normalization, a trainable feature transformation, and ReLU activation. This propagation operator is an established component rather than a new operator introduced in this work. The representation before pooling is

$$H = \mathrm{GCN}_{\Theta_{\mathrm{enc}}}(A_w, Z^{(0)}) \in \mathbb{R}^{N\times d}. \tag{30}$$

where Θ_{enc} is a trainable parameter matrix.

Score-gated pooling and trainability. We use one pooling stage with retention ratio $\rho = 0.5$, retaining $K = \lceil \rho N \rceil$ preference nodes. A trainable projection vector $a \in \mathbb{R}^d$ assigns a score to each node:

$$y = \frac{Ha}{\|a\|_2 + \varepsilon} \in \mathbb{R}^N, \qquad I = \mathrm{TopKIndices}(y, K), \qquad \varepsilon = 10^{-8}. \tag{31}$$

where the vector a is initialized to a nonzero random value. The ordered index list $I = (I_1,\ldots,I_K)$ contains the selected nodes; tied scores are resolved by node index. The same index ordering is used for the selected features, the induced adjacency, and subsequent unpooling.

The selected score values are retained and used to gate the selected features:

$$s = \mathrm{sigmoid}(y[I]) \in \mathbb{R}^K, \ \widetilde{Z} = H[I,:] \odot (s\mathbf{1}_d^\top) \in \mathbb{R}^{K\times d}. \tag{32}$$

$$A_{\mathrm{pool}} = A_w[I,I] \in \mathbb{R}^{K\times K}. \tag{33}$$

where the gate of each selected node is broadcast across its d feature channels. The pooled graph is the induced subgraph of the selected nodes, without additional off-diagonal edges. Self-loops and degree normalization are applied to A_{pool} when the next graph convolution is evaluated. The projection-based node selection, feature gating, and index-based restoration adapt the gPool and gUnpool principles of Gao and Ji [29] to the preference graph. We specify the pooling ratio, gate, selected-node ordering, and restoration rule for reproducibility; these operations are not claimed as independently novel pooling or unpooling operators.

The Top-k indices are discrete and are not differentiated. The gradient path to a is instead provided by the selected score values in the multiplicative gate. To make this distinction explicit, let $h_j^\top = H[j,:]$, $c = \|a\|_2 + \varepsilon$ and $a \neq 0$. For a locally fixed selected index set,

$$\nabla_a y_j = \frac{h_j}{c} - \frac{(h_j^\top a)a}{\|a\|_2 c^2}. \tag{34}$$

For the selected node $j = I_t$, its pooled column-vector representation is $\tilde{z}_t = \sigma(y_j)h_j$, and therefore

$$\frac{\partial \tilde{z}_t}{\partial a^\top} = h_j\sigma(y_j)(1-\sigma(y_j))\big(\nabla_a y_j\big)^\top. \tag{35}$$

Thus, a training gradient reaching $\tilde{z}_t$ can propagate to a through the score gate, without differentiating the selected indices. This is a local, piecewise-differentiable feature path; it does not make Top-k selection globally differentiable or guarantee a nonzero gradient at every training step.

Low-resolution propagation and unpooling. We apply graph convolution to the gated features at the pooled resolution:

$$Y = \mathrm{GCN}_{\Theta_{\mathrm{mid}}}(A_{\mathrm{pool}}, \widetilde{Z}) \in \mathbb{R}^{K\times d}, \tag{36}$$

where $\Theta_{\mathrm{mid}} \in \mathbb{R}^{d\times d}$ is a separate trainable matrix. Using the saved index list I, the features are then restored to the original preference-node positions:

$$(Z_{\mathrm{up}})_{j,:} = \begin{cases} Y_{t,:}, & j = I_t, \quad t = 1, \ldots, K, \\ 0, & j \notin I, \end{cases} \qquad Z_{\mathrm{up}} \in \mathbb{R}^{N\times d}. \tag{37}$$

Unpooling restores the original indexing and tensor size, rather than recovering the information discarded during pooling. We add a skip connection from the pre-pooling features and apply graph convolution on the base graph:

$$Z_w = \mathrm{GCN}_{\Theta_{\mathrm{out}}}(A_w, Z_{\mathrm{up}} + H) \in \mathbb{R}^{N\times d}. \tag{38}$$

where $\Theta_{\mathrm{out}} \in \mathbb{R}^{d\times d}$ is independently trainable. The skip connection provides features for all original preference nodes, including those omitted by pooling. The final convolution propagates the restored information over A_w. The row $Z_w[j:]$ is the learned representation of preference w^j.

The graph weight routing module contains one pooling stage and three graph-convolution layers: one before pooling, one on the pooled graph, and one after index-based restoration and the additive skip connection. Each graph convolution maps 128 input channels to 128 output channels and uses ReLU activation. With a pooling ratio of 0.5, the retained-node count is 51 for $N = 101$ and 46 for $N = 91$. Section 3.2 applies one additional 128-to-128 graph convolution to Z_w before preference-to-city broadcasting. Thus, the number of pooling stages is distinct from both the graph-convolution depth and the three-layer city encoder.

The base adjacency A_w remains fixed. The representations, node scores, and selected pooled subgraph can change as the parameters are updated. Thus, the module learns adaptive preference representations over a fixed base topology rather than learning a new full adjacency matrix at every iteration. For a fixed preference set and fixed model parameters, the routing representation is shared across problem instances in deterministic evaluation, because this branch does not receive city features as input.

Role of graph weight routing. In this work, "routing" refers to the organization of preference-feature flow: propagation over the fixed preference graph, score-based selection of preference nodes, propagation on the retained subgraph, and restoration to the original preference-node positions. It does not denote city-tour routing, learned edge weights, or continuous reconstruction of the base graph. Restoring the original indexing provides a representation for each requested preference, which can then be selected and broadcast to the cities in Section 3.2.

The architectural contribution is the use of this graph-based preference encoder within a shared MOTSP policy, together with its connection to preference-conditioned city-feature fusion. The graph nodes represent scalarization preferences rather than cities. The restored representations condition the fusion of city features before autoregressive tour construction. Thus, the contribution is the task-specific organization and integration of established graph-learning components, rather than a replacement for standard GCN propagation or a new graph pooling operator.

3.2. Graph Co-Guided Fusion Module

Figure 6 illustrates the architecture and information flow of the graph co-guided fusion module. The fusion module combines the global city embeddings $V \in \mathbb{R}^{n\times d}$ and

objective-specific city embeddings $U_m \in \mathbb{R}^{n\times d}$ constructed in Section 3.1.1 with the preference representations $Z_w \in \mathbb{R}^{N\times d}$ produced in Section 3.1.2. Here, n is the number of cities, N is the number of preferences, and $d = 128$. The formulation below describes one preference w^j for one instance, with the instance-batch dimension omitted. Operations are repeated or batched over the N preferences. A consolidated tensor-shape flow is provided in Appendix B.

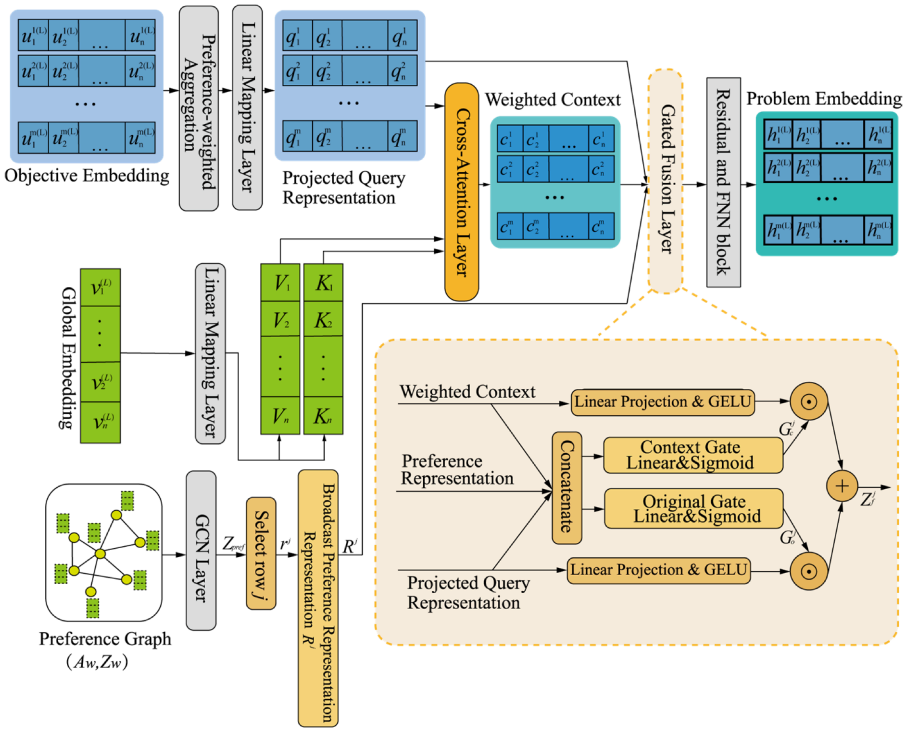


Figure 6. Graph co-guided fusion module. Preference-conditioned objective embeddings query the global city embeddings through cross-attention. The corresponding preference-node representation is broadcast across cities and used with the context and original branch to compute the fusion gates. The fused features are processed by the residual, normalization, and feed-forward operations specified in Section 3.2.

Preference-conditioned cross-attention. The objective-specific embeddings are first aggregated using the current preference:

$$U^j = \sum_{m=1}^{M} w_m^j U_m \in \mathbb{R}^{n\times d}. \tag{39}$$

where $U_m \in \mathbb{R}^{n\times d}$ contains the city embeddings for objective m. All objective embeddings use the same city ordering. Thus, U^j combines the representations of each city according to the preference w^j, making the query explicitly preference-dependent.

For attention head h, the query, key, and value matrices are

$$Q_h^j = U^j W_{Q,h}, \qquad K_h = V W_{K,h}, \qquad V_h = V W_{V,h}. \tag{40}$$

where $W_{Q,h}, W_{K,h}, W_{V,h} \in \mathbb{R}^{d\times d_h}$ are the projection matrices for attention head h. We use $H_{att} = 8$ heads with $d_h = d/H_{att} = 16$. Each projected matrix has shape n by d_h. The projected query representation used as the original fusion branch is

$$Z_o^j = [Q_1^j \| \cdots \| Q_{H_{\text{att}}}^j] \in \mathbb{R}^{n\times d}. \tag{41}$$

where $\|$ denotes concatenation along the feature dimension and $H_{\text{att}} d_h = d = 128$. Thus, the original branch is the concatenated projected query representation, before cross-attention

aggregation. It is used in the fusion-gate input, the original-branch transformation, and the first residual connection below. The global city embedding V is distinct from V_h, which denotes its value projection for attention head h. The attention output is

$$C_h^j = \mathrm{softmax}_{\mathrm{key}}\left(Q_h^j K_h^\top / \sqrt{d_h}\right) V_h. \tag{42}$$

where the attention matrix has shape $n \times n$. For query-city index u and key-city index v, its entries are

$$\alpha_{h,uv}^j = \frac{\exp(S_{h,uv}^j)}{\sum_{v'=1}^{n} \exp(S_{h,uv'}^j)}, \qquad S_h^j = \frac{Q_h^j K_h^T}{\sqrt{d_h}}. \tag{43}$$

where the normalization is over all key cities v' for each fixed query city u; hence $\sum_{v=1}^{n} \alpha_{h,uv}^j = 1$. The output C_h^j has shape $n \times d_h$. The attention heads are concatenated and projected:

$$Z_c^j = [C_1^j \| \cdots \| C_{H_{\mathrm{att}}}^j] W_O \in \mathbb{R}^{n \times d}. \tag{44}$$

where W_O has shape d by d and concatenation is along the feature dimension. The projections in this cross-attention block use no biases. Attention is evaluated over all cities; the decoder subsequently applies its visited-node mask during tour construction.

Graph-guided gated fusion. Before feature fusion, we apply an additional graph convolution to the preference representations obtained from the graph weight routing module:

$$Z_{\mathrm{pref}} = \mathrm{GCN}_{\Theta_{\mathrm{pref}}}(A_w, Z_w) \in \mathbb{R}^{N \times d}. \tag{45}$$

where $A_w \in \mathbb{R}^{N \times N}$ is the fixed preference-graph adjacency, $Z_w \in \mathbb{R}^{N \times d}$ contains the output representations of the graph weight routing module, and $\Theta_{\mathrm{pref}} \in \mathbb{R}^{d \times d}$ is the trainable parameter matrix of this additional layer. The graph convolution follows the self-loop addition, symmetric normalization, and activation defined for graph propagation in the preceding subsection. For the current preference w^j, we extract the corresponding node representation and broadcast it across cities:

$$r^j = Z_{\mathrm{pref}}[j,:] \in \mathbb{R}^{1 \times d}, \qquad R^j = \mathbf{1}_n r^j \in \mathbb{R}^{n \times d}. \tag{46}$$

where r^j is the representation of the j-thpreference node, n is the number of cities, and $\mathbf{1}_n$ is the all-ones column vector. Broadcasting aligns the preference-level representation with the city-level context representation $Z_c^j \in \mathbb{R}^{n \times d}$ for subsequent feature fusion. Specifically, $R_{u,:}^j = r^j$ for every city $u = 1,\ldots,n$. This replication does not change the feature dimension. In the affine transformations below, $\mathbf{1}_n b^\top$ likewise broadcasts a feature-wise bias vector across the city rows.

The broadcast representation R^j is shared across city positions for the current preference. City-dependent gating is obtained only after this representation is concatenated with the city-level features in F^j. This distinguishes the learned preference information supplied by the routing branch from the instance-specific information supplied by the city encoders.

We concatenate the three representations along the feature dimension:

$$F^j = [R^j \| Z_c^j \| Z_o^j] \in \mathbb{R}^{n \times 3d}. \tag{47}$$

The context and original branches use separate sigmoid gates:

$$G_c^j = \sigma(F^j W_{g,c} + \mathbf{1}_n b_{g,c}^\top), \qquad G_o^j = \sigma(F^j W_{g,o} + \mathbf{1}_n b_{g,o}^\top). \tag{48}$$

where $W_{g,c}, W_{g,o} \in \mathbb{R}^{3d \times d}$ and $b_{g,c}, b_{g,o} \in \mathbb{R}^d$ are trainable parameters. Both gates have shape $n \times d$, with one coefficient per city and feature channel. They are computed independently and

are not constrained to sum to one. Here, $\sigma(z) = 1/(1 + \exp(-z))$ is applied element-wise. The products $G_c^j \odot T_c^j$ and $G_o^j \odot T_o^j$ combine tensors of identical shape $n \times d$; no additional gate broadcasting is used.

The two branches are transformed separately:

$$T_c^j = \mathrm{GELU}(Z_c^j W_{t,c} + \mathbf{1}_n b_{t,c}^\top), \qquad T_o^j = \mathrm{GELU}(Z_o^j W_{t,o} + \mathbf{1}_n b_{t,o}^\top). \tag{49}$$

where $W_{t,c}$,$W_{t,o} \in \mathbb{R}^{d\times d}$ and $b_{t,c}$,$b_{t,o} \in \mathbb{R}^d$. Their gated fusion is

$$Z_f^j = G_c^j \odot T_c^j + G_o^j \odot T_o^j \in \mathbb{R}^{n\times d}. \tag{50}$$

where $\odot$ denotes element-wise multiplication.

Residual connections and normalization. The gated fused features are combined with the projected query representation and processed by two residual blocks:

$$T^j = \mathrm{BN}_1(Z_o^j + Z_f^j), \qquad E^j = \mathrm{BN}_2(T^j + \mathrm{FFN}(T^j)). \tag{51}$$

where BN_1 and BN_2 denote batch normalization after the first and second residual additions, respectively. The first residual adds the untransformed projected query representation Z_o^j to Z_f^j, while the second adds the feed-forward output to its input T^j. All terms in these additions have shape $n \times d$, with $d = 128$, and no broadcasting is required for the residual sums.

The feed-forward block is defined as

$$\mathrm{FFN}(T^j) = \mathrm{ReLU}\big(T^j W_1 + \mathbf{1}_n b_1^\top\big) W_2 + \mathbf{1}_n b_2^\top. \tag{52}$$

where $W_1 \in \mathbb{R}^{d\times d_{\mathrm{ff}}}$, $W_2 \in \mathbb{R}^{d_{\mathrm{ff}}\times d}$, $b_1 \in \mathbb{R}^{d_{\mathrm{ff}}}$, and $b_2 \in \mathbb{R}^d$ are trainable parameters. We use $d_{\mathrm{ff}} = d = 128$, giving a $128 \to 128 \to 128$ block with element-wise ReLU between the two affine transformations. The terms $\mathbf{1}_n b_1^\top$ and $\mathbf{1}_n b_2^\top$ broadcast the corresponding biases across city rows.

For batch normalization during training, the instance, preference, and city axes are merged. Specifically, each residual-sum tensor of shape $B \times N \times n \times d$ is reshaped to $L_{\mathrm{BN}} \times d$, where $L_{\mathrm{BN}} = BNn$. Let $X^{(a)}$ denote this flattened input to BN_a, for $a \in \{1,2\}$. Its channel-wise normalization is

$$[\mathrm{BN}_a(X^{(a)})]_{u,c} = \gamma_{a,c} \frac{X_{u,c}^{(a)} - \mu_{a,c}}{\sqrt{v_{a,c} + \epsilon_{\mathrm{BN}}}} + \delta_{a,c}. \tag{53}$$

where $u = 1,\ldots,L_{\mathrm{BN}}$ indexes the merged instance–preference–city positions, $c = 1,\ldots,d$ indexes feature channels, γ_a,$\delta_a \in \mathbb{R}^d$ are the affine scale and shift parameters, and $\epsilon_{\mathrm{BN}} > 0$ is the numerical-stability constant. The training forward-pass statistics are

$$\mu_{a,c} = \frac{1}{L_{\mathrm{BN}}} \sum_{u=1}^{L_{\mathrm{BN}}} X_{u,c}^{(a)}, \qquad v_{a,c} = \frac{1}{L_{\mathrm{BN}}} \sum_{u=1}^{L_{\mathrm{BN}}} (X_{u,c}^{(a)} - \mu_{a,c})^2. \tag{54}$$

where $X^{(1)}$ is obtained from the first residual sum $Z_o + Z_f$ and $X^{(2)}$ from the second residual sum $T + \mathrm{FFN}(T)$, after collecting all instances and preferences in the batch. Each normalization uses the statistics of its own input. The normalized tensor is then reshaped back to $B \times N \times n \times d$. Thus, each feature channel is normalized over the merged positions, rather than across the d features of an individual city. The final representation for one instance and preference is $E^j \in \mathbb{R}^{n\times 128}$, which is supplied to the decoder.

3.3. Adaptive Top-p Baseline

In REINFORCE, an appropriately chosen action-independent baseline can reduce gradient variance without changing the expected score-function update. The proposed

Top-p baseline is sample-dependent, and its statistical properties therefore require separate consideration.

The adaptive Top-p baseline first collects the solution set $\mathcal{X}_{all} = \{x_i^1, \ldots, x_i^N\}$ generated in parallel by the model for instance S_i and calculates the scalarized objective value $g(x|w^j)$ for each solution based on the current weight vector w^j. Subsequently, the algorithm converts the objective values into selection probabilities $q(x)$, sorts the candidate solutions according to their probabilities, and selects high-quality solutions whose cumulative probability reaches the threshold p to form $\mathcal{X}_{top-p}$. Finally, the weighted average of the objective values of the solutions in this set is used as the baseline $b(S_i,w^j)$. This construction reuses the candidate tours generated during parallel training. Here, "adaptive" refers exclusively to the number of candidates retained for each baseline evaluation. The cumulative probability threshold p is an externally selected hyperparameter that remains fixed throughout a training run; it is neither learned by gradient updates nor adjusted by a training schedule. Even with fixed p, the retained candidate count can vary across instances, preferences, and training iterations because the candidate-cost distribution changes. Its statistical properties are discussed in Appendix A.1.

Let $q_{i,(1)}^j \geq \cdots \geq q_{i,(N)}^j$ denote the candidate probabilities sorted for instance S_i and current preference w^j. The retained candidate count is

$$L_i^j(p) = \min\left\{\ell \in \{1, \ldots, N\} : \sum_{k=1}^{\ell} q_{i,(k)}^j \geq p\right\}, \qquad 0 < p \leq 1. \tag{55}$$

where $L_i^j(p)$ is recomputed from the current candidate probabilities at each baseline evaluation, while p remains unchanged during the run. The selected set is the shortest sorted prefix whose cumulative probability reaches or exceeds p. Thus, its size is data-dependent, whereas the threshold itself is fixed.

The candidate probabilities are obtained using the softmin transformation

$$q(x) = \frac{\exp[-\beta c(x)]}{\sum_{x' \in \mathcal{X}_{\text{all}}} \exp[-\beta c(x')]}, \qquad c(x) = g(x \mid w^j). \tag{56}$$

where $\beta > 0$ is a fixed inverse-temperature parameter that controls the concentration of probability mass on lower-cost candidate tours. Larger values of β yield a more concentrated distribution, whereas smaller values yield a flatter distribution. We set $\beta = 1.0$ and do not optimize it during training. Unlike the cumulative probability threshold p, which determines where the sorted candidate list is truncated, β controls the softmin probabilities used for both subset selection and the normalized weighted baseline. These probabilities are baseline-construction weights, not the autoregressive policy probabilities used to generate tours. The complete baseline-construction procedure is given in Algorithm 2.

Where $c(x) = g(x|w^j)$ is the scalarized cost of candidate tour x under the current preference w^j, and $\mathcal{X}_{\text{all}}$ is the set of candidate tours generated for the same instance. The parameter $\beta > 0$ is the inverse temperature of the softmin distribution. A larger β concentrates probability mass more strongly on lower-cost tours, whereas a smaller β produces a flatter distribution, approaching the uniform distribution as $\beta \to 0^+$. We use a fixed, non-trainable value $\beta = 1.0$. The threshold p determines the cumulative probability mass used to select the Top-p subset, while β controls the probabilities from which that subset is formed. The selected subset and its cardinality are recomputed at every call, but p and β are not updated by Algorithm 2. For each instance, the candidate set $\mathcal{X}_i$ contains all N sampled tours, including the current tour x_i^j. Algorithm 2 evaluates these candidates under the current preference w^j when constructing b_i^j. Thus, the candidate costs used for this baseline are $\{g(x_i^k \mid w^j)\}_{k=1}^N$, rather than the costs of the candidates under their respective generating preferences. Detaching the coefficient does not remove the statistical dependence between

the baseline and the current tour; the qualification concerning the sample-dependent update in Section 3 and Appendix A.1 therefore remains applicable.

Algorithm 2 Adaptive Top-p Baseline

Input: Candidate tours $\mathcal{X}_{\text{all}} = \{x_i^k\}_{k=1}^N$, current preference w^j, fixed cumulative probability threshold $p \in (0,1]$, and fixed inverse temperature $\beta > 0$.
Output: Baseline value $b(S_i, w^j)$.
// Collect the solution set generated in parallel by the GRAWA model for the instance
$\mathcal{X}_{all} \leftarrow \{x_i^1, \ldots, x_i^N\}$;
// Calculate the scalarized objective value for each solution based on the current weight vector
for each $x \in \mathcal{X}_{all}$ **do**
  $c(x) \leftarrow g(x|w^j)$;
end for
// Convert scalarized costs into softmin probabilities using β
for each $x \in \mathcal{X}_{all}$ **do**
  $q(x) \leftarrow \frac{\exp(-\beta \cdot c(x))}{\sum_{x' \in \mathcal{X}_{all}} \exp(-\beta \cdot c(x'))}$;
end for
// Sort the candidate solutions in descending order according to their probabilities
Sort all x in $\mathcal{X}_{all}$ in descending order by $q(x)$;
// Select the shortest sorted prefix reaching the fixed threshold p
$\mathcal{X}_{top-p} \leftarrow \emptyset$;
$P_{sum} \leftarrow 0$;
for each x in sorted $\mathcal{X}_{all}$ **do**
  $\mathcal{X}_{top-p} \leftarrow \mathcal{X}_{top-p} \cup \{x\}$;
  $P_{sum} \leftarrow P_{sum} + q(x)$;
  if $P_{sum} \geq p$ **then**
    break;
  end if
end for
// Use the weighted average of the objective values in this set as the baseline value
$Z \leftarrow \sum_{x \in \mathcal{X}_{top-p}} q(x)$;
$b(S_i, w^j) \leftarrow \sum_{x \in \mathcal{X}_{top-p}} \left[\frac{q(x)}{Z}\right] \cdot g(x|w^j)$;

4. Experimental Analysis

To comprehensively evaluate the proposed method, we first introduce the experimental settings in detail, covering the test problems, performance evaluation metrics, baseline algorithm selection, and parameter configurations. Subsequently, a comparative analysis between the proposed algorithm and current mainstream multi-objective combinatorial optimization algorithms is conducted on multiple benchmark problems to verify its performance superiority. Finally, the effectiveness of each core module in GRAWA is validated through ablation studies.

4.1. Test Problems and Evaluation Metrics

4.1.1. Test Problems

In the experimental study of this paper, two mainstream types of MOTSP test problems [15] are adopted for model training and algorithm testing. Let M denote the number of objectives. The specific instances are defined as follows:

(1) Euclidean Distance Multi-Objective Traveling Salesman Problem (Euclidean-MOTSP): This type of test problem aims to simulate multiple independent and conflicting dis-

tance costs. Given an instance $S = (s_1,\ldots,s_n)$ with n cities, the feature vector s_i of city i contains $2M$ coordinates. That is, each objective function corresponds to a set of independent two-dimensional Euclidean coordinates, and each coordinate value is uniformly and randomly sampled from the interval [0,1]. For this problem type, this paper conducted comparative experiments and algorithm testing on 200 randomly generated instances.

(2) Mixed Distance Multi-Objective Traveling Salesman Problem (Mixed-MOTSP): This type of instance aims to simulate more complex real-world scenarios (such as simultaneously optimizing route length and elevation smoothness). In this setting, a objectives, with $1 \leq a < M$, use independent two-dimensional coordinates, while the remaining $M - a$ objectives use one-dimensional coordinates. Therefore, the feature vector s_i of city i, $i = 1,\ldots,n$, contains $M + a$ coordinates, and all coordinate values are similarly sampled randomly from [0,1]. According to the common settings in reference [15], we set $a = 1$ in bi-objective problems and $a = 2$ in tri-objective problems; the test set also contains 200 randomly generated problem instances.

It should be emphasized that all learning-based algorithms in the experiments of this paper are trained exclusively on instances with $n = 40$ nodes. After training is complete, performance evaluation is conducted directly on random test sets with node counts $n \in \{40,70,100,200\}$.

Section 4.6 reports validation learning curves and target-attainment results. In each of the four 40-city scenarios, both baselines and all five training runs use the same 200 validation instances. Section 4.7 selects p using 200 validation instances per problem family and city count, shared across candidate thresholds and training seeds. All validation sets are disjoint from the corresponding final test sets. Section 4.7 presents validation results for threshold selection, while Section 4.3 reports final test results, which are not used to select p.

4.1.2. Evaluation Metrics

We evaluate the algorithms using hypervolume (HV) and inverted generational distance plus (IGD+). All objectives are minimized. Both indicators are calculated from the tour costs obtained after the input-coordinate preprocessing specified for each dataset. No additional normalization of the objective values is applied before metric calculation. In particular, the tour costs are not divided by the number of cities or normalized using ideal and nadir points.

(1). HV: The hypervolume (HV) metric is used to measure the volume of the space formed by the Pareto front set and a reference point, which can simultaneously reflect the convergence and diversity of the nondominated solutions. Let $\mathcal{P}$ be the set of objective vectors obtained by an algorithm, then HV is defined as follows:

$$\mathrm{HV}(\mathcal{P}) = \mathrm{VOL}\left(\bigcup_{u \in \mathcal{P}} [u_1, r_1^*] \times \cdots \times [u_M, r_M^*] \right) \tag{57}$$

where $\mathrm{VOL}(\cdot)$ denotes the M-dimensional Lebesgue measure, $\mathbf{u} = (u_1, \ldots, u_M)^\top$ is an objective vector obtained by the algorithm, and $\mathbf{r}^* = (r_1^*, \ldots, r_M^*)^\top$ is the reference objective vector. Generally speaking, a larger HV metric value indicates that the algorithm has a better approximation effect on the Pareto front. For an instance with n cities and M objectives, the HV reference point is the M-dimensional vector whose components are all equal to n. For example, the reference point is $(40, 40)$ for a bi-objective 40-city instance and $(200, 200, 200)$ for a tri-objective 200-city instance. All algorithms evaluated on the same instance use the same reference point. This HV reference point defines the reference corner for hypervolume calculation and is distinct from the ideal and utopian points introduced for scalarization

in Section 2.2. HV is calculated using the pymoo implementation on the tour-cost scale described above.

(2). IGD$^+$: The inverted generational distance plus (IGD$^+$) indicator measures the average modified distance from a reference set to the objective vectors obtained by an algorithm. Since the true Pareto fronts are unavailable for the evaluated instances, we use an empirical reference set constructed separately for each test instance. Let P denote the objective-vector set obtained by an algorithm and Z denote the empirical reference set for the same instance. IGD+ is defined as follows:

$$\mathrm{IGD}^+(\mathcal{P}) = \frac{1}{|\mathcal{Z}|}\sum_{\mathbf{z}\in\mathcal{Z}} \min_{\mathbf{u}\in\mathcal{P}} d^+(\mathbf{u},\mathbf{z}) \tag{58}$$

here, $|\mathcal{Z}|$ is the number of reference vectors, $\mathbf{u} = (u_1, \ldots, u_M)^\top$ is an objective vector obtained by the algorithm, and $\mathbf{z} = (z_1, \ldots, z_M)^\top$ is a reference objective vector. For minimization, the modified distance is

$$d^+(\mathbf{u},\mathbf{z}) = \sqrt{\sum_{m=1}^{M} [\max\{u_m - z_m, 0\}]^2}. \tag{59}$$

here, u_m and z_m are the components of the obtained and reference objective vectors, respectively; m indexes an objective, and M is the total number of objectives.

A smaller IGD+ value indicates a better approximation to the empirical reference set under the specified objective scale. It does not directly measure distance to a certified true Pareto front.

A separate empirical IGD+ reference set is constructed for each test instance by pooling the saved nondominated objective vectors from NSGA-II, MOEA/D, GRAWA, MOPN, DRL-MOA, MORAM-WS, and P-MOCO. For every learning-based method, the pool includes the outputs of all five independently trained models, using training seeds 12345–12349, evaluated on that same instance. On the randomly generated test datasets, each evolutionary solver contributes its single-run output for the instance. On each KroAB benchmark, NSGA-II and MOEA/D each contribute the outputs of all 30 runs using seeds 12345–12374. The five-model pooling rule for learning-based methods also applies to KroAB. After pooling across methods and runs for the same instance, we apply joint nondominated filtering under the minimization dominance relation, followed by duplicate handling using objective vectors quantized at a resolution of 10^{-6}. One common reference set is then used for every method and evaluated run on that instance. Objective vectors are pooled as set elements rather than averaged across runs, and outputs from different instances are never combined. The reference-set cardinality is determined by the retained vectors and is not fixed at 200; the number 200 refers to the random test instances within each configuration. Reference-set construction is distinct from aggregation of the reported metrics. Each evaluated run is scored separately against the common reference set. On random test instances, the five learning-model scores are averaged as described in Section 4.2. On KroAB, the learning-based score is calculated only for the fixed checkpoint selected by median mean HV on an independent validation set, which corresponds to training seed 12347 for each method; nevertheless, all five models contribute outputs to the reference pool. The evolutionary-solver scores are calculated separately for all 30 runs and then averaged. No metric is calculated on a union of multiple runs as a substitute for the reported single-run or selected-checkpoint output.

The reference sets are constructed separately for each test instance using pairwise dominance comparisons in a Python implementation. Let C_i denote the total number of objective vectors contributed by all included methods and runs for instance i before joint nondominated filtering, and let M denote the number of objectives. The nested comparisons have worst-case time complexity $O(MC_i^2)$. Skipping points already identified as dominated

can reduce the number of comparisons in practice but does not change this worst-case bound. If H_i vectors remain after filtering, quantization and sorting-based duplicate removal require $O(MH_i \log H_i)$ time for $H_i \geq 2$. Since $H_i \leq C_i$, the overall worst-case time remains $O(MC_i^2)$. Reference sets are constructed independently for different test instances, so the total processing time is the sum of the instance-level construction times.

We measured the offline reference-set construction time using the saved objective vectors. For each of the 16 random test configurations, one complete construction pass over its 200 instances was timed, excluding file reading. The total time ranged from 1.27 to 2.88 s per configuration, corresponding to 6.35–14.40 ms per instance when the total was divided by 200. These are single-pass measurements, not averages over repeated timing runs, and they do not quantify timing variability. Reference-set construction is performed offline after the timed algorithm evaluations and is excluded from the reported algorithm evaluation times.

The same reference-set construction procedure is used for all evaluated problem sizes, including the larger random instances and the KroAB benchmarks. IGD+ is calculated using the pymoo IGDPlus implementation without additional objective-value normalization. The contributing algorithms and their solution-generation budgets are described in Section 4.2. These reference sets are empirical approximations whose composition depends on the contributing algorithms, their computational budgets, and their retained solutions. Accordingly, the reported IGD+ results should be interpreted relative to these empirical sets rather than as comparisons against known true Pareto fronts.

4.2. Baseline Algorithms and Parameter Configurations

To comprehensively evaluate the performance of the GRAWA algorithm, this paper selected six representative multi-objective combinatorial optimization algorithms for comparison, including two classic heuristic algorithms and four advanced learning-based algorithms. All experiments were conducted on a workstation equipped with an AMD Ryzen 5 9600X CPU and an NVIDIA GeForce RTX 5060 Ti GPU with 16 GB of GPU memory. The software environment consisted of Python 3.12.13, PyTorch 2.12.0, CUDA 12.8, and pymoo 0.6.1.6. PyCharm 2026.3 was used as the development environment. Table 1 summarizes the architecture and graph-processing configurations of GRAWA.

Table 1. Architecture and graph-processing configurations of GRAWA.

Core Parameter	Value
Number of stacked encoders	3
Number of attention layer heads	8
Embedding layer dimension	128
Hidden layer dimension	128
Pooling ratio	0.5
Preference-graph neighborhood size	$k_w = 4$
Neighborhood symmetrization	Union
Base adjacency values	Binary
Number of pooling stages	1
Number of retained preference nodes	$\lceil 0.5N \rceil$
Pooling score gate	Sigmoid
Projection normalization constant	$\varepsilon = 10^{-8}$
Graph convolution activation	ReLU
Cross-attention heads	8
Cross-attention head dimension	16

Table 2 summarizes the parameter counts, model sizes, and mean training and evaluation times of the learning-based methods.

Table 2. Parameter counts, model sizes, and mean training and evaluation times. Training time is averaged over five independent runs with seeds 12345–12349. Total evaluation time is the mean of five evaluation times, one for each corresponding trained model, with each evaluation processing the same complete test set of 200 instances with 200 cities each. Times follow the algorithm-specific timing boundaries described in Section 4.2.

M	Type	Algorithm	#Parameters	Model Size	Training Time	Total Evaluation Time
2	Euclidean	DRL-MOA	18.461 M	70.85 MB	34.73 h	59.562 s
		MOPN	0.183 M	0.7 MB	6.21 h	53.289 s
		MORAM	1.307 M	15.04 MB	2.5 h	11.355 s
		P-MOCO	1.421 M	16.38 MB	0.29 h	81.339 s
		GRAWA	1.637 M	18.85 MB	3.38 h	13.619 s
2	Mixed	DRL-MOA	18.461 M	70.85 MB	37.82 h	58.957 s
		MOPN	0.183 M	0.7 MB	6.14 h	52.831 s
		MORAM	1.306 M	15.04 MB	2.0 h	11.312 s
		P-MOCO	1.421 M	16.38 MB	0.26 h	83.894 s
		GRAWA	1.636 M	18.85 MB	2.53 h	12.811 s
3	Euclidean	DRL-MOA	16.680 M	64.01 MB	29.47 h	62.390 s
		MOPN	0.184 M	0.7 MB	8.77 h	61.939 s
		MORAM	1.307 M	15.05 MB	2.88 h	35.966 s
		P-MOCO	1.505 M	17.34 MB	0.61 h	121.960 s
		GRAWA	1.637 M	18.86 MB	3.88 h	11.593 s
3	Mixed	DRL-MOA	16.657 M	63.92 MB	37.52 h	62.003 s
		MOPN	0.183 M	0.7 MB	8.21 h	61.516 s
		MORAM	1.307 M	15.05 MB	2.3 h	36.985 s
		P-MOCO	1.505 M	17.34 MB	0.64 h	126.455 s
		GRAWA	1.637 M	18.86 MB	2.91 h	11.946 s

The recorded evaluation times use algorithm-specific timing boundaries. For GRAWA, the measured interval includes greedy decoding and HV computation, with an evaluation batch size of 50. For MORAM, decoding time is recorded using the same evaluation batch size of 50 for both bi-objective and tri-objective problems. For P-MOCO, the measured interval includes multi-start decoding over all preference vectors, nondominated filtering, and HV computation. For MOPN, forward computation and tour-cost evaluation are included, whereas HV computation is excluded; its evaluation batch sizes are 128 and 64 for bi-objective and tri-objective problems, respectively. For DRL-MOA, the reported forward-computation time excludes subproblem-network loading. NSGA-II and MOEA/D are executed on the CPU, and their reported times correspond to the optimization runs over the evaluated instances. The offline pooling, joint nondominated filtering, and duplicate removal used to construct the common IGD+ reference sets are excluded from these timing intervals. Their separately measured construction cost is reported in Section 4.1.2.

These measurements describe computational cost under the reported implementations and evaluation settings. Differences in batching, candidate-generation budgets, and timed operations limit their interpretation as a comparison of end-to-end solving speed.

The compared heuristic algorithms include NSGA-II [7] and MOEA/D [8]. For two-objective problems, the population size for both algorithms (and the number of weight vectors in MOEA/D) is set to 101, while for three-objective problems, it is set to 91. The maximum number of generations is set to 1000, and the order crossover operator is used. MOEA/D adopts the Chebyshev method. Both algorithms use the implementations provided by pymoo version 0.6.1.6 [30].

The compared learning-based algorithms include DRL-MOA [18], MOPN [26], MORAM [25], and P-MOCO [24]. The number of subproblems for the learning-based models is set to 101 in bi-objective problems and 91 in tri-objective problems. Due to its

subproblem decomposition framework, DRL-MOA trains an independent sub-model for each subproblem for five epochs, with each epoch containing 10,000 instances. In contrast, MOPN, P-MOCO, MORAM, and the proposed GRAWA are all single-model methods, trained for 100 epochs with 10,000 instances per epoch. Both MORAM and GRAWA adopt the weighted-sum method for multi-objective scalarization decomposition. GRAWA was trained on 40-city instances for 100 epochs, with 10,000 training instances per epoch and a batch size of $B = 50$ problem instances. Each instance is evaluated under the N fixed preference vectors defined in Section 3.1.2. The embedding dimension is 128, the city encoder contains three layers, and attention uses eight heads with 16 channels per head. The initial learning rate is 5×10^{-4} and is multiplied by 0.99 after each epoch. Specifically,

$$\eta_e = 5 \times 10^{-4}(0.99)^e, \qquad e = 0,...,99 \tag{60}$$

where η_e is the learning rate used during epoch $e + 1$. The decoder uses a tanh logit-clipping coefficient of 10, which is distinct from gradient-norm clipping. Batch normalization is used. The softmin inverse-temperature parameter in the Top-p baseline is fixed at $\beta = 1.0$. The Top-p threshold is selected using the independent validation results in Section 4.7. We use $p = 0.5$ for bi-objective Euclidean problems, $p = 0.9$ for bi-objective mixed-distance problems, $p = 0.4$ for tri-objective Euclidean problems, and $p = 0.8$ for tri-objective mixed-distance problems. The candidate thresholds, validation-based selection procedure, and separation from the final test data are described in Section 4.7. Within each problem family, the same selected threshold is used across all five training seeds in the main GRAWA comparisons, the graph-routing and fusion ablations, the Top-p arm of the Top-p versus Top-k comparison, and both the original and leave-one-out Top-p variants in Appendix A.2. The threshold remains unchanged throughout each training run; "adaptive" refers to the number of selected candidates, not to an update of p. In the sensitivity study, each candidate p defines a separate training configuration and is likewise fixed throughout its run. Neither the final test results nor the KroAB results are used in the validation-based selection of p. Table 1 summarizes the architecture and graph-processing settings; the optimization settings are specified in this section. Appendix B summarizes the layer and tensor dimensions.

Model parameters are optimized using Adam with $(\beta_1,\beta_2) = (0.9,0.999)$, $\epsilon_{\text{Adam}} = 10^{-8}$, zero weight decay, and AMSGrad disabled. The global L_2 norm of the parameter gradients is clipped to 1.0 after backpropagation and before each optimizer step. One optimizer step is performed per batch, without gradient accumulation. With 10,000 instances and a batch size of 50, this gives 200 updates per epoch and 20,000 updates per training run. No learning-rate warm-up is used.

The trainable affine and graph-convolution weight matrices are initialized using Xavier uniform initialization with gain 1, and their biases, where present, are initialized to zero. Each component of the pooling projection vector is sampled from the uniform distribution $\mathcal{U}(-1/\sqrt{d}, 1/\sqrt{d})$. The vector is resampled if its norm is zero. Batch-normalization scale and shift parameters are initialized to one and zero, respectively. These initialization rules are applied independently for each training seed.

All baseline methods are executed using their official open-source implementations and evaluated within our unified experimental platform. In the main comparisons, each learning-based method—GRAWA, DRL-MOA, MOPN, MORAM, and P-MOCO—was trained independently five times under each training setting using seeds 12345, 12346, 12347, 12348, and 12349. The same set of five training-seed values was used for each ablation configuration, including the fully connected and degree-matched random-graph variants, each Top-p/Top-k configuration, each candidate threshold in the sensitivity analysis, and each original or leave-one-out Top-p configuration in Appendix A.2. For each preference set, the random-graph variant uses one fixed graph shared across the five independent training runs.

Thus, these repetitions vary the training seed rather than the random graph realization. For the test-set comparisons, all five runs and all compared methods were evaluated on the same 200 test instances within each configuration. The Top-p/Top-k learning curves and target-attainment analysis in Section 4.6 instead use 200 shared 40-city validation instances per scenario, separate from the final test instances. The candidate-threshold comparisons in Section 4.7 use 200 validation instances per configuration, shared across thresholds and training seeds and disjoint from the corresponding final test instances. For each learning-based method, HV and IGD+ were computed separately for each training run and test instance. The metric values were then averaged across the five training runs for each instance, and the resulting 200 instance-level averages were used to calculate the reported mean and standard deviation. Thus, the results of these comparisons on randomly generated instances are averaged over five independently trained models rather than taken from one selected training run. The KroAB evaluation in Section 4.8 instead uses one fixed checkpoint per learning-based method, selected by median mean HV on an independent validation set from the five trained models; the selected training seed is 12347 for every learning-based method. The descriptive standard deviation uses a divisor of 200 and characterizes dispersion across the instance-level averages; it is not a standard deviation across training seeds. On the randomly generated test datasets, NSGA-II and MOEA/D were each executed once per test instance. Their repeated stochastic evaluations on the fixed KroAB benchmarks are described separately in Section 4.8.

The empirical IGD+ reference sets pool saved nondominated objective vectors across both methods and runs, separately for each evaluated instance. The contributing methods are NSGA-II, MOEA/D, GRAWA, MOPN, DRL-MOA, MORAM-WS, and P-MOCO; MORAM-TCH is not included. Every learning-based method contributes the outputs of all five training seeds, 12345–12349. NSGA-II and MOEA/D each contribute one run per randomly generated test instance and all 30 runs with seeds 12345–12374 per KroAB instance. Joint filtering and duplicate handling then produce the single common reference set defined in Section 4.1.2.

For GRAWA, MORAM-WS, and MOPN, one greedy tour is generated for each of 101 preference vectors in bi-objective problems and 91 preference vectors in tri-objective problems. DRL-MOA generates one greedy tour from each corresponding subproblem network. P-MOCO uses multiple starting tours for each preference, with the number of starts set to $\min(40, n)$, and retains one tour per preference according to the Chebyshev criterion. Thus, its number of generated tours is larger than its number of retained tours. The nondominated objective vectors of the retained tours are used in reference-set construction. These output-generation and retention rules are applied separately to each included training run before its saved nondominated vectors are added to the common reference pool.

NSGA-II and MOEA/D return the nondominated objective vectors from their final populations after 1000 generations. On the randomly generated datasets, each algorithm is executed once on each of the 200 test instances per configuration. On each fixed KroAB benchmark, NSGA-II and MOEA/D are each executed independently 30 times, using every integer seed from 12345 through 12374 once per algorithm and instance. The per-run parameter settings and computational budgets remain unchanged. HV, IGD+, and elapsed solver time are recorded separately for each run. For NSGA-II and MOEA/D, the KroAB results in Section 4.8 report the arithmetic mean of each quantity across the 30 runs on the corresponding instance; Time/s is the mean duration of one solver execution, not the cumulative duration of all 30 executions. The empirical reference sets combine outputs obtained under algorithm-specific budgets, as described in the reference-set construction protocol.

Table 2 reports parameter counts, model sizes, mean training times, and mean total evaluation times for the learning-based methods. Training time is the arithmetic mean

of the elapsed training times from the five independent runs with seeds 12345–12349. Evaluation time is measured separately for the model obtained from each of these training runs. For each configuration, every model is evaluated on the same test set of 200 instances, each containing 200 cities, using the algorithm-specific evaluation batch size and timing boundaries described above. The reported total evaluation time is the arithmetic mean of these five complete-test-set evaluation times. It is neither the sum of the five times nor the average latency per instance. The timing repetition therefore spans the five independently trained models rather than five repeated evaluations of a single selected checkpoint.

4.3. Comparative Experimental Analysis

We compare HV and IGD+ across all 16 test configurations. Figure 7 presents the Pareto front approximations for the four 200-city configurations, covering bi-objective and tri-objective problems with Euclidean and mixed-distance objectives. To keep the main figure readable, the corresponding plots for 40, 70, and 100 cities are provided in Figure A2 in Appendix E. Each of the 16 test configurations contains 200 shared test instances. For every learning-based method, each instance-level metric is the average obtained from five independently trained models. For NSGA-II and MOEA/D, the metric from one solver execution on each instance is used. The reported mean and parenthesized standard deviation summarize these 200 instance-level values, with a divisor of 200 for the descriptive standard deviation. For the learning-based methods, the standard deviations therefore describe dispersion across instances after averaging over five training runs; they do not quantify variability across the five training seeds. For NSGA-II and MOEA/D, the observed dispersion across instances also includes the effects of stochastic solver execution. Boldface identifies the best mean value within each configuration.

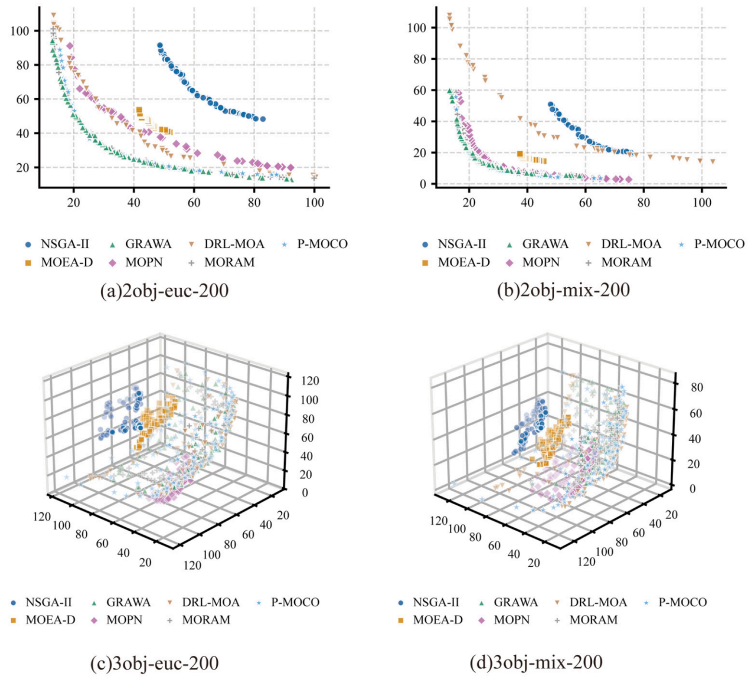


Figure 7. Pareto front approximations obtained by the compared algorithms for 200-city MOTSP instances: (**a**) bi-objective Euclidean; (**b**) bi-objective mixed-distance; (**c**) tri-objective Euclidean; and (**d**) tri-objective mixed-distance. Points show objective vectors of generated tours, with all objectives minimized. Corresponding plots for 40, 70, and 100 cities are provided in Figure A2 in Appendix E.

For each metric and configuration, GRAWA is compared with each competing algorithm using a two-sided paired Wilcoxon signed-rank test at $\alpha = 0.05$. Each pair contains

the two algorithms' metric values on the same test instance. For comparisons between learning-based methods, both values are averages over five independent training runs. For comparisons with NSGA-II or MOEA/D, GRAWA's five-run average is paired with the corresponding solver's single-run value on that same instance. Thus, the test uses 200 instance-level pairs before the handling of zero differences. The five training runs are aggregated within each instance and are not counted as additional independent observations. Preference vectors and individual tours are likewise not treated as independent observations. The symbols are defined relative to GRAWA: "+" indicates a significant difference in favor of GRAWA, "−" indicates a significant difference against GRAWA, and "≈" indicates that no significant difference was detected (p-value ≥ 0.05). Larger HV and smaller IGD+ indicate better performance. A nonsignificant result is not interpreted as evidence of equivalence. These are instance-level comparisons under the evaluated training and solver protocol, not tests of differences across training seeds. No correction for multiple comparisons was applied. The significance symbols therefore refer to unadjusted, two-sided p-values at a nominal significance level of 0.05. These results are interpreted as individual pairwise comparisons and do not provide family-wise error control across algorithms, configurations, or metrics. Table 3 presents the HV results for all 16 test configurations.

HV Result Comparison: As shown in Table 3, GRAWA attains the best mean HV in eight of the 16 test configurations, covering all Euclidean cases for both the bi-objective and tri-objective settings. Across the 16 configurations, GRAWA has the lowest average rank for mean HV at 1.75, followed by P-MOCO (2.06), MORAM (3.31), DRL-MOA (3.50), MOPN (5.19), MOEA/D (5.56), and NSGA-II (6.62). These ranks provide a descriptive summary of the configuration-level ordering and are not presented as a Friedman omnibus test result. Under the unadjusted two-sided Wilcoxon signed-rank tests at a nominal significance level of 0.05, GRAWA has significantly higher HV than NSGA-II, MOEA/D, and MOPN in all 16 configurations and than MORAM in 15 configurations. In the remaining comparison with MORAM, no significant difference is detected; this does not establish equivalence between the methods.

The mixed-distance results differ between the bi-objective and tri-objective settings. P-MOCO obtains the best mean HV in six of the eight mixed-distance configurations and significantly higher HV than GRAWA in all eight. Conversely, GRAWA obtains significantly higher HV than P-MOCO in all eight Euclidean configurations under the reported unadjusted Wilcoxon tests. DRL-MOA achieves the best mean HV on the two largest tri-objective mixed configurations, and the overall comparison with GRAWA is 12/4/0, with the four losses all occurring in the tri-objective mixed group. DRL-MOA uses 91 separately trained sub-models for the 91 predefined preferences in the tri-objective setting, whereas GRAWA uses a single shared model. This difference provides architectural context for the comparison, but the present experiments do not isolate whether separate sub-models or differences in training resources explain DRL-MOA's higher mean HV in these configurations.

GRAWA generates a Pareto front approximation using one set of network parameters. Under the reported protocol, it has higher mean HV than DRL-MOA in 12 of the 16 configurations and lower mean HV in the four tri-objective mixed-distance configurations. Its average HV rank is the lowest among the compared methods. These results characterize the performance of the complete model under the evaluated settings; they do not establish robustness across training seeds or identify the architectural mechanism responsible for the observed differences. Training-resource measurements are reported separately in Table 2.

Table 3. Mean HV and standard deviation across 200 test instances for each of the 16 MOTSP configurations. For each learning-based method, the instance-level HV is first averaged over five independent training runs. NSGA-II and MOEA/D are each executed once per instance. Parenthesized values describe between-instance dispersion, not variability across training seeds. Significance symbols are based on unadjusted two-sided paired Wilcoxon signed-rank tests at $\alpha = 0.05$. Average rank is a descriptive summary across the 16 configurations; lower values indicate better ranks.

Type	NSGA-II	MOEA/D	DRL-MOA	MOPN	MORAM	P-MOCO	GRAWA
2obj-euc-40	1.099e+03(1.877e+01)+	1.102e+03(1.860e+01)+	1.118e+03(1.688e+01)+	1.031e+03(1.814e+01)+	1.152e+03(1.520e+01)+	1.147e+03(1.465e+01)+	**1.154e+03(1.501e+01)**
2obj-euc-70	3.255e+03(5.059e+01)+	3.390e+03(5.002e+01)+	3.690e+03(3.248e+01)+	3.478e+03(3.497e+01)+	3.813e+03(2.864e+01)+	3.770e+03(2.717e+01)+	**3.822e+03(2.760e+01)**
2obj-euc-100	6.329e+03(9.761e+01)+	6.796e+03(1.018e+02)+	7.789e+03(5.269e+01)+	7.420e+03(5.879e+01)+	8.056e+03(4.555e+01)+	7.947e+03(4.436e+01)+	**8.076e+03(4.414e+01)**
2obj-euc-200	2.253e+04(3.487e+02)+	2.554e+04(3.780e+02)+	3.277e+04(1.114e+02)+	3.150e+04(1.364e+02)+	3.385e+04(8.854e+01)+	3.333e+04(9.446e+01)+	**3.392e+04(8.917e+01)**
2obj-mix-40	1.264e+03(1.548e+01)+	1.268e+03(1.502e+01)+	1.120e+03(1.688e+01)+	1.259e+03(1.441e+01)+	1.280e+03(1.364e+01)≈	**1.295e+03(1.219e+01)−**	1.281e+03(1.342e+01)
2obj-mix-70	3.797e+03(4.398e+01)+	3.931e+03(3.835e+01)+	3.694e+03(3.166e+01)+	4.102e+03(2.806e+01)+	4.136e+03(2.412e+01)+	**4.184e+03(1.973e+01)−**	4.147e+03(2.471e+01)
2obj-mix-100	7.455e+03(8.927e+01)+	7.933e+03(7.458e+01)+	7.799e+03(5.180e+01)+	8.599e+03(4.275e+01)+	8.639e+03(4.191e+01)+	**8.739e+03(3.118e+01)−**	8.674e+03(3.727e+01)
2obj-mix-200	2.706e+04(3.306e+02)+	3.034e+04(3.270e+02)+	3.280e+04(1.145e+02)+	3.575e+04(8.666e+01)+	3.572e+04(8.387e+01)+	**3.607e+04(6.856e+01)−**	3.589e+04(8.521e+01)
3obj-euc-40	2.862e+04(1.101e+03)+	2.976e+04(9.374e+02)+	3.560e+04(7.661e+02)+	2.806e+04(1.382e+03)+	3.467e+04(1.158e+03)+	3.596e+04(7.362e+02)+	**3.624e+04(7.430e+02)**
3obj-euc-70	1.394e+05(4.995e+03)+	1.491e+05(4.791e+03)+	2.150e+05(2.856e+03)+	1.608e+05(6.713e+03)+	2.080e+05(6.527e+03)+	2.144e+05(2.836e+03)+	**2.181e+05(2.917e+03)**
3obj-euc-100	3.720e+05(1.210e+04)+	4.070e+05(1.136e+04)+	6.659e+05(5.440e+03)+	4.853e+05(1.551e+04)+	6.405e+05(1.724e+04)+	6.584e+05(5.203e+03)+	**6.726e+05(5.392e+03)**
3obj-euc-200	2.452e+06(6.085e+04)+	2.763e+06(7.411e+04)+	5.828e+06(2.697e+04)+	4.102e+06(9.844e+04)+	5.578e+06(1.351e+05)+	5.706e+06(2.659e+04)+	**5.831e+06(2.857e+04)**
3obj-mix-40	3.470e+04(1.160e+03)+	3.569e+04(9.735e+02)+	4.111e+04(7.185e+02)−	3.224e+04(1.495e+03)+	3.965e+04(8.088e+02)+	**4.157e+04(7.120e+02)−**	4.024e+04(7.935e+02)
3obj-mix-70	1.706e+05(5.175e+03)+	1.807e+05(4.641e+03)+	2.433e+05(2.471e+03)−	1.879e+05(6.219e+03)+	2.336e+05(2.741e+03)+	**2.439e+05(2.257e+03)−**	2.373e+05(2.674e+03)
3obj-mix-100	4.601e+05(1.303e+04)+	4.939e+05(1.169e+04)+	**7.439e+05(5.216e+03)−**	5.652e+05(1.491e+04)+	7.121e+05(5.757e+03)+	7.422e+05(4.795e+03)−	7.234e+05(5.774e+03)
3obj-mix-200	3.093e+06(7.274e+04)+	3.426e+06(8.094e+04)+	**6.386e+06(2.596e+04)−**	4.766e+06(1.002e+05)+	6.092e+06(2.817e+04)+	6.336e+06(2.221e+04)−	6.173e+06(2.698e+04)
+/−/≈	16/0/0	16/0/0	12/4/0	16/0/0	15/0/1	8/8/0	\
Average rank	6.62	5.56	3.50	5.19	3.31	2.06	1.75

For each metric, the seven algorithms are ranked within each configuration using their reported mean performance, with rank 1 assigned to the best value: higher HV or lower IGD+. The average rank is the arithmetic mean of these ranks across the 16 configurations. This is a descriptive summary, separate from the instance-level Wilcoxon comparisons. The reported average ranks are not used as evidence of an omnibus significance test or of significant differences between average ranks. Table 4 presents the corresponding IGD+ results for all 16 test configurations.

IGD+ Result Comparison: Table 4 reports the mean IGD+ values, for which lower is better. GRAWA attains the best mean IGD+ in seven of the 16 configurations, namely the three larger bi-objective Euclidean cases (n = 70,100,200) and all four tri-objective Euclidean cases. P-MOCO obtains the lowest mean IGD+ in eight configurations, while MOEA/D obtains the lowest value in one configuration, 2obj-mix-40. For mean IGD+, the average ranks across the 16 configurations are 1.81 for P-MOCO, 2.00 for GRAWA, 3.25 for MORAM, 4.00 for DRL-MOA, 5.00 for MOEA/D, 5.38 for MOPN, and 6.56 for NSGA-II, with a lower rank indicating better performance. These values summarize the observed ordering and do not by themselves establish statistically significant differences between algorithms. GRAWA and P-MOCO have the two lowest average ranks for both metrics. Their relative performance across Euclidean and mixed-distance configurations is discussed below.

GRAWA has the lowest mean IGD+ in seven of the eight Euclidean configurations; the exception is 2obj-euc-40. For example, on 3obj-euc-200, its mean IGD+ is 2.263, compared with 2.436 for DRL-MOA and 3.255 for P-MOCO. In the mixed-distance configurations, P-MOCO has lower mean IGD+ than GRAWA in all eight cases. It has the lowest mean IGD+ among all compared methods in seven of these configurations, while MOEA/D has the lowest value on 2obj-mix-40. The IGD+ results show different performance patterns across the Euclidean and mixed-distance configurations. Under the stated empirical reference-set protocol, P-MOCO has lower mean IGD+ in the mixed-distance configurations, while GRAWA performs better in most Euclidean configurations. These aggregate comparisons do not isolate the contribution of preference-conditioned sampling or graph-based guidance, nor do they establish uniform local coverage of the reference set.

Under the unadjusted two-sided Wilcoxon signed-rank tests at a nominal significance level of 0.05, GRAWA has significantly lower IGD+ than NSGA-II and MOPN in all 16 configurations. Against MOEA/D, GRAWA has significantly lower IGD+ in 15 configurations and significantly higher IGD+ in one; against MORAM, the corresponding counts are 14 and one, with no significant difference detected in the remaining configuration. Against DRL-MOA, GRAWA has significantly lower IGD+ in 12 configurations and significantly higher IGD+ in the four tri-objective mixed-distance configurations. DRL-MOA uses 91 separately trained sub-models in the tri-objective setting, whereas GRAWA uses one shared model. Across the 16 configurations, GRAWA has the second-lowest average rank for mean IGD+ and the lowest average rank for mean HV. These comparisons evaluate the complete methods under their stated settings. They do not isolate the effects of the routing and fusion modules or establish which mechanism explains the performance differences. The corresponding module-level ablation comparisons are reported in Sections 4.4 and 4.5.

Table 4. Mean IGD+ and standard deviation across 200 test instances for each of the 16 MOTSP configurations. For each learning-based method, the instance-level IGD+ is first averaged over five independent training runs. NSGA-II and MOEA/D are each executed once per instance. Parenthesized values describe between-instance dispersion, not variability across training seeds. Significance symbols are based on unadjusted two-sided paired Wilcoxon signed-rank tests at $\alpha = 0.05$. Average rank is a descriptive summary across the 16 configurations; lower values indicate better ranks.

Type	NSGA-II	MOEA/D	DRL-MOA	MOPN	MORAM	P-MOCO	GRAWA
2obj-euc-40	9.244e-01(1.750e-01)+	3.399e-01(1.053e-01)+	1.488e+00(2.305e-01)+	2.816e+00(3.925e-01)+	2.900e-01(6.581e-02)+	**1.864e-01(5.329e-02)−**	2.364e-01(6.096e-02)
2obj-euc-70	4.922e+00(4.418e-01)+	2.411e+00(3.139e-01)+	2.522e+00(2.437e-01)+	4.059e+00(4.120e-01)+	2.546e-01(6.277e-02)+	3.062e-01(5.856e-02)+	**1.562e-01(4.250e-02)**
2obj-euc-100	1.097e+01(7.058e-01)+	5.611e+00(5.213e-01)+	3.770e+00(3.233e-01)+	5.397e+00(4.839e-01)+	3.092e-01(7.169e-02)+	4.529e-01(7.142e-02)+	**1.672e-01(4.687e-02)**
2obj-euc-200	3.772e+01(1.570e+00)+	2.102e+01(1.328e+00)+	8.030e+00(4.830e-01)+	1.061e+01(7.669e-01)+	5.055e-01(1.014e-01)+	7.745e-01(1.026e-01)+	**2.765e-01(6.465e-02)**
2obj-mix-40	6.669e-01(1.673e-01)+	**1.484e-01(5.786e-02)−**	5.517e+00(7.314e-01)+	9.364e-01(1.579e-01)+	3.281e-01(7.482e-02)−	1.625e-01(5.286e-02)−	3.687e-01(8.429e-02)
2obj-mix-70	3.804e+00(4.586e-01)+	1.208e+00(2.984e-01)+	8.705e+00(8.260e-01)+	1.188e+00(1.727e-01)+	4.117e-01(9.514e-02)≈	**9.699e-02(4.326e-02)−**	4.125e-01(8.522e-02)
2obj-mix-100	8.817e+00(7.213e-01)+	3.400e+00(4.288e-01)+	1.193e+01(9.207e-01)+	1.510e+00(1.845e-01)+	5.774e-01(1.092e-01)+	**9.398e-02(3.546e-02)−**	5.381e-01(9.732e-02)
2obj-mix-200	3.183e+01(1.558e+00)+	1.471e+01(1.173e+00)+	2.249e+01(1.179e+00)+	2.314e+00(1.999e-01)+	1.230e+00(1.374e-01)+	**1.400e-01(4.348e-02)−**	1.106e+00(1.390e-01)
3obj-euc-40	4.228e+00(4.349e-01)+	2.054e+00(2.765e-01)+	8.962e-01(1.131e-01)+	4.300e+00(6.028e-01)+	1.596e+00(4.856e-01)+	7.587e-01(9.387e-02)+	**6.968e-01(9.762e-02)**
3obj-euc-70	1.207e+01(7.852e-01)+	7.181e+00(5.195e-01)+	1.273e+00(1.244e-01)+	8.412e+00(9.425e-01)+	2.528e+00(8.695e-01)+	1.257e+00(1.257e-01)+	**9.959e-01(1.201e-01)**
3obj-euc-100	2.191e+01(1.101e+00)+	1.378e+01(7.994e-01)+	1.574e+00(1.272e-01)+	1.276e+01(1.154e+00)+	3.610e+00(1.216e+00)+	1.753e+00(1.310e-01)+	**1.286e+00(1.244e-01)**
3obj-euc-200	6.129e+01(1.719e+00)+	4.300e+01(1.662e+00)+	2.436e+00(1.444e-01)+	2.806e+01(2.040e+00)+	6.945e+00(2.660e+00)+	3.255e+00(1.442e-01)+	**2.263e+00(1.516e-01)**
3obj-mix-40	3.372e+00(3.765e-01)+	1.289e+00(2.138e-01)+	4.929e-01(5.992e-02)−	3.287e+00(6.142e-01)+	1.166e+00(1.894e-01)+	**2.769e-01(3.747e-02)−**	8.727e-01(1.477e-01)
3obj-mix-70	1.019e+01(7.762e-01)+	5.809e+00(4.958e-01)+	6.868e-01(7.021e-02)−	6.346e+00(7.965e-01)+	1.944e+00(2.157e-01)+	**4.374e-01(4.125e-02)−**	1.381e+00(1.648e-01)
3obj-mix-100	1.870e+01(1.072e+00)+	1.179e+01(7.309e-01)+	8.901e-01(7.312e-02)−	9.953e+00(1.030e+00)+	2.801e+00(2.525e-01)+	**6.347e-01(4.832e-02)−**	1.953e+00(1.966e-01)
3obj-mix-200	5.409e+01(1.750e+00)+	3.767e+01(1.659e+00)+	1.550e+00(1.017e-01)−	2.301e+01(1.918e+00)+	5.867e+00(3.517e-01)+	**1.251e+00(7.636e-02)−**	4.197e+00(2.806e-01)
+/−/≈	16/0/0	15/1/0	12/4/0	16/0/0	14/1/1	7/9/0	\
Average rank	6.56	5.00	4.00	5.38	3.25	1.81	2.00

4.4. Effectiveness Analysis of the Graph Weight Routing Module

To distinguish the contribution of graph weight routing from that of the preference-neighborhood structure, we compare four variants in Table 5: the original preference graph, a variant without graph weight routing, a fully connected graph, and a degree-matched random graph. The original preference graph uses the four-nearest-neighbor selection and union-based symmetrization defined in Section 3.1.2. The variant without graph weight routing removes the routing module and supplies preference vectors independently to the subsequent modules. The two alternative graph variants retain graph processing, with the remaining model and training settings unchanged.

Table 5. HV comparison of the preference graph, the variant without graph weight routing, the fully connected graph, and the degree-matched random graph. Each variant is trained independently five times and evaluated on the same 200 test instances per configuration. For each instance, HV is first averaged across the five training runs; the reported mean and parenthesized standard deviation are then calculated across the resulting 200 instance-level averages. The standard deviation uses a divisor of 200 and describes between-instance dispersion, not variability across training seeds or random graph realizations. For each preference set, the random-graph variant uses one fixed graph shared by all five training runs. Boldface indicates the highest reported mean HV in each row.

M	*n*	Type	Preference Graph	Without Graph Routing	Fully Connected Graph	Degree-Matched Random Graph
2	40	Euc	**1.154e+3(1.501e+1)**	1.151e+3(1.491e+1)	1.152e+3(1.513e+1)	1.151e+3(1.518e+1)
	70		**3.822e+3(2.760e+1)**	3.812e+3(2.739e+1)	3.813e+3(2.783e+1)	3.810e+3(2.793e+1)
	100		**8.076e+3(4.414e+1)**	8.055e+3(4.548e+1)	8.056e+3(4.481e+1)	8.050e+3(4.487e+1)
	200		**3.392e+4(8.917e+1)**	3.380e+4(9.225e+1)	3.384e+4(9.424e+1)	3.382e+4(9.467e+1)
2	40	Mix	**1.281e+3(1.342e+1)**	1.255e+3(1.788e+1)	1.266e+3(1.643e+1)	1.273e+3(1.477e+1)
	70		**4.147e+3(2.471e+1)**	4.059e+3(3.585e+1)	4.100e+3(2.947e+1)	4.123e+3(2.597e+1)
	100		**8.674e+3(3.727e+1)**	8.493e+3(5.426e+1)	8.573e+3(4.779e+1)	8.625e+3(4.073e+1)
	200		**3.589e+4(8.521e+1)**	3.519e+4(1.198e+2)	3.547e+4(9.685e+1)	3.567e+4(9.522e+1)
3	40	Euc	**3.624e+4(7.430e+2)**	2.613e+4(7.399e+2)	3.609e+4(7.404e+2)	3.612e+4(7.271e+2)
	70		**2.181e+5(2.917e+3)**	1.178e+5(2.970e+3)	2.174e+5(2.951e+3)	2.173e+5(2.927e+3)
	100		**6.726e+5(5.392e+3)**	5.720e+5(5.510e+3)	6.695e+5(5.423e+3)	6.699e+5(5.372e+3)
	200		**5.831e+6(2.857e+4)**	4.838e+6(2.826e+4)	5.806e+6(2.726e+4)	5.809e+6(2.783e+4)
3	40	Mix	**4.024e+4(7.935e+2)**	3.340e+4(7.892e+2)	4.023e+4(7.791e+2)	4.003e+4(7.711e+2)
	70		**2.373e+5(2.674e+3)**	1.380e+5(2.703e+3)	2.359e+5(2.681e+3)	2.370e+5(2.589e+3)
	100		**7.234e+5(5.774e+3)**	6.252e+5(6.138e+3)	7.109e+5(5.851e+3)	7.209e+5(5.953e+3)
	200		**6.173e+6(2.698e+4)**	5.194e+6(2.797e+4)	6.147e+6(2.911e+4)	6.145e+6(2.710e+4)

The fully connected base graph connects every pair of distinct preference nodes. The random base graph preserves the degree of every node in the original symmetrized preference graph and therefore preserves the total number of edges. Preference vectors remain assigned to their original node indices. For each preference set, one random graph is generated and shared across all five independent training runs; it is not resampled during training. The alternative base graphs are undirected and binary, with self-loops added separately during graph convolution and degree normalization calculated from the graph used by each variant. The base adjacency remains fixed within each variant, while learned node scores and the selected pooled subgraph may change during training.

For each graph-based variant, the alternative base adjacency is used in both the graph weight routing module and the additional preference-graph convolution preceding fusion in Section 3.2. The pooled adjacency is extracted from the corresponding alternative base graph using the selected node indices. Self-loops and degree normalization are recomputed at each graph resolution. The preference-node inputs, graph-convolution depth, pooling ratio, and fusion architecture are otherwise unchanged.

Each variant is trained independently five times and evaluated on the same 200 test instances per configuration. For each instance, HV is first averaged over the five training runs. The mean and standard deviation in Table 5 are then calculated across the resulting

200 instance-level averages, following Section 4.2. The comparison with no graph weight routing assesses the contribution of the complete module; the alternative graph comparisons examine the reported performance under different connections between preference nodes.

Table 5 reports the highest mean HV for the original preference graph in all 16 evaluated configurations, compared with the variant without graph weight routing, the fully connected graph, and the degree-matched random graph. Based on the rounded means displayed in the table, the preference graph's relative HV advantage is approximately 0.025–1.758% over the fully connected graph and 0.127–0.628% over the degree-matched random graph. These are descriptive differences in reported means; they are not presented as evidence that every difference is statistically significant.

The additional graph comparisons provide evidence beyond removal of the complete routing module. The degree-matched random graph preserves each node's degree and the total edge count, so these quantities do not explain the difference between that control and the preference graph. The observed results support the usefulness of preference-based connectivity relative to the evaluated random realization. The fully connected comparison shows that dense connectivity does not improve mean HV in these experiments, although it changes both graph density and locality and therefore does not isolate locality alone.

The gains over alternative graphs are smaller than the gains over removing graph weight routing in several configurations, particularly in the tri-objective settings. The benefit of the complete module should therefore not be attributed entirely to the neighborhood structure. In addition, the five random-graph runs share one fixed graph realization for each preference set, so the experiment does not quantify variability across independently sampled random graphs. These results do not establish global optimality of the proposed topology, isolate the individual contributions of propagation, pooling, and restoration, or prove proximity of optimal tours or Pareto-front points for neighboring preferences. Appendix D provides a separate cost-based sensitivity argument for information sharing between nearby preferences.

4.5. Effectiveness Analysis of the Graph Co-Guided Fusion Module

The GRAWA model introduces a graph co-guided fusion module to optimize problem embeddings. To verify the effectiveness of this module, this paper constructed an ablation experiment, where the baseline embedding was generated using the traditional weighted aggregation method. Table 6 compares the two embedding strategies using mean HV and the between-instance standard deviation in parentheses. As in Table 5, these statistics summarize the 200 instance-level HV values after averaging over five independent training runs.

Table 6. HV comparison between graph co-guided fusion and weighted aggregation. Entries report mean HV (standard deviation), calculated across 200 instance-level averages after averaging each instance's HV over five independent training runs. The same test instances are used for both variants. Parenthesized values use a divisor of 200 and describe between-instance dispersion, not between-training-run variability.

M	*n*	Type	Graph Fusion	Weighted Aggregation
2	40	Euc	**1.154e+3(1.501e+1)**	1.151e+3(1.520e+1)
	70		**3.822e+3(2.760e+1)**	3.814e+3(2.862e+1)
	100		**8.076e+3(4.414e+1)**	8.056e+3(4.490e+1)
	200		**3.392e+4(8.917e+1)**	3.383e+4(9.576e+1)
2	40	Mix	**1.281e+3(1.342e+1)**	1.254e+3(1.414e+1)
	70		**4.147e+3(2.471e+1)**	4.073e+3(3.057e+1)
	100		**8.674e+3(3.727e+1)**	8.514e+3(4.597e+1)
	200		**3.589e+4(8.521e+1)**	3.527e+4(1.024e+2)

Table 6. *Cont.*

M	*n*	Type	Graph Fusion	Weighted Aggregation
3	40	Euc	**3.624e+4(7.430e+2)**	2.973e+4(1.453e+3)
	70		**2.181e+5(2.917e+3)**	1.628e+5(5.929e+3)
	100		**6.726e+5(5.392e+3)**	4.845e+5(1.425e+4)
	200		**5.831e+6(2.857e+4)**	4.007e+6(9.181e+4)
3	40	Mix	**4.024e+4(7.935e+2)**	3.931e+4(8.404e+2)
	70		**2.373e+5(2.674e+3)**	2.306e+5(2.725e+3)
	100		**7.234e+5(5.774e+3)**	7.031e+5(6.515e+3)
	200		**6.173e+6(2.698e+4)**	6.009e+6(3.149e+4)

The graph co-guided fusion module achieves higher mean HV than weighted aggregation in the evaluated configurations. These results are averaged over five independent training runs. The standard deviations in Table 6 describe dispersion across the instance-level averages and are generally smaller for graph co-guided fusion. This observation concerns differences across the evaluated instances; it does not establish reduced variability across training seeds or directly demonstrate a noise-filtering mechanism. We therefore interpret the results as evidence concerning the observed mean solution quality under the reported evaluation protocol.

4.6. Effectiveness Analysis of Top-p Baseline

This section compares the empirical learning behavior of the Top-p and Top-k baselines in bi-objective and tri-objective Euclidean and mixed-distance settings. Top-p selects a variable number of candidates using a fixed cumulative probability threshold, whereas Top-k retains a fixed number. Following Wang et al. [25], the Top-k candidate count is obtained by dividing 20% of the number of subproblems by the number of objectives and rounding down. With 101 bi-objective and 91 tri-objective subproblems, this gives 10 and six retained candidates, respectively.

Figure 8 reports validation learning curves from five independent training runs for each method and scenario. All four scenarios use 40-city validation instances. Within each scenario, the same 200 validation instances are used for both methods and all five training runs, and these instances are separate from the final test instances. At each epoch, HV is first averaged over the five runs for each validation instance. The solid curve is the mean of the resulting 200 instance-level averages, and the shaded region spans one standard deviation above and below this mean across validation instances. The shading therefore describes between-instance dispersion after averaging over training runs; it is neither a between-seed standard deviation nor a confidence interval.

The vertical axis reports validation HV on its original scale, without percentage conversion. HV measures the quality of the solution set produced by each evaluated checkpoint and is distinct from the scalarized tour cost used in the policy update and its equivalent negative terminal-cost return. The numerical scale differs across scenarios because HV measures area for two objectives and volume for three objectives. The reference points are $(40, 40)$ and $(40, 40, 40)$, respectively. Differences in objective-cost distributions between Euclidean and mixed-distance problems also affect the scale. Raw HV values across the four panels should therefore not be interpreted as directly comparable performance scores.

To supplement the curves with a quantitative description of learning progress, we report the epoch at which a common validation HV target is attained. Within each scenario, the target is defined post hoc as 99% of the larger of the two methods' final-epoch mean validation HV values, averaged over five training seeds. For each seed, attainment is the first logged epoch starting a sequence of five consecutive epochs whose instance-averaged validation HV values all meet or exceed that target. The reported attainment epoch is the

arithmetic mean of the five seed-specific values. Epoch indices follow the original logs, from 0 to 99, with epoch 99 used to define the target. This is a descriptive comparison under a common target, not a training stopping rule, proof of convergence, or measurement of gradient variance.

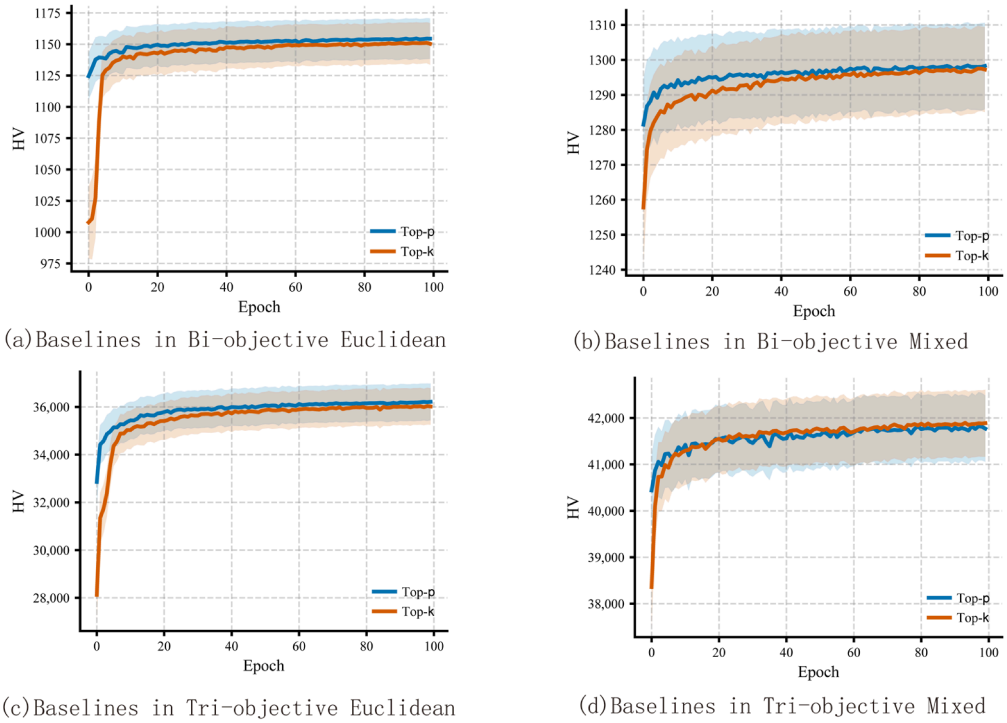


Figure 8. Validation learning curves of Top-p and Top-k on four 40-city MOTSP scenarios, using five independent training runs per method and scenario. The vertical axis reports validation HV on its original scale. Within each scenario, both methods and all training runs use the same 200 validation instances, separate from the final test instances. Solid curves show the mean after averaging HV over the five runs for each validation instance. Shaded regions indicate one standard deviation above and below the mean across these instance-level averages; they are neither between-seed standard deviations nor confidence intervals. Raw HV scales differ across objective counts and distance settings.

Under this criterion, Top-p reaches the target at mean logged epochs of 5.6, 1.2, and 23.0 in the bi-objective Euclidean, bi-objective mixed-distance, and tri-objective Euclidean settings, respectively, compared with 20.2, 7.6, and 55.4 for Top-k. In the tri-objective mixed-distance setting, Top-k reaches the target earlier on average, at epoch 22.2 compared with 28.6 for Top-p. These results describe target-attainment progress under the specified validation protocol and do not establish uniformly faster progress for either baseline.

The two baselines differ in how they select reference candidates, but the observed target-attainment differences do not isolate the mechanism responsible for their learning behavior. Policy-gradient estimation variance was not directly measured, and neither the curves nor their shaded regions establish improved stability across training seeds or reduced gradient variance. Moreover, the original Top-p baseline depends on the current sampled tour and may change the expected update. Appendix A.1 analyzes this potential bias, while Appendix A.2 provides a separate comparison with a leave-one-out variant. We therefore restrict the conclusions of this section to the observed validation learning behavior under the reported settings.

4.7. Validation-Based Selection and Sensitivity Analysis of Hyperparameter p

The threshold p is selected offline using validation data. It specifies the cumulative candidate-probability mass retained in Algorithm 2 and remains fixed throughout each training run. We compare the candidate set $\mathcal{P} = \{0.1,0.2,\ldots,0.9\}$. Each candidate is used to train a separate configuration with five independent seeds on 40-city training instances. The resulting models are evaluated on validation instances with $n \in \{40,70,100,200\}$ cities. For each problem family and city count, the same 200 validation instances are used across all candidate thresholds and training seeds. These validation instances are separate from the final test instances. Figure 9 reports validation performance used for p selection; it does not depict an online update of p or use the final test results for selection.

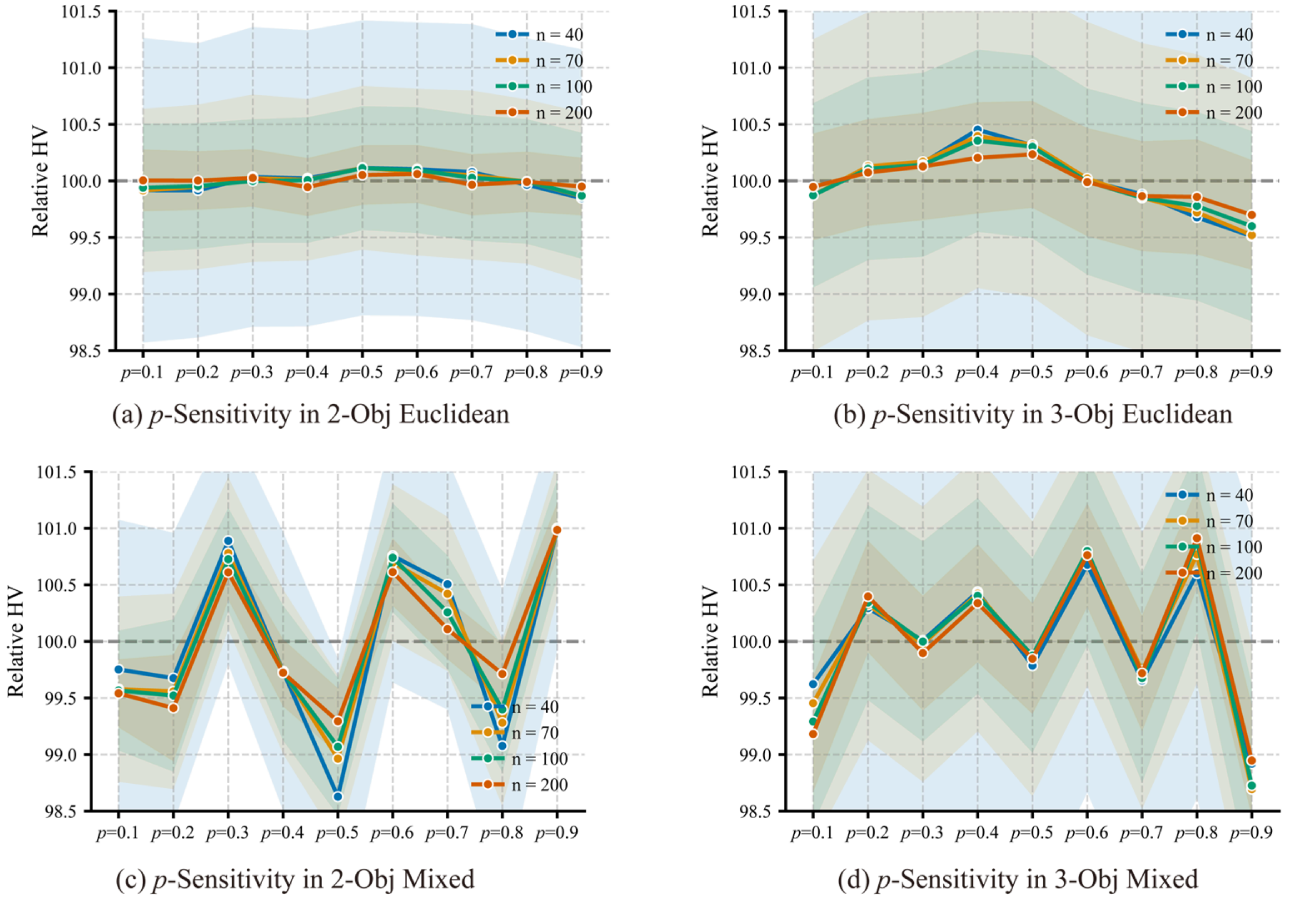


Figure 9. Validation-based sensitivity analysis of the fixed Top-p threshold. For each problem family and city count, Relative HV is the mean validation HV at each threshold divided by the average of the mean validation HV values over the nine candidate thresholds, expressed as a percentage. Each threshold setting uses five independent training runs evaluated on the same 200 validation instances, which are separate from the final test instances. Solid curves show normalized mean HV, obtained by first averaging over training seeds for each validation instance and then over validation instances. Shaded regions indicate ± one between-instance standard deviation of the seed-averaged validation HV values, scaled by the same denominator. These validation results are used for empirical threshold selection; p remains fixed throughout each training run.

For a fixed problem family, let $H_{s,i}(n,p)$ denote the HV obtained on validation instance i with n cities by the model trained using seed s and threshold p. We first average HV over the five training runs for each validation instance and then over the 200 validation instances:

$$\overline{H}_i(n,p) = \frac{1}{5}\sum_{s=1}^{5} H_{s,i}(n,p), \qquad \mu(n,p) = \frac{1}{200}\sum_{i=1}^{200} \overline{H}_i(n,p). \tag{61}$$

The normalization baseline is the arithmetic mean of these mean validation HV values over the nine candidate thresholds for the same problem family and city count:

$$D(n) = \frac{1}{9} \sum_{p' \in \mathcal{P}} \mu(n, p'). \tag{62}$$

Relative HV is defined as

$$\text{RelativeHV}(n, p) = 100 \frac{\mu(n, p)}{D(n)}. \tag{63}$$

Here, $D(n) > 0$ is calculated separately for each problem family and city count and is shared by all nine thresholds within that configuration. Relative HV is expressed as a percentage. A value of 100% indicates that the mean validation HV equals the average over the candidate threshold grid; values above or below 100% indicate higher or lower mean validation HV relative to that average.

The between-instance standard deviation is calculated from the 200 seed-averaged validation-instance values:

$$\sigma(n, p) = \sqrt{\frac{1}{200} \sum_{i=1}^{200} \left[\overline{H}_i(n, p) - \mu(n, p) \right]^2}. \tag{64}$$

The shaded region in Figure 9 has limits

$$\left[100 \frac{\mu(n, p) - \sigma(n, p)}{D(n)}, \quad 100 \frac{\mu(n, p) + \sigma(n, p)}{D(n)} \right]. \tag{65}$$

Thus, the solid curve and its shaded limits use the same normalization denominator, computed entirely from validation results. The shaded region represents dispersion across validation instances after averaging over five training seeds; it is neither a confidence interval nor a measure of variability across training seeds. This normalization is applied only to summarize the sensitivity results and does not change the threshold during training.

Based on validation performance averaged over five independent training runs, the Euclidean configurations show smaller observed changes in mean Relative HV across the candidate values of p than the mixed-distance configurations. This observation concerns the sensitivity of average validation solution quality to p under the reported settings. The shaded regions describe dispersion across validation instances and do not establish stability across training seeds or low gradient-estimation variance. The selected values are empirical choices within the evaluated parameter set; they are not claimed to be unique or statistically established optima across problem sizes. The present experiments do not isolate the mechanism responsible for the difference between the Euclidean and mixed-distance settings.

The task-specific thresholds were selected empirically by comparing the mean validation Relative HV curves across the four evaluated city counts in Figure 9. This comparison led to one common threshold per problem family: $p = 0.5$ for bi-objective Euclidean problems, $p = 0.9$ for bi-objective mixed-distance problems, $p = 0.4$ for tri-objective Euclidean problems, and $p = 0.8$ for tri-objective mixed-distance problems. Selection was based on the observed mean validation performance, rather than the widths of the shaded regions. This was an empirical curve-based choice, not optimization of a separately specified aggregate score across city counts. The selected thresholds were fixed for the main experiments, and the separate final test results were not used to select or revise them.

4.8. Analysis of Model Generalization Performance

To evaluate the transfer of the trained model to the KroAB benchmarks without retraining, we conduct experiments on three bi-objective instances—KroAB100, KroAB150,

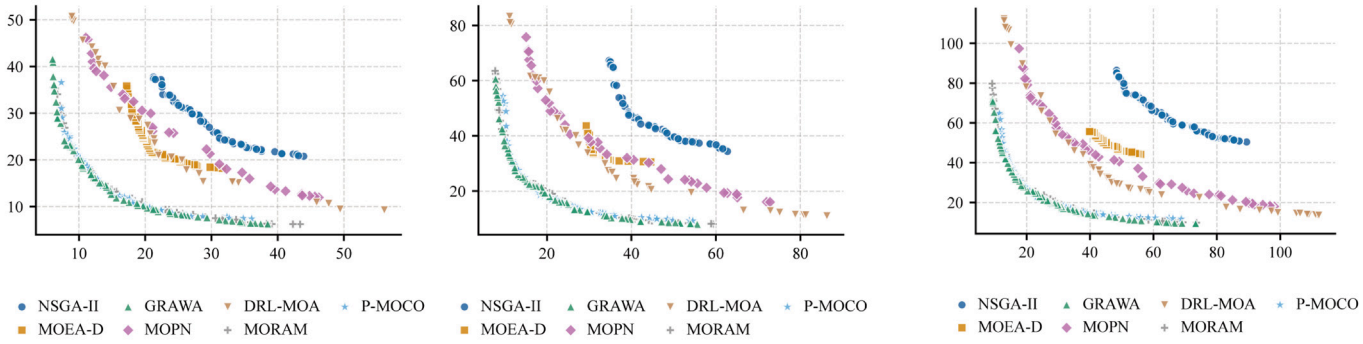
and KroAB200—constructed from the classical kroA and kroB instances in TSPLIB. For each KroAB instance, the coordinate arrays from the corresponding kroA and kroB files are scaled separately to $[0, 1]$. Within each file, a single minimum and maximum taken over all coordinate entries are used for both coordinate axes. Tour costs are then computed from the scaled coordinates. No further objective-value normalization is applied before evaluating HV or IGD+. The HV reference point is set to (n, n), where n denotes the problem size, measured by the number of cities. Accordingly, the reference points for KroAB100, KroAB150, and KroAB200 are $(100, 100)$, $(150, 150)$, and $(200, 200)$, respectively. For each KroAB instance, the empirical IGD+ reference set pools the saved nondominated outputs from all five trained models of each learning-based method and all 30 runs of each evolutionary solver. The pooled vectors undergo the joint nondominated filtering and duplicate handling described in Section 4.1.2. This common reference set is used to evaluate the selected seed-12347 checkpoint of each learning-based method and each of the 30 solver runs separately. Including all five trained models in reference construction does not change the selected-checkpoint reporting protocol. All learning-based models are trained solely on 40-node instances and directly evaluated on these benchmarks without any fine-tuning. Each KroAB benchmark contains one fixed test instance. NSGA-II and MOEA/D were each executed independently 30 times on KroAB100, KroAB150, and KroAB200, using every integer seed from 12345 through 12374 once per algorithm and instance. The per-run parameter settings and computational budgets were held fixed. HV, IGD+, and elapsed solver time were recorded separately for each run, and their arithmetic means were calculated across the 30 runs for each algorithm and instance. In particular, Time/s is the mean duration of one solver execution, rather than the sum of the 30 durations or the duration of the run selected for Figure 10. These means summarize average solution quality and runtime; run-to-run dispersion is not reported, and no claim of run-to-run stability is made from the means. A fixed benchmark instance does not imply deterministic outputs from a stochastic algorithm. For each learning-based method, the five training runs used seeds 12345, 12346, 12347, 12348, and 12349. We ranked the resulting models by their mean HV on an independent validation set and selected the middle, third-ranked model. This median-validation-HV criterion selected training seed 12347 for every learning-based method. The selected checkpoint, or corresponding set of subproblem-network checkpoints for DRL-MOA, was then held fixed for deterministic evaluation on KroAB100, KroAB150, and KroAB200 without fine-tuning. KroAB results were not used to select or revise the checkpoint. The reported learning-based results therefore describe the performance of the selected model, rather than a five-model average, and do not quantify variability across independent training runs. Table 7 summarizes solution quality and runtime on the three KroAB benchmarks under this evaluation protocol.

Table 7. Results on the KroAB benchmarks. For NSGA-II and MOEA/D, HV, IGD+, and Time/s are arithmetic means over 30 runs with seeds 12345–12374; Time/s denotes the mean duration of one solver execution. Learning-based quality values use deterministic decoding with the fixed checkpoint from training seed 12347, selected for each method by median mean HV on an independent validation set among five training runs with seeds 12345–12349. IGD+ uses one common reference set per instance, constructed from all five training-model outputs of each learning-based method and all 30 runs of each evolutionary solver.

Algorithm	KroAB100			KroAB150			KroAB200		
	HV	IGD+	Time/s	HV	IGD+	Time/s	HV	IGD+	Time/s
NSGA-II	6.169e+03	1.776e+01	6.722	1.286e+04	3.410e+01	8.142	2.186e+04	5.203e+01	8.897
MOEA/D	6.695e+03	1.081e+01	2.196e+01	1.451e+04	1.995e+01	2.412e+01	2.510e+04	3.034e+01	2.684e+01
DRL-MOA	7.729e+03	9.620e+00	5.413	1.797e+04	1.478e+01	7.489	3.273e+04	1.940e+01	9.593

Table 7. *Cont.*

Algorithm	KroAB100			KroAB150			KroAB200		
	HV	IGD+	Time/s	HV	IGD+	Time/s	HV	IGD+	Time/s
MOPN	7.411e+03	1.191e+01	4.628	1.714e+04	1.756e+01	6.812	3.149e+04	2.320e+01	8.649
MORAM	8.604e+03	3.681e-01	3.119e-01	1.979e+04	4.360e-01	1.985e-01	3.564e+04	8.359e-01	2.155e-01
P-MOCO	8.452e+03	4.478e-01	4.843	1.948e+04	5.518e-01	6.409	3.508e+04	7.491e-01	8.092
GRAWA	**8.625e+03**	**7.688e-02**	4.177e-01	**1.980e+04**	**2.088e-01**	1.452e-01	**3.574e+04**	**1.369e-01**	2.118e-01



(a)KroAB100 (b)KroAB150 (c)KroAB200

Figure 10. Pareto front approximations on KroAB100, KroAB150, and KroAB200. For NSGA-II and MOEA/D, each plotted set is taken from one run selected using the median-HV criterion among 30 runs with seeds 12345–12374 on the corresponding instance. Learning-based results use deterministic decoding with the fixed checkpoint from training seed 12347, selected by median mean HV on an independent validation set. The solver-run selection is used for visualization; Table 7 reports the 30-run mean HV and IGD+ for NSGA-II and MOEA/D.

Figure 10 shows Pareto front approximations obtained on KroAB100, KroAB150, and KroAB200. For NSGA-II and MOEA/D, the displayed objective vectors are taken from a single run selected using the median-HV criterion among the 30 independent runs for each algorithm and benchmark instance. Each displayed solution set comes entirely from that selected run. For the learning-based methods, the displayed results are obtained using deterministic decoding with the fixed checkpoints described above. In contrast, Table 7 reports the arithmetic means of HV and IGD+ over all 30 runs for NSGA-II and MOEA/D. The plots provide a visual comparison under the stated evaluation settings. These results assess transfer from 40-city training instances to the evaluated KroAB benchmarks; they do not isolate the contribution of individual architectural components or establish uniform coverage of the entire Pareto front.

Table 7 further compares the algorithms in terms of hypervolume (HV), IGD+, and runtime. GRAWA obtains the best HV on all three instances, although its margin over MORAM is small (for example, 8.625×10^3 versus 8.604×10^3 on KroAB100). The advantage becomes far more pronounced under IGD+, where GRAWA achieves 7.688×10^{-2}, 2.088×10^{-1} and 1.369×10^{-1} on the three instances, reducing the best baseline value by factors of about 4.8, 2.1 and 6.1, respectively. The HV values of GRAWA, MORAM, and P-MOCO are numerically close on these benchmarks, while GRAWA has lower reported IGD+. This indicates a smaller average modified distance from the empirical reference vectors to GRAWA's approximation under the stated metric definition. These aggregate values do not, by themselves, establish more uniform coverage or fewer uncovered regions throughout the reference front. For each KroAB benchmark, runtime refers to solving one individual instance; for NSGA-II and MOEA/D, Table 7 reports the mean runtime over 30 independent executions. In Table 2, total evaluation time is averaged over five evaluations, one for each independently trained model, with every evaluation processing the

same complete set of 200 random test instances containing 200 cities each. Thus, the tables differ in both the workload covered by each timed execution and the number of executions used to calculate the reported mean.

Table 7 reports the runtime measurements obtained on the KroAB benchmarks under the specified hardware and algorithmic settings. These measurements depend on the decoding strategy, candidate-generation budget, and operations included in the timed interval. Because the implementations use different timing boundaries, the reported values should not be interpreted as a controlled comparison of end-to-end solving speed. The solution-quality results provide evidence of transfer from training instances with 40 cities to the evaluated KroAB benchmarks with up to 200 cities.

5. Conclusions

This paper proposes an end-to-end algorithm, GRAWA, to solve the multi-objective traveling salesman problem based on a single deep reinforcement learning model. GRAWA generates approximate Pareto sets through preference-conditioned tour construction, with solution quality evaluated using HV and IGD+. Specifically, the graph weight encoder combines global and objective-specific city encoders with a graph weight routing module operating on a fixed preference graph. The routing module uses established graph-convolution and pooling/unpooling components to learn preference representations, which are integrated with city features through the graph co-guided fusion module. The architectural contribution lies in this explicit preference-relation modeling and its integration into a shared MOTSP policy. A Top-p training baseline selects reference candidates from the sampled tours according to their scalarized costs. The experiments demonstrate that GRAWA can be applied without retraining to larger unseen KroAB benchmarks with up to 200 cities. The reported runtime results are specific to the evaluated hardware, decoding budgets, and timing boundaries. Meanwhile, the results also delineate its current boundary: P-MOCO maintains an advantage on mixed-distance instances, indicating that modeling heterogeneous metric spaces remains an open direction. The Top-p update is not guaranteed to be unbiased; the leave-one-out comparison assesses its effect on final HV performance rather than establishing gradient-variance reduction. In future work, GRAWA can be further extended to other multi-objective combinatorial optimization problems, such as the multi-objective vehicle routing problem [31] and the multi-modal multi-objective traveling salesman problem [32].

Author Contributions: Conceptualization, Z.J. and X.Z. (Xiaoxiong Zhang); methodology, Z.J. and X.Z. (Xiaoxiong Zhang); data curation, X.Z. (Xiaoxiong Zhang) and S.Y.; writing—original draft preparation, Z.J.; writing—review and editing, X.Z. (Xiaoxiong Zhang), S.Y. and X.Z. (Xixi Zhu); Supervision, S.Y. and X.Z. (Xixi Zhu). All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded in part by the National Natural Science Foundation of China under Grant No. 72471236 and by the Open Fund of the State Key Laboratory under Grant No. 2025-JCJQ-LB-009-002.

Data Availability Statement: The original contributions presented in the study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Statistical Analysis and Empirical Evaluation of the Top-p Baseline

Appendix A.1. Bias Analysis of the Top-p Update Estimator

For a fixed instance and preference vector, we analyze the expected update associated with the proposed Top-p baseline. The scalarized cost of a given tour does not explicitly depend on the policy parameters. Define the expected scalarized cost as

$$J_{ij}(\theta) = \mathbb{E}_{x_i^j \sim p_\theta(\cdot|S_i, w^j)}\Big[g(x_i^j \mid w^j)\Big]. \tag{A1}$$

where S_i is the i-th problem instance, w^j is the j-th preference vector, x_i^j is the tour sampled from the policy p_θ, θ denotes the policy parameters, and $g(x_i^j|w^j)$ is the scalarized tour cost.

The Top-p baseline is computed from the candidate tours generated for the same instance, including the current tour. All candidates are evaluated under the current preference vector before Top-p selection and weighted averaging. Although the candidate costs are detached from the computational graph, the baseline remains statistically dependent on the sampled tours. The corresponding update term is

$$\hat{h}_{ij} = \Big[g(x_i^j \mid w^j) - b_{ij}(\mathcal{X}_i)\Big]s_{ij}. \tag{A2}$$

where $\hat{h}_{ij}$ is the update term for the instance–preference pair (S_i, w^j), $\mathcal{X}_i = (x_i^1, \ldots, x_i^N)$ is the collection of N candidate tours, $b_{ij}(\mathcal{X}_i)$ is the Top-p baseline evaluated under w^j, and $s_{ij} = \nabla_\theta \log p_\theta(x_i^j \mid S_i, w^j)$ is the score-function gradient.

Under the standard score-function identity, the bias of this update term is

$$\mathbb{E}[\hat{h}_{ij}] - \nabla_\theta J_{ij}(\theta) = -\mathbb{E}\Big[b_{ij}(\mathcal{X}_i)s_{ij}\Big]. \tag{A3}$$

where the expectations are taken over the joint sampling of candidate tours conditional on the instance and preference vectors, $\nabla_\theta J_{ij}(\theta)$ is the gradient of the original expected scalarized cost, and the right-hand side represents the bias induced by the baseline term.

For a baseline conditionally independent of the current tour, the right-hand side of Equation (A3) vanishes because the expected score-function gradient is zero. This cancellation is not generally guaranteed for the original Top-p baseline, since the current tour contributes to its construction. Therefore, the proposed update is not guaranteed to be unbiased.

This analysis identifies the potential bias but does not establish its magnitude or a reduction in gradient variance. Accordingly, Top-p is treated as a heuristic training baseline. Its empirical performance, including the comparison with a leave-one-out variant, should not be interpreted as evidence of unbiasedness or variance reduction.

Appendix A.2. Performance Comparison with a Leave-One-Out Top-p Baseline

To assess the effect of including the current trajectory in baseline construction, we compare the original Top-p baseline with a leave-one-out variant. For each policy-gradient term, the leave-one-out variant excludes the corresponding sampled trajectory before recomputing the softmin probabilities, selecting the Top-p subset, and calculating the normalized weighted baseline. All remaining candidate tours are evaluated under the current preference vector.

For each training setting, the original and leave-one-out Top-p configurations were each trained independently five times using five different random seeds. Within each test configuration, all five runs of both variants were evaluated on the same 200 test instances. For each variant, HV was first averaged across the five training runs for each instance and then averaged across the 200 instances. The comparison in Figure A1 is based on these

aggregated mean HV values, rather than on a single selected training run. The five training runs are not treated as additional independent test instances.

Figure A1 presents the relative differences between the aggregated mean HV values of the two variants across 16 MOTSP test configurations. For each configuration, the relative difference is defined as

$$\Delta_{\mathrm{HV}} = \frac{\overline{\mathrm{HV}}_{\mathrm{original}} - \overline{\mathrm{HV}}_{\mathrm{LOO}}}{\overline{\mathrm{HV}}_{\mathrm{LOO}}} \times 100\%. \tag{A4}$$

where $\overline{HV}_{\mathrm{original}}$ and $\overline{HV}_{\mathrm{LOO}}$ denote the mean HV values for the original and leave-one-out Top-p baselines, respectively, after averaging over five independent training runs for each test instance and then across the same 200 test instances. The denominator is the aggregated mean HV of the leave-one-out variant for the corresponding test configuration. Positive values indicate higher mean HV for the original formulation, whereas negative values favor the leave-one-out variant. Thus, Δ_{HV} is the relative difference between the aggregated means, not the average of separately calculated per-seed or per-instance relative differences.

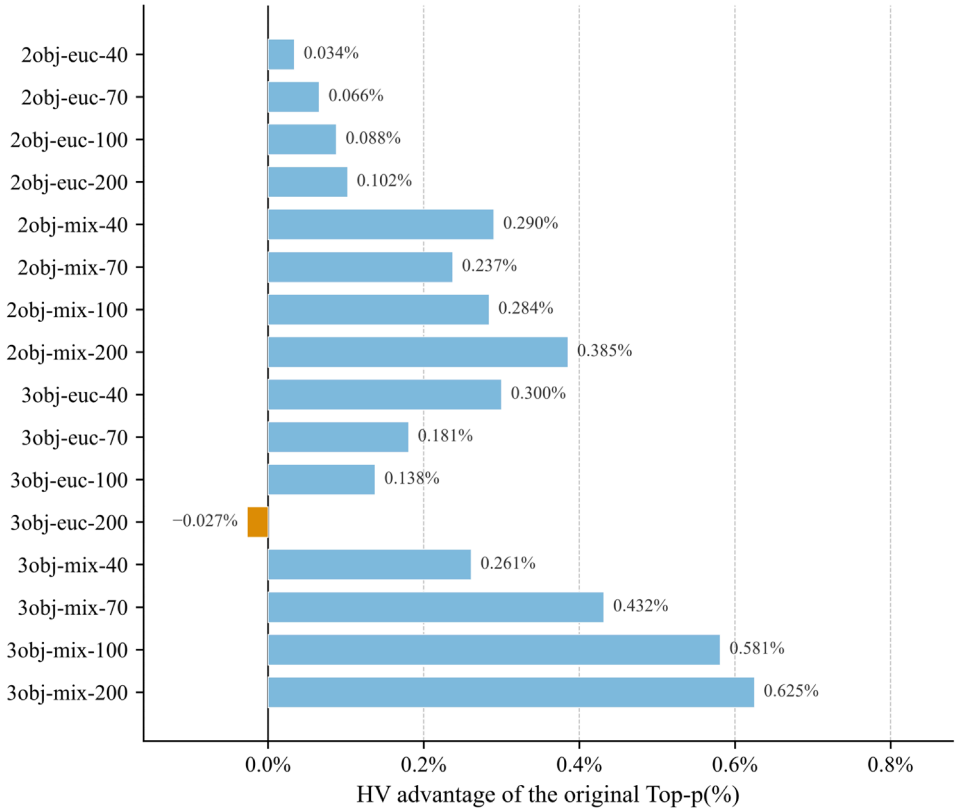


Figure A1. Relative differences in mean HV between the original and leave-one-out Top-p baselines across 16 MOTSP test configurations. Blue bars indicate positive differences favoring the original Top-p baseline, while orange bars indicate negative differences favoring the leave-one-out variant.

Based on mean HV aggregated over five independent training runs and the shared test instances, the relative differences in Figure A1 range from −0.027% to 0.625%. The original Top-p baseline achieves higher mean HV in 15 of the 16 test configurations, while the leave-one-out variant achieves slightly higher mean HV on 3obj-euc-200. The largest observed advantage of the original formulation occurs on 3obj-mix-200, with a relative difference of 0.625%. All absolute relative differences remain below 0.7%, indicating numerically close average final HV values under the evaluated conditions. These differences are descriptive and are not presented as evidence of statistical significance or equivalence.

This comparison provides an empirical assessment of average final HV performance over five independent training runs. Figure A1 summarizes differences between aggregated means; it does not quantify between-training-run variability or establish that the bias identified in Appendix A.1 is negligible or that gradient variance is reduced. Similar average final HV values do not imply identical expected updates or statistical equivalence. Moreover, excluding the current trajectory changes both the baseline values and their statistical properties; therefore, observed performance differences cannot be attributed exclusively to estimator bias. We retain the original Top-p formulation as the default to maintain consistency with the training procedure underlying the main experiments. The leave-one-out comparison serves as a sensitivity analysis of this choice. The numerically close final HV values do not establish a statistically significant advantage for either variant, and retention of the original formulation is a pragmatic choice rather than a claim of theoretical superiority. Leave-one-out remains a principled alternative when the remaining candidate tours are sampled conditionally independently of the excluded trajectory, given the instance, preferences, and policy parameters. The original update remains subject to the potential bias identified in Appendix A.1.

Appendix B. Tensor Dimensions in Graph Weight Routing and Fusion

Table A1 summarizes the tensor dimensions used in Sections 3.1.2 and 3.2. The batch dimension is omitted. City-level tensors are shown for one instance and one preference, while preference-graph tensors retain the full preference dimension N. We use $d = 128$ and $d_h = 16$. Here, N denotes the number of preference vectors, n the number of cities, M the number of objectives, d the embedding dimension, K the number of retained preference nodes, and d_h the dimension of each attention head.

Table A1. Tensor dimensions in the graph weight routing and graph co-guided fusion modules.

Symbol	Shape	Description
W	$N \times M$	Preference vectors
A_w	$N \times N$	Base adjacency
$Z^{(0)}$, H, Z_{up}, Z_w, Z_{pref}	$N \times d$	Preference-node features before and after graph processing
a,y,I	d,N,K	Projection, scores, selected indices
$\widetilde{Z}$, Y	$K \times d$	Pooled features
A_{pool}	$K \times K$	Induced adjacency
V,U_m,U^j	$n \times d$	City embeddings
Q_h^j, K_h, V_h	$n \times d_h$	Per-head attention projections
$Q_h^j (K_h)^\top$	$n \times n$	Attention scores
Z_o^j, Z_c^j	$n \times d$	Concatenated query representation and cross-attention context
r^j,R^j	$1 \times d$, $n \times d$	Graph feature and its broadcast
F^j	$n \times 3d$	Concatenated fusion input
G_c^j, G_o^j	$n \times d$	Channel-wise gates
T_c^j, T_o^j	$n \times d$	Transformed context and original branches
T^j	$n \times d$	Representation after the first residual addition and batch normalization
Z_f^j, E^j	$n \times d$	Fused and final city features

Appendix C. Construction of the Fixed Preference Graph

This appendix provides an explicit construction procedure for the base preference graph defined in Section 3.1.2. The input is the preference matrix $W = [(w^1)^\top; \ldots; (w^N)^\top] \in \mathbb{R}^{N \times M}$, whose rows are stored in a fixed lexicographic order. Each preference vector satisfies $w_m^j \geq 0$ and $\sum_{m=1}^{M} w_m^j = 1$. The experiments use 101 bi-objective vectors $(t/100, 1 - t/100)$ for

$t = 0,\ldots,100$, or 91 tri-objective vectors $(a/12, b/12, c/12)$ for nonnegative integers satisfying $a + b + c = 12$. The nearest-neighbor selection size is fixed at $k_w = 4$.

For each node, candidates are ordered first by Euclidean distance in preference space and then by their fixed node index to resolve distance ties. The four nearest distinct nodes are selected. The union of these directed selection relations defines the symmetric binary adjacency matrix. The procedure uses only preference vectors and does not require city coordinates, sampled tours, objective-space points, or the true Pareto front.

Algorithm A1 Construction of the fixed preference graph

Input: $W = [(w^1)^T; \ldots; (w^N)^T]$, with rows in fixed lexicographic order; $k_w = 4$.
Output: Symmetric binary adjacency A_w; final graph neighborhoods $B(1),\ldots,B(N)$.
Initialize A_w as an $N \times N$ zero matrix;
for $j = 1$ **to** N **do**
  $C_j \leftarrow \{1,\ldots,N\} \setminus \{j\}$;
  For each $r \in C_j$, compute $d_w(j,r) \leftarrow \sqrt{\sum_{m=1}^{M} (w_m^j - w_m^r)^2}$;
  Sort C_j in ascending order by the pair $(d_w(j,r), r)$;
  $\mathcal{N}_w(j) \leftarrow$ the first k_w indices in the sorted C_j;
end for
for $j = 1$ **to** N **do**
  for each $r \in \mathcal{N}_w(j)$ **do**
    $(A_w)_{j,r} \leftarrow 1$;
    $(A_w)_{r,j} \leftarrow 1$;
  end for
end for
for $j = 1$ **to** N **do**
  $(A_w)_{j,j} \leftarrow 0$;
  $B(j) \leftarrow \left\{ r \neq j : (A_w)_{j,r} = 1 \right\}$;
end for
Return A_w **and** $B(1),\ldots,B(N)$.

Here, $\mathcal{N}_w(j)$ is the four-neighbor selection set before symmetrization, whereas $B(j)$ is the final adjacent-node index set. Because an edge is included if either endpoint selects the other, the resulting node degree may exceed k_w. All existing edges have binary value one, and no trainable edge weights are introduced. The graph is constructed once for a given preference set and shared by all problem instances using that set. The base adjacency has no self-loops. Self-loops and symmetric degree normalization are introduced subsequently within graph convolution, as specified in Section 3.1.2; they are not part of Algorithm A1.

Appendix D. Sensitivity of Weighted-Sum Subproblems to Preference Perturbations

We provide a cost-based justification for exchanging information between neighboring preferences. Weighted-sum scalarization and its approximation properties in multi-objective optimization have been studied by Bazgan et al. [33]. The elementary sensitivity bounds below are proved here for a fixed finite feasible set; they do not assume convexity or continuity of the discrete Pareto front.

Fix one MOTSP instance. Let $\mathcal{X}$ be its nonempty finite set of feasible tours, and let $f(x) = (f_1(x), \ldots, f_M(x))^\top$ contain the complete tour costs, including the closing edge. These are the objective costs used by the model, with no additional normalization introduced for this analysis. For a preference vector in the unit simplex, define

$$\Delta^{M-1} = \{w \in \mathbb{R}^M : w_m \geq 0,\ \mathbf{1}^\top w = 1\}, \qquad g(x|w) = w^\top f(x), \qquad v(w) = \min_{x \in \mathcal{X}} g(x|w). \tag{A5}$$

Because the feasible set is finite, the following instance-dependent constants are finite:

$$C_f = \max_{x \in \mathcal{X}} \|f(x)\|_2, \qquad D_f = \max_{x,z \in \mathcal{X}} \|f(x) - f(z)\|_2. \tag{A6}$$

Proposition A1 (sensitivity in scalarized cost). *For any w,$w' \in \Delta^{M-1}$, the scalarized objectives and optimal values satisfy*

$$\sup_{x \in \mathcal{X}} \left| g(x \mid w) - g(x \mid w') \right| \leq C_f \|w - w'\|_2, \qquad \left| v(w) - v(w') \right| \leq C_f \|w - w'\|_2. \tag{A7}$$

Furthermore, if a feasible tour $\hat{x}$ has an additive optimality gap at most $\varepsilon \geq 0$ under w', that is, $g(\hat{x} \mid w') - v(w') \leq \varepsilon$, then

$$0 \leq g(\hat{x} \mid w) - v(w) \leq \varepsilon + D_f \|w - w'\|_2. \tag{A8}$$

In particular, when $\hat{x}$ is optimal under w', the bound holds with $\varepsilon = 0$.

Proof. For every feasible tour, the Cauchy–Schwarz inequality gives

$$\left| g(x \mid w) - g(x \mid w') \right| = \left| (w - w')^\top f(x) \right| \leq C_f \|w - w'\|_2$$

Taking an optimizer under w' gives $v(w) - v(w') \leq C_f \|w - w'\|_2$; interchanging the weights proves the optimal-value bound. To prove the transfer bound, choose $x_w \in \operatorname{argmin}_{x \in \mathcal{X}} g(x|w)$. Then

$$\begin{aligned} g(\hat{x} \mid w) - v(w) &= w^\top [f(\hat{x}) - f(x_w)] \\ &= w'^\top [f(\hat{x}) - f(x_w)] + (w - w')^\top [f(\hat{x}) - f(x_w)] \\ &\leq \varepsilon + \|w - w'\|_2 \|f(\hat{x}) - f(x_w)\|_2 \\ &\leq \varepsilon + D_f \|w - w'\|_2. \end{aligned} \tag{A9}$$

The first term is at most ε because $g(x_w|w') \geq v(w')$. Nonnegativity follows from the definition of $v(w)$.□

Corollary A1 (local constancy under a strict optimality gap). *Let $\mathcal{Y} = \{\mathbf{f}(\mathbf{x}) : \mathbf{x} \in \mathcal{X}\}$ contain at least two distinct objective vectors. Suppose $\mathbf{y}^*$ uniquely minimizes $\mathbf{w}^\top \mathbf{y}$ over $\mathcal{Y}$, and define*

$$\gamma = \min_{\mathbf{y} \in \mathcal{Y} \setminus \{\mathbf{y}^*\}} \mathbf{w}^\top (\mathbf{y} - \mathbf{y}^*) > 0. \tag{A10}$$

For any $\mathbf{w}' \in \Delta^{M-1}$ satisfying $\|\mathbf{w}' - \mathbf{w}\|_2 < \gamma / D_f$, the same objective vector $\mathbf{y}^*$ remains the unique minimizer.

Proof. For every $\mathbf{y} \neq \mathbf{y}^*$,

$$\begin{aligned} \mathbf{w}'^\top (\mathbf{y} - \mathbf{y}^*) &\geq \gamma - \|\mathbf{w}' - \mathbf{w}\|_2 \|\mathbf{y} - \mathbf{y}^*\|_2 \\ &\geq \gamma - D_f \|\mathbf{w}' - \mathbf{w}\|_2 > 0. \end{aligned} \tag{A11}$$

Hence, $\mathbf{y}^*$ remains uniquely optimal in objective space. This conclusion does not imply a unique tour, since distinct tours can share the same objective vector. □

Implications and limitations. Proposition A1 applies directly to the Euclidean preference distance used to construct the fixed graph. It supports a notion of related subproblems based on scalarized cost: a tour that performs well under one preference incurs a bounded

additional cost under a nearby preference. The constants depend on the instance and objective scales, so the result does not establish a universal numerical neighborhood radius or justify a particular nearest-neighbor count. For a learned tour, the bound retains its original optimality gap; it does not assert that GRAWA attains a known approximation guarantee.

The corollary gives a stronger local conclusion only when a strict gap exists. Near a tie between distinct minimizing objective vectors, an arbitrarily small weight perturbation can change the optimizer discontinuously. For example, consider the illustrative finite objective set $\mathcal{Y} = \{(1,3),(3,1)\}$. For $0 < \delta < 1/2$, the preferences $(1/2 + \delta, 1/2 - \delta)$ and $(1/2 - \delta, 1/2 + \delta)$ select different vectors. Their distance is $2\sqrt{2}\delta$, while the distance between the selected objective vectors remains $2\sqrt{2}$. This example illustrates the limitation for finite objective sets; it is not an additional MOTSP experiment.

Accordingly, neither small preference distance nor the cost bound guarantees proximity of optimal objective vectors or similarity of tour edges. GRAWA uses preference adjacency as an inductive bias for exchanging learned features, rather than as a known adjacency relation on the Pareto front. The routing ablation evaluates the contribution of the complete module but does not establish continuity of the optimal-solution mapping.

Appendix E. Additional Pareto Front Approximations for 40-, 70-, and 100-City MOTSP Instances

This appendix presents the Pareto front approximation plots for 40-, 70-, and 100-city MOTSP instances that supplement the 200-city comparisons in Figure 7. The additional panels cover the same four problem families: bi-objective Euclidean, tri-objective Euclidean, bi-objective mixed-distance, and tri-objective mixed-distance. They are provided separately to retain the comparisons across problem sizes while reducing the visual density of the main figure.

In Figure A2, the columns correspond to 40, 70, and 100 cities, respectively. The rows show bi-objective Euclidean, tri-objective Euclidean, bi-objective mixed-distance, and tri-objective mixed-distance problems, respectively. The plotted coordinates are objective costs, with all objectives minimized. These visualizations complement the quantitative results for all 16 configurations in Tables 3 and 4 and do not imply that the generated tours are Pareto-optimal over the full feasible space.

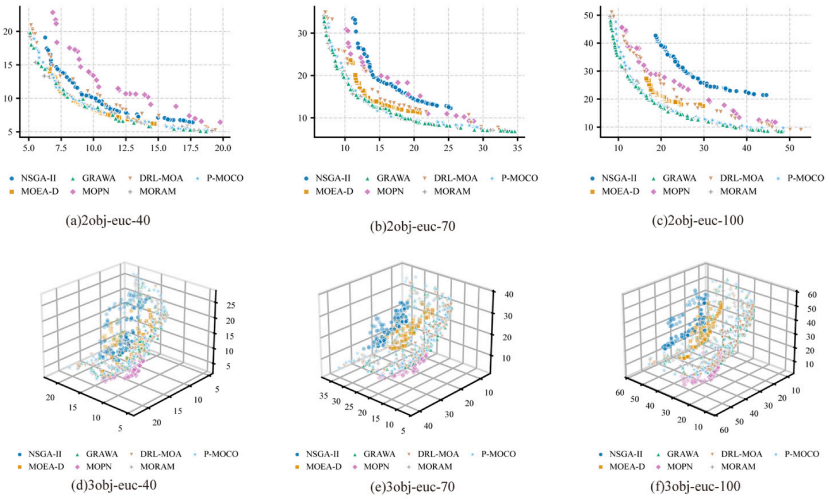


Figure A2. *Cont.*

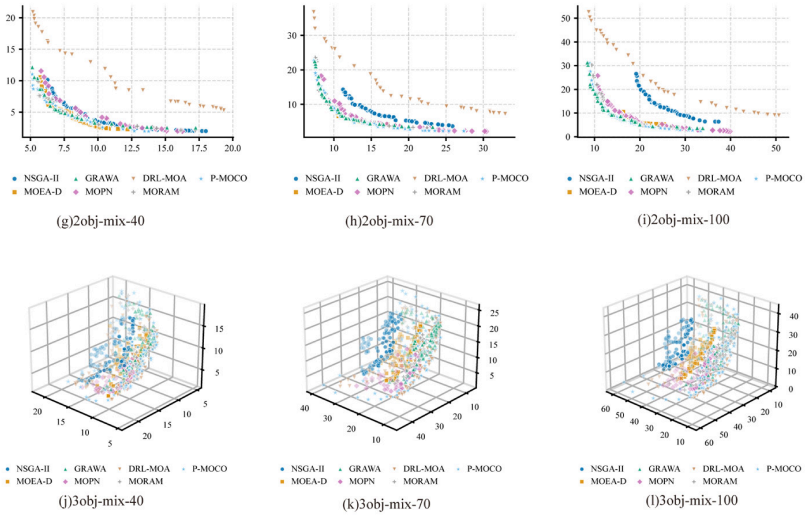


Figure A2. Additional Pareto front approximations for MOTSP instances with 40, 70, and 100 cities.

References

1. Zhang, Z.; Liu, F. Logistics-supply-chain-enhanced human urbanization algorithm for global optimization and engineering applications. *Mathematics* **2026**, *14*, 3053. [CrossRef]
2. Shang, W.; Fu, J.; Ma, J.; Lin, J.; Tian, J.; Yu, M. Internet-of-things-based low-carbon distribution route optimization for logistics. *IEEE Internet Things J.* **2025**, *12*, 3577–3588. [CrossRef]
3. Castañeda Londoño, J.F.; Gallego Rendón, R.A.; Toro Ocampo, E.M. Chu–Beasley genetic algorithm and Google's OR-Tools for the bi-objective multiple traveling salesman problem with weighted arcs. *Oper. Res. Forum* **2026**, *7*, 78. [CrossRef]
4. Zheng, A.; Liang, X.; Zhong, H.; Zhang, Z.; Lv, Z.; Zhang, Z.; Zheng, M. Distributionally robust integrated "decision–control" task assignment for multiple unmanned aerial systems in emergency response under stochastic disturbances. *Mathematics* **2026**, *14*, 3044. [CrossRef]
5. He, J.; Jia, Z.; Cui, C.; Cui, Y.; Zhu, Q.; Wu, Q. Joint resource allocation and task scheduling for LAWN-enabled emergency response. *IEEE Internet Things J.* **2026**, *13*, 42940–42953. [CrossRef]
6. Shao, S.; Tian, Y.; Zhang, Y.; Yang, S.; Zhang, P.; He, C.; Zhang, X.; Jin, Y. Evolutionary computation for sparse multi-objective optimization: A survey. *ACM Comput. Surv.* **2025**, *57*, 1–35. [CrossRef]
7. Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **2002**, *6*, 182–197. [CrossRef]
8. Zhang, Q.; Li, H. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Trans. Evol. Comput.* **2007**, *11*, 712–731. [CrossRef]
9. Deb, K.; Jain, H. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints. *IEEE Trans. Evol. Comput.* **2014**, *18*, 577–601. [CrossRef]
10. Yang, K.; Pan, D. An improved mayfly optimization algorithm for type-2 multi-objective integrated process planning and scheduling. *Mathematics* **2023**, *11*, 4384. [CrossRef]
11. Paquete, L.; Chiarandini, M.; Stützle, T. Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In *Metaheuristics for Multiobjective Optimisation*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 177–199.
12. Paquete, L.; Stützle, T. A two-phase local search for the biobjective traveling salesman problem. In *Proceedings of the International Conference on Evolutionary Multi-Criterion Optimization*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 479–493.
13. Lust, T.; Teghem, J. Two-phase Pareto local search for the biobjective traveling salesman problem. *J. Heuristics* **2010**, *16*, 475–510. [CrossRef]
14. Lust, T.; Teghem, J. The multiobjective traveling salesman problem: A survey and a new approach. In *Advances in Multi-Objective Nature Inspired Computing*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 119–141.
15. Jaszkiewicz, A. Genetic local search for multi-objective combinatorial optimization. *Eur. J. Oper. Res.* **2002**, *137*, 50–71. [CrossRef]
16. Kumar, R.; Singh, P.K. Pareto evolutionary algorithm hybridized with local search for biobjective TSP. In *Hybrid Evolutionary Algorithms*; Abraham, A., Grosan, C., Ishibuchi, H., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; pp. 361–398.
17. Jaszkiewicz, A.; Zielniewicz, P. Pareto memetic algorithm with path relinking for bi-objective traveling salesperson problem. *Eur. J. Oper. Res.* **2009**, *193*, 885–890. [CrossRef]
18. Li, K.; Zhang, T.; Wang, R. Deep reinforcement learning for multiobjective optimization. *IEEE Trans. Cybern.* **2021**, *51*, 3103–3114. [CrossRef] [PubMed]

19. Vinyals, O.; Fortunato, M.; Jaitly, N. Pointer networks. In Proceedings of the Advances in Neural Information Processing Systems 28 (NIPS 2015), Montreal, QC, Canada, 7–12 December 2015; pp. 2692–2700.
20. Wu, H.; Wang, J.; Zhang, Z. MODRL/D-AM: Multiobjective deep reinforcement learning algorithm using decomposition and attention model for multiobjective optimization. In *Proceedings of the International Symposium on Intelligence Computation and Applications*; Springer: Singapore, 2019; pp. 575–589.
21. Kool, W.; van Hoof, H.; Welling, M. Attention, learn to solve routing problems! In Proceedings of the International Conference on Learning Representations (ICLR), New Orleans, LA, USA, 6–9 May 2019.
22. Zhang, Y.; Wang, J.; Zhang, Z.; Zhou, Y. MODRL/D-EL: Multiobjective deep reinforcement learning with evolutionary learning for multiobjective optimization. In *Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN)*; IEEE: Piscataway, NJ, USA, 2021; pp. 1–8.
23. Zhang, Z.; Wu, Z.; Zhang, H.; Wang, J. Meta-learning-based deep reinforcement learning for multiobjective optimization problems. *IEEE Trans. Neural Netw. Learn. Syst.* **2023**, *34*, 7978–7991. [CrossRef] [PubMed]
24. Lin, X.; Yang, Z.; Zhang, Q. Pareto set learning for neural multi-objective combinatorial optimization. In Proceedings of the International Conference on Learning Representations (ICLR), Virtual, 25–29 April 2022.
25. Wang, Z.; Yao, S.; Li, G.; Zhang, Q. Multiobjective combinatorial optimization using a single deep reinforcement learning model. *IEEE Trans. Cybern.* **2024**, *54*, 1984–1996. [CrossRef] [PubMed]
26. Gao, L.-Y.; Wang, R.; Jia, Z.-H.; Liu, C. Multiobjective optimization for traveling salesman problem: A deep reinforcement learning algorithm via transfer learning. *IEEE Trans. Artif. Intell.* **2025**, *6*, 896–908. [CrossRef]
27. Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.* **1992**, *8*, 229–256. [CrossRef]
28. Kipf, T.N.; Welling, M. Semi-supervised classification with graph convolutional networks. In Proceedings of the International Conference on Learning Representations (ICLR), Toulon, France, 24–26 April 2017.
29. Gao, H.; Ji, S. Graph U-Nets. *IEEE Trans. Pattern Anal. Mach. Intell.* **2022**, *44*, 4948–4960. [CrossRef] [PubMed]
30. Blank, J.; Deb, K. Pymoo: Multi-objective optimization in Python. *IEEE Access* **2020**, *8*, 89497–89509. [CrossRef]
31. Duan, H.; Li, X.; Feng, Y.; Zhang, G.; Lu, Q.; Meng, X. Multi-objective hyper heuristic memetic algorithm for time-dependent vehicle routing problem with drone considering simultaneous pickup and delivery. *Expert Syst. Appl.* **2026**, *297*, 129196. [CrossRef]
32. Zhang, L.; Li, W.; Zheng, Y.; Wang, R.; Zhang, T.; Wang, L. A multi-strategy local search memetic algorithm for multimodal traveling salesman problem. *Swarm Evol. Comput.* **2026**, *105*, 102393. [CrossRef]
33. Bazgan, C.; Ruzika, S.; Thielen, C.; Vanderpooten, D. The power of the weighted sum scalarization for approximating multiobjective optimization problems. *Theory Comput. Syst.* **2022**, *66*, 395–415. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.