PROJECTING LATENT RL ACTIONS: TOWARDS GENERALIZABLE AND SCALABLE GRAPH COMBINATORIAL OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph combinatorial optimization (GCO) has attracted growing interest, as many NP-hard problems naturally admit graph formulations, yet their combinatorial explosion renders exact methods computationally intractable. Recent advances in Reinforcement Learning (RL) combined with Graph Neural Networks (GNNs) have significantly improved learning-based GCO solvers. However, existing approaches face limitations in both generalization across diverse graph instances and computational scalability as action spaces grow. To address both challenges, we introduce *projection agents*, a GCO methodology built on the Wolpertinger framework and adapted to a shared GNN-based latent space for observations and actions. These agents are trained to solve GCO problems by predicting a desired latent action in a single forward pass, which is subsequently decoded into a valid discrete action in the graph using the latent space. Across diverse benchmarks, our approach achieves up to 6.41$\times$ faster inference and up to 75% better generalization than existing solutions using a simple nearest-neighbor decoding, while opening the door to strong RL performance in super-linear decision spaces with multiple interdependent variables. Finally, we release `LaGCO-RL`, a Python library that automates latent action-space construction and supports existing RL-GCO solutions, promoting reproducibility and adaptation to new GCO benchmarks.

1 INTRODUCTION

Solving graph-based combinatorial optimization (GCO) problems has attracted increasing interest, as many NP-hard problems across several domains can naturally be modeled as graphs (Darvariu et al., 2024). In the seminal work of Karp (1972) introducing 21 NP-complete problems (e.g., Traveling Salesman Problem (Jünger et al., 1995)), nearly half correspond to versions of graph optimization problems, and most of them can also be formulated on graphs (Cappart et al., 2021a). However, due to the combinatorial nature of these problems, where the number of possible solutions grows rapidly with graph size, exact methods become computationally intractable (Li et al., 2022).

As a result, heuristic methods are widely employed, trading optimality for computational efficiency (Vesselinova et al., 2020). However, traditional heuristics typically rely on domain expertise, handcrafted rules, and iterative trial-and-error (Li et al., 2018), often demanding substantial human effort to redesign and adapt them for each variation of the problem. More recently, machine learning (ML) approaches have emerged as a promising alternative for data-driven heuristic discovery (Bengio et al., 2021; Cappart et al., 2021a; Mazyavkina et al., 2021). In particular, reinforcement learning (RL) methods (Cappart et al., 2021b) have attracted increasing attention because they do not require labeled samples, which are costly to obtain for large instances (Peng et al., 2021; Sanokowski et al., 2023), and construct solutions autoregressively through sequential decision-making, thereby mirroring the process employed by traditional GCO solvers (Darvariu et al., 2024). RL has been used both for direct heuristic approximation and to accelerate exact solvers, e.g., by guiding branch-and-bound in constraint programming (Cappart et al., 2021b). Therefore, improving RL for GCO (RL-GCO) is important to advance both directions.

A key advancement of RL-GCO is the use of Graph Neural Networks (GNNs) (Scarselli et al., 2009) to learn fixed-dimensional, transferable representations of graph structures, which have been widely

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

adopted to enhance the encoding of graph-based observation spaces (Li et al., 2024). However, standard RL approaches typically rely on *discrete* action spaces, whose semantics and dimensionality are tightly coupled to individual graph instances. Consequently, learned action mappings are not readily transferable across graphs: the same discrete action identifier may refer to different graph entities (van Hasselt and Wiering, 2009; Chen et al., 2021), unless the policy is explicitly permutation-invariant and can infer the underlying semantics of actions from the input graph. Moreover, discrete action spaces limit scalability, as the output dimensionality of function approximators grows with the number of available actions (Dulac-Arnold et al., 2016).

To address this, recent GNN-based approaches introduce action embeddings derived from graph components and evaluate candidate actions via a Q-function $Q(s, a)$, as in `S2V-DQN` (Dai et al., 2017). However, iterating over all candidate actions incurs a cost that grows linearly with the action space, making runtime dependent on graph size and limiting scalability (Manchanda et al., 2020a). In particular, this computational cost is exacerbated in real-world GCO tasks with structured, high-dimensional decision variables, where the number of actions can grow super-linearly with graph size. A prominent example is traffic engineering, where decisions over graph paths induce an exponential explosion in the action space (Xiao et al., 2021). As a result, evaluating a Deep Neural Network (DNN) a number of times proportional to the action space size quickly becomes computationally prohibitive. This issue is amplified in dynamic settings, such as GCO applications in communication networks or data center infrastructures, where graph representations evolve continuously, and decisions must be re-computed repeatedly and under strict latency constraints (Vesselinova et al., 2020).

Contributions. To enhance the applicability of RL-GCO heuristics to large and diverse graphs across a broader range of tasks, we present the following key contributions:

(1) Projection agents for scalable and generalizable RL-GCO. We introduce *projection agents*, which operate directly in continuous GNN-based observation and action embedding spaces. Inspired by the Wolpertinger architecture (Dulac-Arnold et al., 2016), the agent predicts a desired action embedding in a single forward pass and projects it onto a valid discrete action through a Nearest Neighbor (NN) search. This unified approach achieves up to $6.41\times$ faster inference than iterative Q-value methods, while improving relative generalization by up to 75% on unseen instances.

(2) Unlocking RL-GCO for problems with complex and super-linear decision variables. We introduce a framework for constructing structured action embeddings via (i) semantic representations of individual decision components and (ii) composite embeddings capturing their interactions. This supports realistic super-linear decision spaces involving multiple interdependent variables, as found in modern real-world GCO tasks (e.g., traffic engineering, where selections involve paths and edges).

(3) Self-supervised representation learning framework to better compare RL-GCO. We leverage self-supervised GNN representations (Liu et al., 2022) to build aligned observation–action spaces for RL agents. By decoupling representation learning from policy learning, we enable fair comparison across RL methods within a shared embedding space.

(4) Unified Python library. We provide `LaGCO-RL`[1], a modular library for automated latent action-space construction in GCO, requiring only benchmark-specific environment logic to adapt to new problems. Code, checkpoints, scenarios, results, and configurations are publicly available.[2]

2 Related work

This section reviews related work along two main axes illustrated in Figure 1: *(i)* the action space complexity of existing benchmark GCO problems, and *(ii)* the ability of current RL-based solutions to effectively handle the typical action space challenges of GCO. An extended related work against other ML heuristic approaches (including other paradigms beyond RL) is provided in Appendix A.

Action space complexity gap. Prior work on RL-GCO has largely focused on simplified benchmarks with limited decision complexity. Seminal studies (Dai et al., 2017; Manchanda et al., 2020b; Li et al., 2018) consider classical problems such as the Traveling Salesman Problem (*TSP*) and

[1]Provided in the supplementary materials.

[2]Data (anonymized): `https://figshare.com/s/ecf06201726e5c2fd81f`

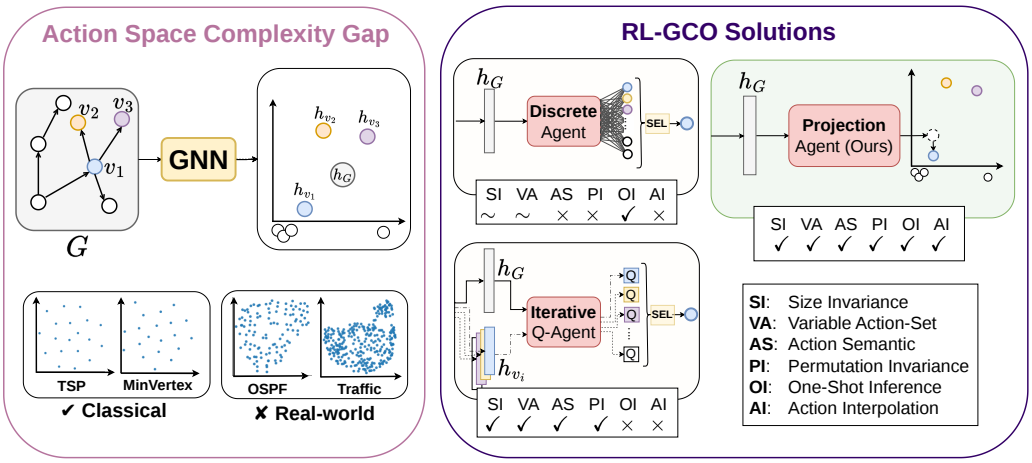


Figure 1: Overview of related work across: (1) action-space complexity, where real-world problems require richer decisions beyond node embeddings h_i; and (2) comparison of our projection method and others based on: *graph-size invariance*, support for *dynamic action sets*, *action semantics* (vs. flat indices), *permutation invariance*, *one-shot inference* (one forward pass for a decision), and support for *action interpolation*. ✓, ∼, and × denote full, partial, and no support, respectively.

Minimum Vertex Cover (*MinVertex*) (Dinur and Safra, 2005) (see Figure 1's action space instances coming from our experimental study), where graph representations remain basic: edge features are often reduced to scalar weights or ignored, and node features are limited to simple indicators (e.g., visited flags). Moreover, these approaches predominantly treat nodes as the sole decision variables, resulting in action spaces that scale linearly with graph size and exhibit relatively simple structure. Under such settings, GNN-based node embeddings are typically sufficient to produce well-separated manifolds of actions, facilitating efficient RL learning. In contrast, real-world applications such as *OSPF* engineering (Bernárdez et al., 2023) or *Traffic* routing (Xiao et al., 2021) involve more complex decision variables (e.g., paths), resulting in action spaces that grow super-linearly not only in size but also in structural complexity. This discrepancy highlights the gap in the resulting action space complexity between commonly used and real-world benchmarks.

RL-GCO solutions. Existing RL approaches fall short of key operational requirements for generalization and scalability in real-world GCO (Fig. 1). *Discrete* methods rely on padding-based action encodings to handle variable graph sizes, assigning one DNN output unit per action and treating graph elements as index-based entities (Terranova et al., 2025). This yields instance-specific policies that do not capture action semantics and transfer poorly across graphs. Such approaches provide only partial size invariance, limited by the padding dimension, and require invalid action masking for varying actions. While they also lack permutation invariance and interpolation, they remain computationally efficient due to a one-shot inference. On the other hand, *iterative* latent action-value methods embed both states and actions, typically using GNNs, and evaluate them sequentially via a learned Q-function (Dai et al., 2017; Manchanda et al., 2020a). This provides semantic representations, permutation and size invariance (from GNN features), and natural support for varying action sets (only valid actions can be iterated). However, this approach requires a separate forward pass per action, resulting in costs that scale with the action space, which itself can grow super-linearly with graph size in some benchmarks. Moreover, the lack of a continuous representation prevents action interpolation.

Adapting Wolpertinger to GCO. By contrast, our proposed *projection* approach adapts Wolpertinger (Dulac-Arnold et al., 2016) to predict a target action embedding in a latent space (created leveraging GNNs), with action selection delegated to a dedicated decoding strategy. It combines semantic embeddings with size and permutation invariance (via GNNs), naturally handles varying action sets (restricting decoding only to valid ones), and enables one-shot DNN inference followed by a fast decoding (e.g., NN search efficiently indexed for sub-linear retrieval). Unlike other methods, it also supports smooth interpolation across actions by projecting vectors over the full manifold.

Previous applications of Wolpertinger have primarily considered benchmarks such as discretized control, GridWorld, and recommendation systems (Dulac-Arnold et al., 2016; Chen et al., 2021; Chandak et al., 2019; Akkerman et al., 2024), where action embeddings remain fixed within an episode. In GCO, however, action representations are dynamic: as graph modifications propagate through the GNN, component embeddings evolve, so the same graph entity may have different action embeddings depending on preceding actions. To our knowledge, this is the first application of Wolpertinger to GCO and, more broadly, to a dynamic latent action space, requiring specialized methods for efficient learning.

3 METHODOLOGY

After a brief introduction of the problem formulation, this section introduces the framework for self-supervised learning, the construction of latent spaces, and the design of projection agents.

3.1 PROBLEM FORMULATION

We formulate GCO as a sequential learning problem aimed at constructing a solution on a graph $G = (V, E, V_f, E_f)$, where V and E denote the sets of nodes and edges, $V_f \in \mathbb{R}^{|V| \times d_v}$ encodes node features, and $E_f \in \mathbb{R}^{|E| \times d_e}$ encodes edge features (with $|V|$ and $|E|$ representing the cardinality of the node and edge sets, respectively, and d_v and d_e their corresponding feature dimensions). Let $\mathcal{C}(G)$ denote the decision space of admissible graph substructures (e.g., node subsets, edge subsets, paths, or induced subgraphs). This formulation generalizes beyond single-node decisions, enabling the selection of arbitrary substructures and thus yielding a richer and more challenging action space. In the autoregressive setting, a solution is constructed sequentially as $\mathbf{S} = (S_1, \ldots, S_T)$, where each decision $S_t \in \mathcal{C}(G)$ may depend on previous selections. The full sequence $\mathbf{S}$ defines a feasible solution to the GCO problem with objective

$$\min_{\mathbf{S} \in \mathcal{C}(G)^T} F(G, \mathbf{S}) \quad \text{s.t.} \quad H(G, \mathbf{S}) \leq 0, \tag{1}$$

where F measures the global cost and H encodes optional feasibility constraints. This formulation captures realistic GCO settings in which both F and H may depend on node features, edge features, and higher-order structural interactions. This autoregressive formulation naturally induces a Markov Decision Process (MDP) (Puterman, 1994) with a discrete action space:

- **Observation.** Each observation $o_t \in \mathcal{O}$ represents the intermediate solution at step t, defined as $o_t = (G, S_1, \ldots, S_{t-1})$.
- **Action.** An action $a_t \in \mathcal{A}(o_t) \subseteq \mathcal{C}(G)$ selects a valid graph substructure that extends the current partial solution.
- **Transition.** The transition deterministically appends the selected substructure to the solution while enforcing constraints H; the episode terminates when a solution is formed, or a cutoff is reached.
- **Reward.** The reward is derived from $F(G, \mathbf{S})$, so maximizing return is equivalent to minimizing the combinatorial objective, with penalties induced by constraint violations H.

For example, in *TSP*, the observation is a city graph with binary indicators of the current partial tour. The action selects an unvisited city, transitions update the tour, and the reward favors shorter tours.

3.2 NODE REPRESENTATION LEARNING

We pre-train a GNN encoder via self-supervised learning to produce node-level embeddings that serve as shared blocks for building both observations and actions of a given RL task. This design *(i)* decouples representation learning from action selection, enabling fair comparison across RL-GCO methods through a shared embedding space; *(ii)* improves sample efficiency by learning only the observation–action mapping with a lightweight controller (Ha and Schmidhuber, 2018), avoiding joint representation learning; and *(iii)* stabilizes training by keeping action embeddings stable, avoiding additional representational drift on top of the embedding changes induced by graph evolution.

We learn node-level representations using a Graph Auto-Encoder (GAE) (Kipf and Welling, 2016) strategy. A GNN encoder maps each node based on its features and neighborhood to a p-dimensional embedding, forming a matrix $\mathbf{Z} = h_\theta(G) \in \mathbb{R}^{|V| \times p}$. A feed-forward decoder g_ϕ reconstructs graph

information from $\mathbf{Z}$. The decoder is multi-headed, with separate heads for node features $x_i \in \mathbb{R}^{d_v}$, edge features $e_{ij} \in \mathbb{R}^{d_e}$ when present, and the adjacency matrix $A \in \mathbb{R}^{|V|\times|V|}$. The encoder–decoder pair is trained jointly by minimizing a weighted reconstruction loss over components:

$$\mathcal{L}(\theta,\phi) = \alpha \frac{1}{|V|}\sum_{i\in V}\mathcal{L}_{\text{node}}(\hat{\mathbf{x}}_i,\mathbf{x}_i) + \beta \frac{1}{|E|}\sum_{(i,j)\in E}\mathcal{L}_{\text{edge}}(\hat{\mathbf{e}}_{ij},\mathbf{e}_{ij}) + \gamma\,\mathcal{L}_{\text{adj}}(\hat{A},A), \tag{2}$$

where $\hat{\mathbf{x}}_i$, $\hat{\mathbf{e}}_{ij}$, and $\hat{A}$ are reconstructions produced by $g_\phi(\mathbf{Z})$, and α,β,γ control the contribution of each component. Node and edge features are decomposed by type (binary, categorical, continuous) to enable appropriate normalization and dedicated decoder heads with suitable loss functions.

We pretrain the GAE on trajectories from random valid actions, enabling maximal policy-independent exploration as the graph evolves. After convergence, we discard the decoder and retain the encoder for meaningful embeddings.

3.3 Continuous latent spaces

This subsection describes the methodology for constructing the observation and action spaces from the set of node embeddings $\mathbf{Z}$ introduced in Section 3.2. By sharing the same GNN representation, changes in graph observations induce semantically consistent shifts in action embeddings, enabling Wolpertinger to adapt to GCO's dynamic action space, where the embedding of a given action entity evolves with the graph state, as discussed in Section 2.

Observation space. We represent each observation as a graph G_t describing the current solution state. A fixed-dimensional observation vector o_t is obtained by applying K permutation-invariant graph aggregation operators $\mathcal{P}_k$ (e.g., sum, mean, or handcrafted graph statistics) over the node embeddings $\mathbf{Z}_t \in \mathbb{R}^{|V|\times p}$, and concatenating the resulting representations:

$$o_t = \Phi_{\text{obs}}(G_t) = \bigoplus_{k=1}^{K}\mathcal{P}_k(\mathbf{Z}_t), \tag{3}$$

where Φ_{obs} denotes the observation mapping function and $\bigoplus$ the concatenation operator. This yields a fixed-dimensional, order-invariant graph embedding that summarizes the agent's observation.

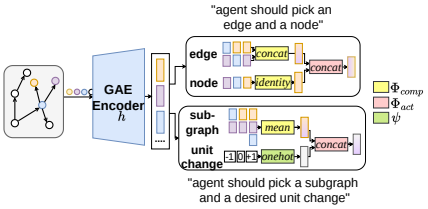


Figure 2: Action points' construction from node embeddings.

Action space. Actions may target graph components of varying granularity, including nodes, edges, paths, subgraphs, or combinations thereof (e.g., edge pairs). An action a is represented as a set of *components* $\mathcal{C}(a) = c_1,\ldots,c_m$, where each component corresponds to a subset of nodes $\mathcal{V}(c_i) \subseteq V$.

A component embedding is obtained by aggregating the embeddings of its nodes with a pooling Φ_{comp}:

$$h(c_i) = \Phi_{\text{comp}}(z_j \mid j \in \mathcal{V}(c_i)). \tag{4}$$

Depending on the application, this operator may be permutation-invariant (e.g., for subgraphs) or permutation-variant (e.g., directed edges). The action embedding is then obtained by combining embeddings of its components through an aggregation operator Φ_{act} (Fig. 2), which again can be permutation-invariant depending on whether component ordering is relevant:

$$\mathbf{u}(a) = \Phi_{\text{act}}(h(c_1),\ldots,h(c_m)), \quad c_i \in \mathcal{C}(a), \tag{5}$$

In many applications, actions involve not only graph components but also additional non-graph attributes α_i, such as discrete options or features associated with graph elements (e.g., in the cybersecurity scenario of Section 4, vulnerabilities are attached to a node and described by text). Depending on the setting, these attributes may either be independent or conditionally determined by previously selected components. These are encoded via a task-specific function $\psi(\cdot)$ and combined as:

$$\mathbf{u}(a) = \Phi_{\text{act}}\big(h(c_1),\ldots,h(c_m),\psi(\alpha_1 \mid c_1,\ldots,c_m),\ldots,\psi(\alpha_k \mid c_1,\ldots,\alpha_{k-1})\big). \tag{6}$$

This unified formulation enables heterogeneous actions typical of real-world applications by embedding both graph components and custom attributes in a shared space.

3.4 PROJECTION AGENT

The learned action embeddings enable the proposed projection agent by defining a bounded continuous latent action space over all possible action embeddings, onto which the agent projects its decisions. The latent action space $\mathbf{U} = \{\mathbf{u}(a_1), \ldots, \mathbf{u}(a_n)\}$ is integrated into the agent through a three-step pipeline: (1) optional preprocessing, such as normalization or dimensionality reduction, while preserving the geometric structure; (2) empirical estimation of the bounds of $\mathbf{U}$ across graph instances; and (3) alignment of its dimensionality and bounds with the agent's `gym.Box` action space.

Given an observation o_t, the policy π outputs a continuous proto-action in this space $\tilde{a}_t = \pi(o_t) \in \mathbf{U}$, where $\tilde{a}_t$ represents a "desired" target point in latent space rather than a valid discrete action. This proto-action is then decoded by retrieving its k nearest embeddings, forming a candidate set $\mathcal{A}_k(o_t)$. A final single action is selected from this set using a proper decoding strategy: e.g., simple NN (k=1), stochastic or rule-based selection, or a learned scoring function over the k-NN candidates. This results in a region-based learning signal in $\mathbf{U}$, where the decoding induces a space partition, and all points within a region map to a discrete action. As a result, the projection is piecewise constant, and the gradients act on the geometry of these regions rather than on individual actions.

4 EXPERIMENTS

Benchmarks. We evaluate our approach on seven GCO benchmarks. This includes three standard problems with node-level decisions and linearly growing action spaces—TSP, MinVertex, and Maximum Cut (*MaxCut*) (Bodlaender and Jansen, 2000); the latter is of particular importance, as more than half of the 21 Karp's canonical CO problems are reducible to MaxCut (Barrett et al., 2020). In addition, we consider four application-driven tasks drawn from delay-sensitive domains (networking and cybersecurity): Virtual Machine Placement (*Placement*) (Caviglione et al., 2021), Cyber-Attack Path Prediction (*Cyber-Path*) (Terranova et al., 2025), *OSPF*, and *Traffic*. Unlike classical benchmarks, these tasks involve structured actions combining graph and non-graph entities, while involving super-linear growth in the action space. Table 1 summarizes their action components, the expected growth of the action space, and the structured action representations used for each benchmark. Additional details on the logic of each environment, the episode structure, the score function, and the graph features are provided in Appendix B.

To generate scenarios, we perform an extended empirical sweep inspired by (Dai et al., 2017) (e.g., 10k sweeps for *TSP*) to approximate worst- and best-case GCO solutions via standard heuristic-guided exploration for each instance (see Appendix D for details). This produces a normalized reference score in $[0, 1]$, which mitigates scale effects (e.g., distance-dependent costs in *TSP*), enables per-step relative rewards to alleviate the challenges of sparse episodic rewards (Barrett et al., 2020) (see Appendix B), and ensures comparability and aggregation across instance scores.

Observation: Figure 3 shows 2D UMAP (Healy and McInnes, 2024) projections of action spaces for 20-node benchmarks (more in Appendix E). We quantify complexity via compactness, $\mathrm{cmp}@K$, the ratio of mean pairwise to k-NN distance ($k \in \{3, 5, 10\}$), averaged over 20 instances. Real-world benchmarks exhibit higher cmp, indicating denser, more overlapping neighborhoods.

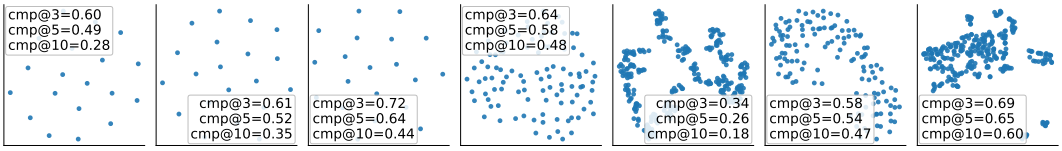
Our library, `LaGCO-RL`, provides a modular interface for integrating GCO environments with diverse action structures and problem formulations, enabling new benchmarks beyond these, without modifying the core learning framework (Appendix H).

Solutions. We compare the three policy parameterizations introduced in Section 2: *Discrete*, *Iterative*, and *Projection*. The discrete baselines use two observation encodings: padding (P-, permutation-variant) and GNN-based pooling (G-, permutation-invariant), each evaluated with and without action masking (-M representing "with"). We implement these baselines using `PPO` and `MaskablePPO` from SB3, respectively. Our projection method uses `PPO` with z-score-normalized actions and (K=1)-NN decoding for a high-precision setting. The iterative method extends SB3's `DQN` via fitted Q-iteration. Both latent-space methods restrict decisions to valid actions only.

We also distinguish between the *non-revisiting* and *revisiting* interaction strategies as orthogonal axes. By default, we follow the non-revisiting strategy inspired by `S2V-DQN` Dai et al. (2017), where each selected action constrains the remaining action space. For example, in *Placement*, assigning a VM to

Table 1: Action-space structure across benchmarks, including worst-case growth and the chosen representation function (h denotes the node embedding function).

Benchmark	Action Components	Worst-Case Growth	Representation ($h : \mathcal{V} \to \mathbb{R}^{16}$)
TSP	node v (not visited)	$O(\|V\|)$	$h(v)$
MinVertex	node v	$O(\|V\|)$	$h(v)$
MaxCut	node v	$O(\|V\|)$	$h(v)$
Placement	source node u (type=VM) target node v (type=PM)	$O(\|V\|^2)$	concat$(h(u), h(v))$
Cyber-Path	source node u (status=owned) target node v (status=discovered) vuln $\alpha \in v$ (not yet exploited)	$O(k\|V\|^2)$	concat$(h(u), h(v), \texttt{BERT}(\alpha_{\text{text}}))$
OSPF	edge (u, v) weight value $w \in \{1, ..., N\}$	$O(k\|V\|^2)$	concat$(h(u), h(v), \texttt{onehot}(w))$
Traffic	edge (u, v) (type=traffic) path $(u_1, \ldots, u_n)$ (type=link)	$O(\|V\|^2 2^{\|V\|})$	pad(concat$(h(u), h(v), ..., h(u_n))$)



(a) TSP (b) MinVertex (c) MaxCut (d) Placement (e) Cyber-Path (f) OSPF (g) Traffic

Figure 3: UMAP projection of benchmark action spaces for a 20-node graph instance, illustrating local geometry and neighborhood proximity with the `cmp@k` ($k \in 3, 5, 10$) (computed on 20 instances).

a PM makes all other actions involving that VM allocation invalid, progressively shrinking the set of feasible actions and yielding a one-shot construction process. We additionally evaluate a *revisiting* strategy inspired by `ECO-DQN` Barrett et al. (2020), which allows previously selected actions to be reconsidered (e.g., reassigning a previously placed VM), rather than removing action points conditional to prior selections. Since revisitation is only meaningful for non-sequential problems, other benchmarks (TSP and Cyber-Path) are evaluated exclusively under the non-revisiting strategy. Details of the revisiting procedure are provided in Appendix B.

Observation: We focus our main comparisons on RL-based parameterizations within a shared methodology, ensuring a consistent autoregressive and data-driven setting. We therefore exclude supervised and unsupervised methods that rely on pre-existing labels or do not enforce the feasibility constraints required by our benchmarks (Appendix A). Since optimal solvers are not uniformly available across benchmarks, we instead use a heuristic-guided empirical reference for min-max score normalization in [0,1] (details in Appendix B).

Solution-specific hyperparameters, along with those that govern the GAE and its architecture, are optimized using the Tree-structured Parzen Estimator (TPE) (Watanabe, 2025) for 25 trials each. The objective is task-specific performance evaluated on a held-out validation set, considering only the *TSP* benchmark for simplicity. Complete search spaces, controller details, seeds, and hyperparameters are provided in Appendix C, with observation-space design in Appendix I.

4.1 How well do RL-GCO approaches generalize across graph distributions?

Experimental design. For each benchmark, we generate 101 diverse instances by sampling key parameters from typical benchmark distributions to evaluate out-of-distribution generalization (Kumar et al., 2020) across instance sizes and training configurations. Scenarios are then ordered by size and evaluated under four transfer-oriented training strategies. In the first three, agents train on a single scenario based on their size—*smallest* (S), *medium* (M), or *largest* (L)—and are tested on all remaining instances; each strategy is repeated $K = 5$ times with different seeds and scenarios

Table 2: IQM of normalized generalization scores across benchmarks and training regimes (S/M/L/V) under the non-revisiting strategy. Δ denotes the average train–test generalization gap. **Underlined bold** indicates the best method per benchmark.

	TSP					MinVertex					MaxCut				
Method	S	M	L	V	Δ	S	M	L	V	Δ	S	M	L	V	Δ
P-Discrete	0.00	0.00	0.00	0.00	+0.00	0.00	0.00	0.00	0.16	-0.62	0.91	0.43	0.06	0.87	+0.00
P-Discrete-M	0.50	0.47	0.47	0.48	-0.05	0.39	0.36	0.39	0.39	-0.44	0.91	0.89	0.88	0.87	-0.02
G-Discrete	0.00	0.00	0.00	0.00	-0.05	0.00	0.00	0.13	0.03	-0.28	0.07	0.89	0.90	0.88	-0.24
G-Discrete-M	0.51	0.50	0.50	0.50	-0.19	0.32	0.35	0.36	0.39	-0.44	0.07	0.90	0.91	0.90	-0.23
Iterative	0.90	0.79	0.97	**0.99**	-0.04	0.40	0.05	0.00	0.16	-0.19	0.91	0.23	0.33	0.21	-0.15
Projection (ours)	0.78	0.76	0.60	0.71	-0.13	**0.70**	0.01	0.03	0.02	-0.18	0.93	**0.94**	0.91	0.93	-0.03

	Placement					Cyber-Path					OSPF					Traffic				
Method	S	M	L	V	Δ	S	M	L	V	Δ	S	M	L	V	Δ	S	M	L	V	Δ
P-Discrete	0.67	0.58	0.58	0.60	-0.36	0.19	0.18	0.19	0.18	-0.05	0.08	0.11	0.03	0.00	-0.33	0.11	0.48	0.16	0.37	-0.35
P-Discrete-M	0.72	0.74	0.70	0.71	-0.30	0.61	0.63	0.61	**0.67**	-0.20	0.86	0.85	0.84	0.81	-0.04	0.81	0.79	0.80	0.81	-0.13
G-Discrete	0.42	0.58	0.66	0.69	-0.33	0.18	0.19	0.18	0.17	-0.05	0.36	0.20	0.19	0.59	-0.55	0.24	0.57	0.49	0.58	-0.42
G-Discrete-M	**0.79**	0.76	0.74	0.72	-0.24	0.45	0.45	0.51	0.50	-0.28	0.51	0.58	0.74	0.76	-0.28	0.69	0.72	0.78	0.77	-0.16
Iterative	0.26	0.02	0.21	0.29	-0.19	0.34	0.24	0.19	0.24	-0.22	0.93	0.93	0.91	0.92	+0.00	0.66	0.54	0.49	0.61	-0.10
Projection (ours)	**0.79**	0.78	0.76	**0.79**	-0.12	0.60	0.64	0.60	**0.67**	-0.17	**0.95**	0.92	0.89	0.90	-0.03	0.79	**0.85**	0.83	0.84	-0.09

Table 3: Effect of revisiting (R) vs. non-revisiting (NR) interaction on revisitable tasks.

	MinVertex				Maxcut				Placement				OSPF				Traffic			
	S	M	L	V	S	M	L	V	S	M	L	V	S	M	L	V	S	M	L	V
P-Discrete-M (NR)	0.39	0.36	0.39	0.39	0.91	0.89	0.88	0.87	0.72	0.74	0.70	0.71	0.86	0.85	0.84	0.81	0.81	0.79	0.80	0.81
P-Discrete-M (R)	0.00	0.00	0.00	0.00	0.91	0.47	0.80	0.17	0.61	0.64	0.66	0.65	0.25	0.22	0.70	0.01	0.36	0.74	0.77	0.73
Iterative (NR)	0.40	0.05	0.00	0.16	0.91	0.23	0.33	0.21	0.26	0.02	0.21	0.29	0.93	0.93	0.91	0.92	0.66	0.54	0.49	0.61
Iterative (R)	0.13	0.01	0.01	0.00	0.69	0.86	0.78	0.86	0.55	0.02	0.07	0.15	0.00	0.04	0.00	0.02	0.01	0.01	0.00	0.01
Projection (NR)	**0.70**	0.01	0.03	0.02	0.93	**0.94**	0.91	0.93	0.79	0.78	0.76	0.79	**0.95**	0.92	0.89	0.90	0.79	**0.85**	0.83	0.84
Projection (R)	0.13	0.38	0.07	0.07	0.84	0.91	0.91	0.93	0.87	0.87	0.86	**0.92**	0.67	0.79	0.84	0.88	0.80	0.82	0.81	0.81

(removing those already selected from the selection pool). The fourth, *varied* (V), adopts a K-fold-inspired approach: agents are trained on five non-overlapping subsets of scenarios (20%), with training instances rotated every 100 episodes. For each run, performance is assessed over 5 episodes per unseen instance (80% for V, 100 instances for S, M, and L), with the best result retained; aggregated outcomes across 20 runs per solution–benchmark pair are reported in Table 2. We report the Interquartile Mean (IQM) over these 100 best scores, following the guidelines on suitable RL indicators from Agarwal et al. (2021). Additionally, the Δ columns report the test-to-training gap (averaged across strategies) for each method. Training time, learning curves, extended generalization results, and additional metrics (including evaluation without best-per-environment selection) are provided in Appendices K, J, and M. Appendix F compares GAE loss variants against plain graph features and random embeddings, assessing the contribution of the learned graph representations.

Analysis. Table 2 reports results under the non-revisiting strategy, aiming to compare the different policy parameterization strategies. The iterative approach performs particularly well on TSP, matching 10k-sweep heuristic optima with only five trials per instance. This demonstrates that RL can match strong heuristic baselines under a highly limited evaluation budget. On the remaining benchmarks, our projection method achieves the best (or tied-best) performance, with particularly pronounced gains on real-world problems with larger and more complex action spaces. The two approaches face different scaling effects. As action-space dimensionality increases, the projection agent benefits from a richer latent space for representing the target action before decoding. In contrast, the iterative approach becomes less effective as the candidate action set grows, since $\arg\max$ selection requires accurately ranking an increasing number of actions, making it more sensitive to Q-value estimation errors. The masked discrete approach remains more robust by restricting selection to valid actions, achieving the second-best overall performance.

Table 3 further examines the effect of the interaction strategy by evaluating the different policy parameterizations (using the best discrete variant) under both non-revisiting (NR) and revisiting (R) strategies. This comparison evaluates whether RL agents can exploit an unrestricted action space that allows revisiting previous decisions, at the cost of more actions per step. Revisitation substantially

degrades the discrete and iterative approaches, whereas projection remains comparatively more stable, suggesting greater robustness to larger action spaces. In terms of solution quality, revisiting provides limited benefits, improving *Placement* while yielding similar or slightly worse results elsewhere.

We additionally demonstrate competitive generalization against classical, unsupervised, and RL-based solvers in Appendix G, including MaxCut, where we achieve a $+0.03$ mean score over physics-informed GNNs Schuetz et al. (2022) and 95% of simulated annealing performance.

4.2 How does inference time scale with graph size?

Experimental design. We evaluate computational scalability by analyzing how action-selection time scales with graph size across three approaches: the proposed GNN-based *projection* method, which uses a FAISS Flat Index for NN lookup with linear retrieval complexity (Johnson et al., 2019); the *discrete* baseline, which requires only a single DNN inference without additional computation; and the *iterative* method, which repeatedly evaluates the DNN. All experiments are conducted on a benchmark machine.[3] Specifically, we fit a power-law model $T(n) = c \cdot n^{\alpha}$ where $T(n)$ is the median action-selection time for a graph of size n, and α is the scaling exponent.

Analysis. Table 4 reports the scaling exponent α and corresponding R^2 values. As expected, *discrete* scales best due to its single DNN evaluation, while *projection* substantially improves over *iterative* by replacing per-action DNN evaluations with a single inference and NN search. This advantage is strongest in delay-sensitive benchmarks with superlinear action spaces; Figure 4 shows TSP and Placement, with additional scaling and approximate-NN results in Appendices L and L.1.

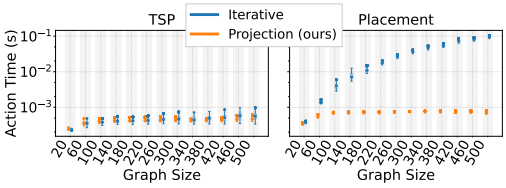


Figure 4: Evolution of inference-time action-selection latency across graph sizes for the representative TSP and Placement benchmarks.

	P-Discrete		Projection (ours)		Iterative	
Env	α	R^2	α	R^2	α	R^2
TSP	0.07	0.89	**0.17**	0.85	0.25	0.94
MinVertex	0.16	0.69	**0.30**	0.88	0.32	0.95
MaxCut	0.03	0.78	**0.22**	0.94	0.25	0.92
Placement	0.09	0.85	**0.22**	0.85	1.81	0.99
Cyber-Path	0.09	0.75	**0.17**	0.73	1.46	0.95
OSPF	0.20	0.76	**0.49**	0.94	1.44	0.99
Traffic	0.05	0.70	**0.73**	0.71	4.68	0.99

Table 4: Scaling of action-selection time with fitted power-law exponent α. Lower α indicates better scaling with graph size.

5 Discussion

Conclusion. We introduce an end-to-end framework for latent-space RL-GCO and a Wolpertinger adaptation that improves generalization and scalability across parameterizations. We further propose realistic benchmarks with structured action spaces and demonstrate consistent gains for our solution.

Limitations. Our methodology has several limitations. (1) Encoding the evolving partial solution as state-dependent node features requires repeated GNN recomputation; future work could decouple graph structure from solution state to reduce this overhead. (2) We use a simple NN decoder, deliberately chosen as a conservative worst-case setting, though learned or heuristic ranking could improve performance. (3) We use a single encoding per action-space setting, leaving richer multi-encoding representations for future work. (4) Self-supervised embeddings are not jointly fine-tuned with the RL policy, enabling fair comparisons but potentially limiting performance.

AI Use Disclosure

We used generative AI (GPT-5.4) in two aspects of this work. First, we used it to improve the readability of the manuscript, including polishing and rewriting sentences in a more concise form. Second, we used it to generate heuristic strategies to guide the normalization procedure. The generated

[3] *Benchmark Machine:* AMD Ryzen 7 PRO 7840U w/ Radeon 780M Graphics, 30 GiB RAM, Ubuntu 22.04.4.

heuristics were subsequently reviewed, refined, and manually validated by the authors. We take responsibility for the final content, methodology, and claims presented in this work.

Reproducibility Statement

We release the *LaGCO-RL* library together with the code and configuration needed to reproduce the experiments presented in the main paper. The accompanying `README` provides general information on the library, its installation, and usage, while the `REPRODUCIBILITY` document provides the exact commands and procedures required to reproduce the main-body experiments. Appendix H describes the library and its interfaces in detail. In addition, the accompanying data repository contains the datasets, trained model checkpoints, experiment logs, and configurations used for all experiments reported in this paper, enabling independent verification of the reported results.

References

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NeurIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc. ISBN 9781713845393. doi: 10.48550/arXiv.2108.13264.

Fabian Akkerman, Julius Luy, Wouter van Heeswijk, and Maximilian Schiffer. Dynamic neighborhood construction for structured large discrete action spaces. In *The Twelfth International Conference on Learning Representations*, 2024.

Thomas D. Barrett, William R. Clements, Jakob N. Foerster, and Alex I. Lvovsky. Exploratory combinatorial optimization with reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 34, pages 3243–3250. Association for the Advancement of Artificial Intelligence, 2020. doi: 10.1609/aaai.v34i04.5723.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021. ISSN 0377-2217. doi: 10.1016/j.ejor.2020.07.063.

Guillermo Bernárdez, José Suárez-Varela, Albert López, Xiang Shi, Shihan Xiao, Xiangle Cheng, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Magnneto: A graph neural network-based multi-agent system for traffic engineering. *IEEE Transactions on Cognitive Communications and Networking*, 9(2):494–506, 2023. doi: 10.1109/TCCN.2023.3235719.

Federico Berto, Chuanbo Hua, Junyoung Park, Laurin Luttmann, Yining Ma, Fanchen Bu, Jiarui Wang, Haoran Ye, Minsu Kim, Sanghyeok Choi, Nayeli Gast Zepeda, André Hottung, Jianan Zhou, Jieyi Bi, Yu Hu, Fei Liu, Hyeonah Kim, Jiwoo Son, Haeyeon Kim, Davide Angioni, Wouter Kool, Zhiguang Cao, Jie Zhang, Kijung Shin, Cathy Wu, Sungsoo Ahn, Guojie Song, Changhyun Kwon, Lin Xie, and Jinkyoo Park. RL4CO: an Extensive Reinforcement Learning for Combinatorial Optimization Benchmark. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2025. URL `https://github.com/ai4co/rl4co`.

Hans L. Bodlaender and Klaus Jansen. On the complexity of the maximum cut problem. *Nordic J. of Computing*, 7(1):14–31, March 2000. ISSN 1236-6064. doi: 10.5555/640044.640046.

Quentin Cappart, Didier Chételat, Elias B. Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4348–4355. International Joint Conferences on Artificial Intelligence Organization, 2021a. doi: 10.24963/ijcai.2021/595. Survey Track.

Quentin Cappart, Thierry Moisan, Louis-Martin Rousseau, Isabeau Prémont-Schwarz, and Andre A. Cire. Combining reinforcement learning and constraint programming for combinatorial optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(5):3677–3687, May 2021b. doi: 10.1609/aaai.v35i5.16484.

Luca Caviglione, Mauro Gaggero, Massimo Paolucci, and Roberto Ronco. Deep reinforcement learning for multi-objective placement of virtual machines in cloud datacenters. *Soft Comput.*, 25 (19):12569–12588, October 2021. ISSN 1432-7643. doi: 10.1007/s00500-020-05462-x.

Yash Chandak, Georgios Theocharous, James Kostas, Scott Jordan, and Philip S. Thomas. Learning action representations for reinforcement learning, 2019. URL https://arxiv.org/abs/1902.00183.

Yu Chen, Yingfeng Chen, Zhipeng Hu, Tianpei Yang, Changjie Fan, Yang Yu, and Jianye Hao. Learning action-transferable policy with action embedding. 2021. doi: 10.48550/arXiv.1909.02291.

Hanjun Dai, Elias B. Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NeurIPS'17, page 6351–6361, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964. doi: 10.48550/arXiv.1704.01665.

Victor-Alexandru Darvariu, Stephen Hailes, and Mirco Musolesi. Graph reinforcement learning for combinatorial optimization: A survey and unifying perspective. 2024. doi: 10.48550/arXiv.2404.06492.

Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of mathematics*, pages 439–485, 2005. doi: 10.4007/annals.2005.162.439.

Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. 2016. doi: 10.48550/arXiv.1512.07679.

Andoni I. Garmendia, Josu Ceberio, and Alexander Mendiburu. Neural improvement heuristics for graph combinatorial optimization problems. *IEEE Transactions on Neural Networks and Learning Systems*, 35(12):18300–18312, 2024. doi: 10.1109/TNNLS.2023.3314375.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. *Exact combinatorial optimization with graph convolutional neural networks*. Curran Associates Inc., Red Hook, NY, USA, 2019. doi: 10.5555/3454287.3455683.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. doi: 10.48550/arXiv.1809.01999.

John Healy and Leland McInnes. Uniform manifold approximation and projection. *Nature Reviews Methods Primers*, 4(1):82, 2024. doi: 10.21105/joss.00861.

Nasimeh Heydaribeni, Xinrui Zhan, Ruisi Zhang, Tina Eliassi-Rad, and Farinaz Koushanfar. Distributed constrained combinatorial optimization leveraging hypergraph neural networks. *Nature Machine Intelligence*, 6(6):664–672, 2024. ISSN 2522-5839. doi: 10.1038/s42256-024-00833-7.

Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, 50(2), April 2017. ISSN 0360-0300. doi: 10.1145/3054912.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019. doi: 10.1109/TBDATA.2019.2921572.

Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. Chapter 4 the traveling salesman problem. In *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 225–330. Elsevier, 1995. doi: 10.1016/S0927-0507(05)80121-5.

Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NeurIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546. doi: 10.48550/arXiv.2006.10643.

Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972. ISBN 978-1-4684-2001-2. doi: 10.1007/978-1-4684-2001-2_9.

Elias B. Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 724–731. AAAI Press, 2016. doi: 10.5555/3015812.3015920.

Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016. doi: 10.48550/arXiv.1611.07308.

Saurabh Kumar, Aviral Kumar, Sergey Levine, and Chelsea Finn. One solution is not all you need: Few-shot extrapolation via structured maxent rl. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 8198–8210. Curran Associates, Inc., 2020. doi: 10.48550/arXiv.2010.14484.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: policy optimization with multiple optima for reinforcement learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Kaiwen Li, Tao Zhang, Rui Wang, Yuheng Wang, Yi Han, and Ling Wang. Deep reinforcement learning for combinatorial optimization: Covering salesman problems. *IEEE Transactions on Cybernetics*, 52(12):13142–13155, 2022. doi: 10.1109/TCYB.2021.3103811.

Zhi-Peng Li, Si-Guo Wang, Qin-Hu Zhang, Yi-Jie Pan, Nai-An Xiao, Jia-Yang Guo, Chang-An Yuan, Wen-Jian Liu, and De-Shuang Huang. Graph pooling for graph-level representation learning: a survey. *Artificial Intelligence Review*, 58(2):45, 2024. doi: 10.1007/s10462-024-10949-2.

Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NeurIPS'18, page 537–546, Red Hook, NY, USA, 2018. Curran Associates Inc. doi: 10.5555/3326943.3326993.

Yixin Liu, Yu Zheng, Daokun Zhang, Hongxu Chen, Hao Peng, and Shirui Pan. Towards unsupervised deep graph structure learning. In *Proceedings of the ACM Web Conference 2022*, WWW '22, page 1392–1403, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450390965. doi: 10.1145/3485447.3512186.

Sahil Manchanda, Akash Mittal, Anuj Dhawan, Sourav Medya, Sayan Ranu, and Ambuj Singh. Gcomb: Learning budget-constrained combinatorial algorithms over billion-sized graphs. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20000–20011. Curran Associates, Inc., 2020a. doi: 10.5555/3495724.3497403.

Sahil Manchanda, Akash Mittal, Anuj Dhawan, Sourav Medya, Sayan Ranu, and Ambuj Singh. Learning heuristics over large graphs via deep reinforcement learning. 2020b. doi: 10.48550/arXiv.1903.03332.

Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021. ISSN 0305-0548. doi: 10.1016/j.cor.2021.105400.

Yifan Peng, Byungki Choi, and Jun Xu. Graph learning for combinatorial optimization: A survey of state-of-the-art. *Data Science and Engineering*, 6:119–141, 2021. doi: 10.1007/s41019-021-00155-3.

Martin L. Puterman. Markov decision processes: Discrete stochastic dynamic programming. USA, 1994. John Wiley & Sons, Inc. ISBN 0471619779. doi: 10.5555/528623.

Sebastian Sanokowski, Wilhelm Berghammer, Sepp Hochreiter, and Sebastian Lehner. Variational annealing on graphs for combinatorial optimization. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NeurIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc. doi: 10.5555/3666122.3668914.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

Martin J. A. Schuetz, J. Kyle Brubaker, and Helmut G. Katzgraber. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4(4):367–377, 2022. ISSN 2522-5839. doi: 10.1038/s42256-022-00468-6.

Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 3706–3731. Curran Associates, Inc., 2023. doi: 10.5555/3666122.3666286.

Franco Terranova, Abdelkader Lahmadi, and Isabelle Chrisment. Scalable and generalizable rl agents for attack path discovery via continuous invariant spaces. In *2025 28th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, pages 440–457, 2025. doi: 10.1109/RAID67961.2025.00029.

Hado van Hasselt and Marco A. Wiering. Using continuous action spaces to solve discrete problems. In *2009 International Joint Conference on Neural Networks*, pages 1149–1156, 2009. doi: 10.1109/IJCNN.2009.5178745.

Natalia Vesselinova, Rebecca Steinert, Daniel F. Perez-Ramirez, and Magnus Boman. Learning combinatorial optimization on graphs: A survey with applications to networking. *IEEE Access*, 8: 120388–120416, 2020. doi: 10.1109/ACCESS.2020.3004964.

Haoyu Wang, Nan Wu, Hang Yang, Cong Hao, and Pan Li. Unsupervised learning for combinatorial optimization with principled objective relaxation. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NeurIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088. doi: 10.5555/3600270.3602550.

Runzhong Wang, Zhigang Hua, Gan Liu, Jiayi Zhang, Junchi Yan, Feng Qi, Shuang Yang, Jun Zhou, and Xiaokang Yang. A bi-level framework for learning to solve combinatorial optimization on graphs. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NeurIPS '21, Red Hook, NY, USA, 2021a. Curran Associates Inc. ISBN 9781713845393. doi: 10.5555/3540261.3541902.

Runzhong Wang, Tianqi Zhang, Tianshu Yu, Junchi Yan, and Xiaokang Yang. Combinatorial learning of graph edit distance via dynamic embedding. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2021b.

Shuhei Watanabe. Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance. 2025. doi: 10.48550/arXiv.2304.11127.

Yang Xiao, Jun Liu, Jiawei Wu, and Nirwan Ansari. Leveraging deep reinforcement learning for traffic engineering: A survey. *IEEE Communications Surveys & Tutorials*, 23(4):2064–2097, 2021. doi: 10.1109/COMST.2021.3102580.

Ming Zhu and Zhiqiang Pu. Parallel environments for combinatorial optimization problems using reinforcement learning. In *2025 IEEE 12th International Conference on Cyber Security and Cloud Computing (CSCloud)*, pages 367–369, 2025. doi: 10.1109/CSCloud66326.2025.00070.

A RELATED WORK: BEYOND RL SOLUTIONS

Several ML paradigms have been proposed for data-driven heuristic discovery in GCO. This section reviews the most relevant approaches according to their underlying learning paradigm.

Supervised learning. Supervised approaches have been extensively studied in the literature, with recent advances including end-to-end methods that directly generate complete solutions—for example, diffusion-based models (Sun and Yang, 2023)—as well as neural improvement heuristics (Garmendia et al., 2024). The latter departs from full solution construction by adopting an iterative refinement strategy, where existing solutions are progressively improved through local decisions. This shift simplifies the learning problem while still achieving competitive performance.

However, these methods inherently rely on labeled solutions, which are often costly or infeasible to obtain with exact methods for large-scale instances (Wang et al., 2021a). As a consequence, in practice, labels are typically generated using existing manually defined heuristics, introducing an inherent performance ceiling: models are trained to mimic potentially suboptimal solutions and cannot surpass this upper bound (Cappart et al., 2021a; Sanokowski et al., 2023).

Unsupervised learning. Unsupervised methods, similar to RL, do not require labeled data and instead optimize the problem objective directly (Peng et al., 2021). A common strategy is to relax these discrete optimization problems into continuous formulations via concave surrogate losses, which are optimized in an unsupervised manner (Sanokowski et al., 2023; Wang et al., 2022). These approaches have been explored using various architectures, including GNNs (Karalias and Loukas, 2020), hypergraph neural networks (Heydaribeni et al., 2024), physics-informed neural networks (Schuetz et al., 2022), and variational annealing (Sanokowski et al., 2023).

Despite their promise, unsupervised approaches face several challenges. Their generalization capabilities remain insufficiently explored, complex constraints are difficult to incorporate (Wang et al., 2021a), and models may converge to suboptimal local minima (Heydaribeni et al., 2024). Additionally, reliance on continuous relaxations introduces approximation gaps that can degrade solution quality and complicate optimization (Cappart et al., 2021a).

Reinforcement learning. In contrast to other approaches, RL optimizes the objective through sequential decision-making, relying on step-wise evaluative feedback rather than fixed supervision over entire trajectories, as in supervised learning or imitation learning (Hussein et al., 2017). Instead of using static labels for complete solutions, RL leverages reward signals to assess the quality of actions over time. This framework enables agents to go beyond the limitations imposed by predefined datasets and known reference solutions.

However, the key challenge of RL lies in the inherently sparse reward structure of GCO problems, where meaningful evaluations are typically only available at the end of an episode, making training difficult (Sanokowski et al., 2023; Wang et al., 2021a). Prior work has addressed this limitation by introducing intermediate reward signals to alleviate sparsity and improve learning efficiency (Barrett et al., 2020). A similar idea is adopted in this work, which leverages the worst- and best-known solutions to compute step-wise improvement signals, thereby providing a denser reward for more effective learning.

Hybrid approaches. Hybrid methods have been proposed to combine learning-based and classical optimization techniques for solving GCO problems. In learning-assisted optimization, several works have proposed neural models able to guide traditional solvers, such as learning heuristics for A* search (Wang et al., 2021b) or branching strategies in branch-and-bound (Gasse et al., 2019; Khalil et al., 2016). Similar ideas have also been explored using RL (Cappart et al., 2021b). Another study proposed a bi-level framework in which RL reduces the problem space before applying a fast heuristic for refinement (Wang et al., 2021a).

More recently, hybrid learning approaches combine multiple ML paradigms to further improve the learning-based solution. For instance, supervised learning can be used to prune the search space explored by RL policies (Manchanda et al., 2020a), improving scalability to large graphs. However, such methods reintroduce the limitations of supervised learning, as their effectiveness depends on

the quality of the initial labels. Nevertheless, these approaches remain complementary and can be naturally integrated into all RL-based frameworks, including the one proposed in this work.

B BENCHMARK ENVIRONMENTS

This section describes the benchmark environments used in the experimental study (Section 4), focusing on their logical structure and dynamics, graph representations, and the scoring criteria used to evaluate solutions. During training, the best-known score is additionally employed as a termination criterion to accelerate convergence. This mechanism is removed at test time to avoid the potential drawbacks associated with access to supervised solutions, as discussed in Appendix A.

B.1 TRAVELING SALESMAN PROBLEM

Logic: Construct a minimum-length tour that visits each city exactly once and returns to the starting city (hard constraint).

- **Parameters:** Number of cities N and coordinate range defining the (x, y) positions of the cities.
- **Graph:** Nodes represent cities, and edges encode distances between cities. The underlying graph is fully connected, but is sparsified by retaining the $K = 10$ nearest neighbors per node to avoid a fully connected graph and the resulting issues with GNN embeddings.
- **Node features:** A binary visited flag $\in \{0, 1\}$ indicating whether a city has already been included in the tour, together with its spatial coordinates (x, y).
- **Edge features:** Euclidean distances between cities.
- **Reset:** A starting city is randomly selected and marked as visited, while all remaining cities are initialized as unvisited. The current city is set to the selected starting city and the tour is initialized from this node.
- **Step helper:** At each step, the agent selects the next city to visit. The selected city is appended to the current tour, marked as visited, and removed from the action space. The distance from the current city to the selected city is added to the accumulated tour length, and the selected city becomes the new current city.
- **Objective and reward:** The objective is to minimize the total tour length, given by the sum of the Euclidean distances along the constructed tour, including the final return to the starting city. The reward is normalized as a relative score using the empirically known best and worst tour lengths. When the tour is incomplete, the remaining cities are incorporated using a worst-case completion heuristic to estimate the final tour cost, encouraging the agent to make decisions that also facilitate completion of the tour.
- **Termination:** The episode terminates once all N cities have been visited. For a complete tour, the distance from the final city back to the starting city is then added to obtain the total tour length.
- **Score:** The score is based on the constructed tour length, normalized relative to the empirically known best and worst solutions.
- **Revisiting strategy:** Not supported by this sequential benchmark, as each action permanently extends the tour and consumes the selected city from the available action space.

B.2 MINIMUM VERTEX COVER

Logic: Select the smallest subset of nodes such that every edge is incident to at least one selected node (hard constraint).

- **Parameters:** Number of nodes N and edge probability p.
- **Graph:** An Erdős–Rényi random graph. A solution is a subset of nodes whose selected nodes collectively cover every edge.
- **Node features:** A binary selected flag $\in \{0, 1\}$ indicating whether a node currently belongs to the cover.

- **Edge features:** A binary covered flag $\in \{0, 1\}$ indicating whether an edge is currently covered by at least one of its incident selected nodes.
- **Reset:** All nodes are initialized as unselected and all edges as uncovered.
- **Step helper (non-revisiting):** Selecting an unselected node adds it permanently to the cover, removes it from the action space, and marks all newly covered incident edges as covered.
- **Step helper (revisiting):** When revisiting is enabled, an action can be applied to any node, including nodes that have already been selected. The action *toggles* the node state: selecting an unselected node adds it to the cover, while selecting an already selected node removes it from the cover.
- **Objective and reward:** At each step, the objective value is computed both before and after the node toggle. The reward is defined as the difference in the size of the cover relative to the previous timestep. Consequently, a positive reward indicates an improvement in the current solution, whereas a negative reward indicates a deterioration.
- **Termination:** In the constructive setting, the episode terminates as soon as all edges are covered, since a feasible vertex cover has been constructed, or when a predefined cutoff is reached. In the revisiting setting, the agent may continue toggling nodes to search for a smaller feasible cover. Therefore, termination is controlled only by the predefined cutoff.
- **Score:** The score is zero if no feasible cover is found; otherwise, the final score is based on the size of the best feasible cover found, normalized relative to the empirically known best and worst solutions.

B.3 Maximum Cut

Logic: Partition nodes into two sets to maximize the total weight of edges crossing between partitions.

- **Parameters:** Number of nodes N and edge weight range.
- **Graph:** A weighted graph with edge weights randomly sampled across node pairs. The graph is sparsified by retaining $K = 10$ NNs per node to avoid a fully connected graph.
- **Node features:** A binary partition assignment $\in \{0, 1\}$ indicating the set to which each node belongs.
- **Edge features:** Edge weight as a scalar value.
- **Reset:** All nodes are initially assigned to partition 0.
- **Step helper (non-revisiting):** A node is assigned to a selected partition and then removed from the action space. Each node is therefore assigned at most once, resulting in a constructive generation of the partition. The reward corresponds to the resulting change in cut value.
- **Step helper (revisiting):** When revisiting is enabled, any node can be selected repeatedly. Each action toggles the node between the two partitions. The cut value is updated according to the change in the crossing edges caused by the move, allowing previously made assignments to be reversed.
- **Objective and reward:** The objective is the current cut value C_t, and the reward is the incremental change with respect to the previous timestep. Thus, a positive reward corresponds to an improvement in the cut, while a negative reward corresponds to a deterioration. With relative rewards, this change is normalized by the difference between the empirically known best and worst cut values.
- **Termination:** In the constructive setting, the episode terminates once all nodes have been assigned, since a complete partition has been constructed, or when the empirically known best cut is reached. In the revisiting setting, node assignments are reversible and the agent can continue modifying the partition to improve the cut. Therefore, termination is controlled by the search horizon, unless the empirically known best cut is reached.
- **Score:** The score is based on the best cut value obtained during the episode, normalized relative to the empirically known best and worst solutions.

B.4 VIRTUAL MACHINE PLACEMENT

Logic: Assign virtual machines (VMs) to physical machines (PMs) to jointly optimize utilization, energy efficiency, packing efficiency, load balancing, and security risk (representing the attack surface), subject to capacity constraints.

- **Parameters:** Number of VMs, PMs, tenants; VM resource demands (CPU, MIPS, RAM, storage); PM capacities; traffic, latency, energy, and security characteristics.
- **Graph:** Nodes represent VMs and PMs; edges encode VM–PM allocations, VM-to-VM traffic, and PM-to-PM latency.
- **Node features:** VM nodes encode resource requirements, while PM nodes encode utilization and power consumption.
- **Edge features:** Traffic demands and latency values.
- **Reset:** The system is initialized with the empirically known worst allocation to encourage exploration of action sequences that improve the allocation.
- **Step helper (non-revisiting):** A (VM, PM) assignment is validated against capacity constraints and applied if feasible. Once a VM has been moved, it is removed from the action space and cannot be selected again, resulting in a constructive allocation process.
- **Step helper (revisiting):** When revisiting is enabled, a VM remains in the action space after being moved and can subsequently be migrated to another PM. Each valid migration updates the allocation and the resulting system metrics, allowing previously made placement decisions to be reversed or refined.
- **Objective and reward:** The reward is based on the variation of the weighted normalized system metrics between consecutive allocations, with an additional penalty for migration cost. Thus, a positive reward corresponds to an improvement in the current allocation, while a negative reward corresponds to a deterioration or costly migration.
- **Termination:** In the constructive setting, the episode terminates when no valid VM assignments remain or when the empirically known best score is reached. In the revisiting setting, VMs remain available for migration, allowing the agent to iteratively improve the allocation. Therefore, termination is controlled by the exploration horizon, unless the empirically known best score is reached.
- **Score:** The score is the weighted sum of the final normalized metric values, with the best allocation encountered during revisiting retained as the solution.

B.5 CYBER-ATTACK PATH PREDICTION

Logic: Predict critical multi-step attack paths in a networked environment by sequentially exploiting vulnerabilities distributed across interconnected host nodes. The agent models an attacker operating under partial observability, incrementally discovering the network topology as the attack progresses. Formulated as a Partially Observable MDP, this setting captures the uncertainty inherent to reconnaissance and lateral movement.

- **Parameters:** Number of nodes, vulnerabilities per node, communication probability distribution (modeling possible remote exploitation), visibility probability distribution, detection probability distribution (modeling failure rates), and attack-related parameters.
- **Graph:** A dynamic attack graph in which nodes and connections become progressively available to the agent. Edges represent attack actions that have already been executed, including the vulnerabilities exploited between source and target nodes.
- **Node features:** Aggregated service/vulnerability BERT embeddings using order- and size-invariant poolings, together with visibility, compromise status, privilege level, data presence, exfiltration status, persistence, DoS, and defense-evasion flags.
- **Edge features:** Vulnerability BERT embeddings representing the vulnerabilities used by the selected attack actions. Multiple vulnerabilities exploited between the same source–target pair are pooled under the corresponding edge.

- **Reset:** The episode starts from a randomly compromised node with partial visibility of its neighborhood, representing an attacker with incomplete knowledge of the network.
- **Step helper:** At each step, the agent selects a $(source, target, vulnerability)$ action. The action is validated against the current exploitability and privilege constraints. If successful, the corresponding node state is updated and the attack graph and available action space may expand as new information is discovered.
- **Reward:** Outcome-driven rewards reflect the attacker's objective under a "control" threat model. Successful outcomes such as credential access or lateral movement provide positive rewards, while unsuccessful or defense-related outcomes can provide negative rewards. Repeating an outcome that has already been achieved provides no additional reward.
- **Termination:** The episode terminates according to the attack-specific termination condition, such as compromising all target nodes, or when a predefined cutoff is reached.
- **Score:** The score is the percentage of nodes successfully compromised during the attack trajectory.
- **Revisiting strategy:** Not supported as a separate revisiting strategy because the benchmark is inherently sequential. Each action represents the next step of an evolving attack path under the current network state and partial observability, rather than a reversible modification of a previously constructed solution.

B.6 OSPF Engineering

Logic: Adjust link weights to minimize congestion by reducing maximum link utilization under routing constraints.

- **Parameters:** Number of nodes, capacity range, traffic range, weight bounds, graph distribution, and Equal-Cost Multi-Path (ECMP) routing flag.
- **Graph:** A spanning graph with undirected communication edges and directed traffic flows.
- **Node features:** Aggregated incoming and outgoing traffic.
- **Edge features:** Capacity, utilization, weight, and traffic load.
- **Reset:** The system is initialized with the empirically known worst-case weight configuration to encourage exploration of action sequences that improve the routing configuration.
- **Step helper (non-revisiting):** A link is selected and assigned a final weight within the allowed bounds. Once modified, the link is marked as visited and cannot be selected again, resulting in a constructive weight configuration. Routing and link utilization are recomputed after each modification, according to the selected ECMP setting.
- **Step helper (revisiting):** When revisiting is enabled, a link can be selected repeatedly and its weight can be increased or decreased by a specified variation. The resulting weight is kept within the allowed bounds, after which routing and link utilization are recomputed. This allows previously made weight adjustments to be reversed or refined.
- **Objective and reward:** The reward is based on the change in maximum link utilization and, when applicable, the change in routing feasibility. For relative rewards,

$$r_t = \alpha\,(F_t - F_{t-1}) + \beta \frac{U_{t-1} - U_t}{U_{\text{initial}} - U_{\text{best}}},$$

  where F_t is the feasible traffic percentage and U_t is the aggregated link utilization. Thus, a positive reward corresponds to an improvement in feasibility and/or a reduction in link utilization.
- **Termination:** In the constructive setting, the episode terminates once all communication links have been assigned a weight or when the empirically known best utilization is reached. In the revisiting setting, links remain available for further weight adjustments, allowing the agent to iteratively improve the routing configuration. Termination therefore occurs when the target configuration is reached or according to the predefined exploration horizon.
- **Score:** The score is based on the reduction in maximum link utilization relative to the initial configuration, while accounting for routing feasibility.

B.7 TRAFFIC ENGINEERING

Logic: Route traffic demands through feasible paths to minimize congestion while respecting capacity constraints. This represents a variant of OSPF engineering without relying on OSPF as an intermediary helper.

- **Parameters:** Number of nodes, capacity range, traffic characteristics, graph distribution, and maximum path length, which restricts the set of communication paths considered as candidate solutions.
- **Graph:** A spanning graph with communication and traffic edges.
- **Node features:** Aggregated incoming and outgoing traffic.
- **Edge features:** Capacity, utilization, and traffic load.
- **Reset:** The system is initialized with the empirically known worst-case allocation configuration to encourage exploration of action sequences that improve the routing configuration.
- **Step helper (non-revisiting):** A traffic demand is assigned to a feasible communication path. Once selected, the traffic demand is marked as visited and cannot be selected again, resulting in a constructive routing process.
- **Step helper (revisiting):** When revisiting is enabled, a traffic demand can be selected repeatedly and assigned to a different communication path. The previously allocated path is deallocated before the new path is assigned, allowing previously made routing decisions to be reversed or refined.
- **Objective and reward:** The reward is based on the change in routing feasibility and link utilization between consecutive allocations. For relative rewards,

$$r_t = \alpha \frac{F_t - F_{t-1}}{F_{t-1}} + \beta \frac{U_{t-1} - U_t}{U_{\text{initial}} - U_{\text{best}}},$$

  where F_t is the feasible traffic percentage and U_t is the aggregated link utilization. Thus, a positive reward corresponds to improved feasibility and/or reduced link utilization.
- **Termination:** In the constructive setting, the episode terminates once all traffic demands have been assigned or when the empirically known best routing is reached. In the revisiting setting, traffic demands remain available for rerouting, allowing the agent to iteratively improve the allocation. Termination therefore occurs when the target routing quality is reached or according to the predefined exploration horizon.
- **Score:** The score is based on the reduction in maximum link utilization relative to the initial configuration, while accounting for routing feasibility.

C HYPER-PARAMETER OPTIMIZATION

	Learning Rate	Batch Size	γ	N Steps	Ent Coef	Max Grad Norm	Tau	Target Update Interval	Num Layers	NN Channels	Out Channels	Activation
Discrete PPO	0.00001, 0.0001, **0.001**	32, **64**, 128	**0.9**, 0.95, 0.99	512, 1024, **2048**	**0.01**, 0.1, 0.2	0.1, 0.3, **0.5**	–	–	–	–	–	–
Projection PPO	0.00001, 0.0001, **0.001**	32, **64**, 128	**0.9**, 0.95, 0.99	512, 1024, **2048**	**0.01**, 0.1, 0.2	0.1, 0.3, **0.5**	–	–	–	–	–	–
Iterative IDQN	0.00001, 0.00005, **0.0001**	**32**, 64, 128	0.9, **0.95**, 0.99	1, 3, **5**	–	–	0.01, **0.05**, 0.1	2000, 5000, **10000**	–	–	–	–
GAE	0.00005, 0.0001, 0.001, **0.01**, 0.05	16, **32**, 64	–	–	–	–	–	–	**2**, 3	16, **32**	**16**, 32, 64	LeakyReLU, **ReLU**, null

Table 5: Hyperparameter ranges explored for each method with best configurations in **bold**.

Table 5 reports the hyperparameter search space explored for each optimized model, along with the best configuration identified. The hyperparameter optimization was conducted exclusively on the *TSP* benchmark for simplicity, using 25 trials for each model. Regarding discrete solutions, this process has been performed only on the P-Discrete-M variant for simplicity, and hyperparameters were used across all other discrete variants.

The following additional hyperparameters were tuned manually and selected based on empirical evaluation:

- **Node embeddings dimensionality:** 16
- **Nearest neighbor search (action space):** FAISS Flat Index
- **Distance metric (action space search):** Cosine similarity
- **Action vector processing:**
  - No normalization in the iterative approach
  - Z-score normalization in the projection approach (ensures projection outputs remain within a suitable range for policy-based methods)
- **Action space bounds:** Extended by a margin of $(-1, +1)$ (to the z-score bounds) relative to the empirical latent space range discovered across the training instances per benchmark
- **Episode length:** To maintain consistency across varying problem scenarios, the episode length is scaled proportionally to the instance complexity. Concretely, it is defined as a coefficient multiplied by the number of decision variables (e.g., number of VMs in *Placement*), ensuring that agents are given sufficient interaction steps with respect to scenario size:
  - Default: $1\times$ scenario size
  - Cyber-Path / traffic scenarios: $3\times$ scenario size
  - OSPF engineering scenarios: $2\times$ scenario size
  - Maxcut scenarios: $2\times$ scenario size
- **Reward normalization:** Min-max normalization (Stable-Baselines3 default)
- **Feature normalization:**
  - Node features are normalized independently using min-max scaling based on their respective ranges within the graph, ensuring feature scale independence
  - L1 normalization applied to language model embeddings when used as features, as ranges can hardly be determined
- **Scenario switching during training (only for *V* experiments):**
  - Interval: every 50 episodes
  - Strategy: random selection
- **Policy architecture:**
  - Hidden layers: [128, 64]
  - Subsequent output layer proportional to the action space and agent formulation used
  - Activation: LeakyReLU
  - Optimizer: Adam ($\epsilon = 10^{-7}$, weight decay $= 10^{-4}$, AMSGrad disabled)
- **Graph Autoencoder (GAE):**
  - Final layer: no activation function, with normalization enabled to ensure properly scaled embeddings
  - Loss functions (all adjusted to have the same scale, and weights all set to 1):
    - Continuous features: Mean Squared Error (MSE)
    - Adjacency matrix: Contrastive loss ($\tau = 0.5$)
    - Binary features: Binary Cross-Entropy with logits
    - Multi-categorical features: Cross-Entropy loss

Seeds. The random seeds used in the experiments are defined as follows. A seed of 42 is used for the scenario generation, initial experimental setup, and splitting strategies. For the generalization study, five independent runs per strategy and RL solution are performed with the following seeds used during training: 42, 100, 123, 200, and 300. For the testing phase, seeds are generated based on the number of episodes per scenario instance, ensuring a one-to-one correspondence and symmetry across test episode identifiers of several scenario instances. The resulting sequence of test seeds is: 42, 100, 123, 200, followed by increments of 100 up to 5000 (i.e., 400, 500, 600, . . . , 5000). The same sequence is used for assessing the distribution of action-selection time.

All other hyperparameters were set to their respective library default values and are provided in the accompanying configuration files.

D SCENARIO GENERATION

As described in Section 4.1, the generalization study relies on 101 generated instances obtained by randomly sampling the parameters of the environment within predefined ranges. These ranges are designed to induce variability across instances and are reported below; all other parameters are held constant as indicated.

Traveling Salesman Problem
`num_cities`:[10,100]
`max_coord`:[100,1000]

Maximum Cut
`num_nodes`:[10,100]
`max_weight`:[10,100]

Minimum Vertex Cover
`num_nodes`:[10,50]
`edge_prob`:[0.1,0.4]

OSPF Engineering
`num_nodes`:[10,30]
`communication_edge_ratio`:[0.1,0.3]
`non_zero_traffic_ratio`:[0.1,0.3]
`min_capacity`:[10,100]
`max_capacity`:[500,1000]
`max_traffic`:[25,50]
`graph_distribution`: `spanning tree`
`min_weight`: 1
`max_weight`: 5

Traffic Engineering
`num_nodes`:[10,25]
`communication_edge_ratio`:[0.1,0.2]
`non_zero_traffic_ratio`:[0.1,0.2]
`min_capacity`:[10,100]
`max_capacity`:[500,1000]
`max_traffic`:[25,50]
`max_path_len`: 4
`graph_distribution`: `spanning tree`

Virtual Machine Placement
`n_vms`:[10,50]
`n_pms`:[10,50]
`n_tenants`:[2,5]
`vm_vuln_prob_min`:[0.01,0.1]
`vm_vuln_prob_max`:[0.2,1.0]
`pm_escape_prob_min`:[0.01,0.1]
`pm_escape_prob_max`:[0.2,0.33]
`pm_capacity_memory_min`:[32,64]
`pm_capacity_memory_max`:[128,512]
`vm_demand_memory_min`:[1,8]
`vm_demand_memory_max`:[16,24]
`pm_capacity_storage_min`:[100,500]
`pm_capacity_storage_max`:[1000,5000]
`vm_demand_storage_min`:[10,50]
`vm_demand_storage_max`:[60,100]
`pm_capacity_pe_min`:[8,16]
`pm_capacity_pe_max`:[32,128]
`vm_demand_pe_min`:[2,4]
`vm_demand_pe_max`:[8,16]
`pm_capacity_mips_min`:[1000,5000]
`pm_capacity_mips_max`:[10000,50000]
`vm_demand_mips_min`:[100,500]
`vm_demand_mips_max`:[750,1000]
`latency_min`:[0.1,1.0]
`latency_max`:[2,8]
`p_idle`:[50,100]
`p_peak`:[150,300]
`coefficients`: 1
`min_traffic`: 1
`max_traffic`: 10
`traffic_density`: 0.8

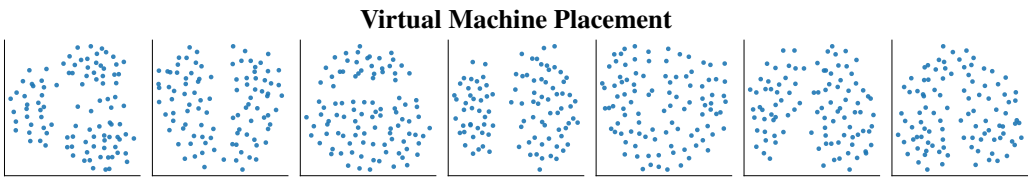
Cyber-Attack Path Prediction
`n_nodes`:[10,20]
`n_vulns_per_node`:5
`vulns_overlap`:[0.0,0.1]
`p_data_present`:[0.6,0.9]
`p_feature_visible`:[0.5,0.7]
`p_recon`:[0.2,0.4]
`p_detection`:[0.05,0.2]
`goal`: control
`fully_connected`: true
`outcome_selection`: false

Scalability experiments present an evolving set of values for the main dimension (number of nodes) and the same ranges for secondary features.
Scenario instances are exhaustively evaluated using heuristic-based exploration to estimate empirical upper and lower bounds. The number of sweeps is set based on preliminary experiments (10000 for TSP, 2000 for MaxCut, 10000 for MVC, 5000 for Placement, 2000 for OSPF, and 2000 for Traffic). The Cyber-Path scenario is omitted as its bounds can be directly inferred from the number of nodes (which becomes the maximum relative score). The heuristics guiding exploration were designed using a language model (gpt-5.4) and refined through manual validation.

- *TSP*: We generate random tours and iteratively refine them using stochastic 2-opt moves. Improvements (or degradations) are sampled from candidate swaps, biased by their impact on tour length. This allows simultaneous estimation of both best and worst-case tour costs.
- *MinVertex*: A greedy-stochastic strategy is used to iteratively select nodes covering the largest number of uncovered edges. At each step, selection is randomized among the top-k candidates to maintain diversity. A pruning phase removes redundant nodes to refine solutions. The best cover size is tracked across multiple sweeps. The worst is set to the overall number of nodes.
- *MaxCut*: Random graph partitions are initialized and improved via stochastic local search. Node flips that increase the cut value are identified, and one is selected probabilistically based on gain. This process continues until no further improvements are possible. The maximum cut value observed across samples defines the upper bound. The worst value is set to 0.
- *Placement*: Allocations are constructed using heuristics that either maximize or balance resource utilization across physical machines. Initial solutions are built by prioritizing high-demand VMs, followed by local refinements through stochastic reassignment. Both best and worst allocations are identified through repeated sweeps.
- *OSPF*: OSPF link-weight settings are explored through stochastic search over candidate configurations. Each sweep starts either from a random solution or from a perturbed elite solution sampled from small best/worst pools, allowing intensified search without losing diversity. Neighboring configurations are obtained by randomly modifying a subset of link weights, and are selected with a bias toward lower and higher utilization to estimate both worst and best. Previously explored configurations are excluded.
- *Traffic*: Same idea as *OSPF* but optimizing path allocations and not edge weights.

E EMBEDDING ACTION SPACE PLOTS

Figure 5 presents the UMAP projection of the latent action spaces for seven representative 20-node scenarios from each real-world benchmark, highlighting both their variability and the common structural patterns that emerge in the action space manifolds. The scenarios were manually selected as the most representative manifolds based on exploratory analysis. Classical benchmarks were excluded due to their limited diversity in terms of manifolds, with respect to the already provided plots in Section 4.



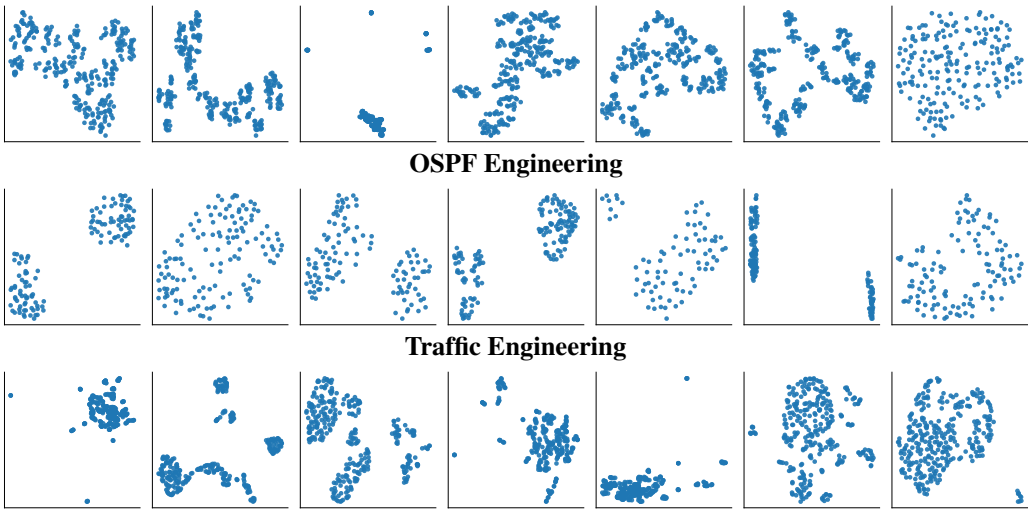


Figure 5: UMAP projections of action spaces for seven 20-node graph instances across the four real-world benchmarks, with up to 300 actions (sampled by maintaining a representative approximation of the underlying action distribution).

F Graph Representation Learning Ablations

We conduct ablations to assess the role of GNN-based representations in the action embedding space used by the projection agent.

F.1 Action Space Ablation

We study how the representation of the action space affects the projection agent. We compare three representations: randomly initialized vectors, the raw feature vector of each node, and GNN-derived embeddings for benchmarks with nodes as actions (TSP, MinVertex, MaxCut). All variants use the same embedding dimensionality and projection-agent architecture, and are evaluated under the smallest training regime (S) using the same protocol as in Section 4.1.
Table 6 reports the results. We report Δ_{train}, the change from training to test performance, and Δ_{GNN}, the difference relative to the GNN-based representation. The random-vector variant performs substantially worse across all problems, showing that the projection agent benefits from semantically meaningful action representations rather than merely from placing action anchors in a continuous space. Using raw node features recovers much of this performance, demonstrating that node semantics alone already provide useful information. However, the GNN representation performs better, indicating that there is advantage in enriching node features with information about the surrounding graph structure for these benchmarks.

F.2 Ablation of GAE Loss Weights

We further investigate the contribution of the individual reconstruction objectives used to pre-train the GAE. As described in Section 3.2, the self-supervised objective jointly reconstructs node features, edge features, and the adjacency matrix. Since these objectives are optimized jointly within a limited-dimensional embedding space, they may impose competing constraints on the learned representation. We therefore conduct a two-stage ablation to first verify that each objective captures its intended graph information, and then assess whether this information benefits downstream RL-GCO.

Reconstruction ablation. We first evaluate whether each reconstruction objective effectively contributes to learning its corresponding graph information, independently of the RL downstream task. We train four GAE variants: the full model and three ablated versions, each removing one reconstruction objective (adjacency, edge, or node reconstruction). Preliminary results on TSP are reported below, with normalized reconstruction scores. Models are pre-trained on 50 graph instances (25 for training and 25 for testing) and evaluated over 50 test episodes. Reconstruction scores are normalized such that 0 corresponds to the average loss of 50 randomly initialized GAEs and 1 to perfect reconstruction (loss = 0).

Table 6: Projection-agent performance with different action embeddings. Random embeddings are compared against embeddings initialized with the feature vector of the corresponding node.

Problem	Embedding	Score	Δ_{train}	Δ_{GNN}
TSP	Random	0.36	-0.37	-0.42
	Node feature	0.64	-0.32	-0.12
MinVertex	Random	0.36	-0.57	-0.34
	Node feature	0.21	-0.19	-0.59
MaxCut	Random	0.36	-0.57	-0.57
	Node feature	0.79	-0.19	-0.14

Table 7: Ablation of GAE reconstruction objectives on TSP. The left block reports normalized reconstruction scores for each objective, while the right block reports downstream projection-agent performance under the S training regime.

	Self-supervised reconstruction			Downstream optimization			
Model	Adj.	Edge	Node	Mean	IQM	Std	95% BCI
Full model	0.04	0.97	0.48	0.571	0.566	0.115	[0.561, 0.581]
− Adjacency	0.03	0.97	0.48	0.593	0.601	0.109	[0.584, 0.603]
− Edge	0.15	0.12	0.47	0.566	0.558	0.099	[0.558, 0.575]
− Node	0.07	0.96	0.01	0.519	0.504	0.117	[0.508, 0.529]

Results in Table 7 (self-supervised reconstruction column) shows how removing each objective primarily degrades reconstruction of the corresponding graph component, confirming that the three objectives encode distinct and relevant aspects of the graph structure. The slight improvements observed in some components (e.g., adjacency after removing edge reconstruction) highlights the trade-off between competing objectives during optimization.

Impact on RL-GCO. We next assess whether these learned signals translate into improved policy performance. Using the smallest training regime (S), we follow the experimental protocol of Section 4.1 and evaluate each variant over five runs on TSP. For each ablation, one reconstruction objective is removed while all other components remain unchanged. Results are reported in Table 7 (downstream optimization column).
Removing node or edge reconstruction reduces downstream performance, indicating that the corresponding information is useful for policy learning. Interestingly, removing adjacency reconstruction improves performance. This highlights that some graph information is not necessarily beneficial for the downstream RL objective: jointly reconstructing multiple graph properties can introduce competing constraints within the limited-dimensional latent space. Thus, the reconstruction objective should be viewed as a representation-design choice rather than assuming that incorporating every available graph signal necessarily produces the best policy representation.

G Comparison with Other Solvers

To assess the value of learning heuristics with our RL-GCO methodology and further validate our experimental framework, we compare the projection agent against both classical solvers and alternative learning-based heuristics. We consider two representative benchmarks, MaxCut and TSP, and use the same instances and generalization regimes as in Section 4.1 to ensure a direct and consistent comparison.

G.1 Maximum Cut: Beyond RL Solvers

For MaxCut, we compare the projection agent with PI-GNN Schuetz et al. (2022), a physics-inspired, unsupervised GNN that provides a representative non-RL, non-autoregressive approach to MaxCut, as well as simulated annealing (SA) as a classical optimization baseline. Since PI-GNN was originally evaluated on unweighted MaxCut, we adapt its methodology to the weighted setting considered in our experiments.

Table 8: Comparison between PI-GNN and the projection agent on MaxCut under the S, M, and L generalization regimes. PI-GNN is evaluated both with the hyperparameters reported in the original work and after hyperparameter optimization on a held-out validation set.

Method	S	M	L
PI-GNN (reported)	0.52	0.61	0.22
PI-GNN (tuned)	0.92	0.91	0.92
Projection (LaGCO-RL)	**0.95**	**0.95**	0.92

Validation of the normalization reference. Our evaluation normalizes solution quality relative to the best solution obtained from a heuristic-driven sweep of 20,000 candidate solutions. To assess whether this provides a reliable reference, we compare the resulting solutions with those obtained by simulated annealing on the same instances. Across 101 scenarios, the two procedures identified identical best solutions in 98 cases. In the remaining three cases, the difference in normalized solution quality was only 0.0002. This close agreement indicates that the heuristic sweep provides a reliable approximation to the SA reference on our MaxCut instances. Thus, the normalized scores reported throughout this section can also be interpreted as relative performance with respect to a strong classical optimization baseline.

Comparison with PI-GNN. We evaluate PI-GNN and the projection agent on identical graph instances and train/test splits, using the S, M, and L generalization regimes defined in Section 4.1. We do not consider the V regime because PI-GNN assumes learning from a single instance. For both methods, we use the same best-of-five evaluation protocol and normalization procedure, allowing for a direct comparison between the two learning paradigms.

PI-GNN is known to be sensitive to hyperparameter selection. We therefore report two configurations: (i) the hyperparameters specified in the original work and (ii) a configuration obtained through hyperparameter optimization on a held-out validation set from our benchmark. This distinction separates the performance of the published configuration from the performance achievable with problem-specific tuning.

The projection agent achieves the strongest overall performance, outperforming the tuned PI-GNN configuration in the S and M regimes and matching it in the L regime. The substantial improvement obtained by tuning PI-GNN also highlights the sensitivity of this approach to hyperparameter selection. Importantly, the close agreement between the heuristic-sweep reference and simulated annealing provides additional context for interpreting these results: a normalized score of approximately 0.95 corresponds to achieving roughly 95% of the solution quality obtained by the strong classical SA reference on these instances.

Overall, these results indicate that the projection agent is competitive with a specialized non-RL, non-autoregressive solver on MaxCut, while requiring no autoregressive decoding and remaining robust across the considered generalization regimes.

G.2 TSP: Beyond RL Solvers

For TSP, we compare the projection agent with POMO Kwon et al. (2020), an RL-based solver that represents a prominent approach for this benchmark. We additionally use Concorde as a reference solver for assessing solution quality against high-quality, and where applicable optimal, solutions.

Validation of the normalization reference. As in the other benchmarks, our initial evaluation normalizes solution quality relative to the best solution obtained from a heuristic-driven sweep, using 10,000 candidate solutions per instance. Unlike MaxCut, however, we found that the solutions obtained by this procedure frequently differed from those produced by Concorde. This indicates that, for TSP, the heuristic-sweep reference does not provide a sufficiently reliable proxy for the reference solution quality and makes direct comparison with Concorde less meaningful under the original normalization.

We therefore perform an additional retraining and evaluation using an empirical Concorde-based reference. This rerun ensures that the reported results are directly comparable to the solutions produced by the reference solver and avoids conflating differences in normalization with differences in solver performance.

Table 9: Comparison between POMO and the projection agent on TSP under the S, M, L, and V generalization regimes.

Method	S	M	L	V
POMO	**0.81**	**0.81**	**0.81**	**0.81**
Iterative (LaGCO-RL)	$\bar{0}.\bar{7}\bar{2}$	$\bar{0}.\bar{6}\bar{9}$	$\bar{0}.\bar{6}\bar{7}$	$\bar{0}.\bar{7}\bar{1}$
Projection (LaGCO-RL)	0.67	0.52	0.42	0.44

Comparison with POMO. We evaluate POMO and the projection agent on identical graph instances and train/test splits, using the S, M, L, and V generalization regimes defined in Section 4.1. Both methods are evaluated using the same best-of-five protocol and the same normalization procedure within the comparison, replacing only the upper bound estimate.
In contrast to the MaxCut results, the projection agent does not reach the performance of the specialized POMO-RL solver on TSP. For a comparison within our framework, however, the relevant reference is our iterative baseline, which is the strongest of the considered parameterization approaches on this benchmark. The iterative agent remains below the POMO baseline, with a minimum gap of 0.04 in the V regime. Overall, the comparison with POMO provides a useful indication of the remaining gap to a solver specifically designed for TSP.

H Library and Data

This subsection presents our `LaGCO-RL` library and its primary research contribution: a framework for studying graph representation learning, latent action-space design, and Wolpertinger-based RL for GCO (Sections 3.2–3.4). It complements existing frameworks such as RL4CO Berto et al. (2025), which focuses on unified benchmarking across CO problems and RL algorithms, and RL-Solver Zhu and Pu (2025), which emphasizes scalable RL through massively parallel GPU execution, by providing complementary capabilities for research on graph representations and latent action spaces for GCO.
At the environment level, users define a standard Gym environment encoding the task logic and extend it through a class inheriting from both `ContinuousEnv` (Fig. 6) and a task-specific base class. The framework core idea is to minimize task-specific engineering, requiring only a small set of high-level attributes and methods leveraged by a `gym.Wrapper` module. The graph structure should be provided through **get_graphs()**, with step-varying attributes, while node and edge encodings are defined via **_node_attributes** and **_edge_attributes**, enabling automatic construction of decoder heads and loss functions for GNN self-supervised learning, supported by **sample_valid_action()**. Observations are then derived through **_observation_type**, using pooling-based or custom invariant functions. The action space is specified via **_action_type** as combinations of proper decision elements; the framework generates the full action set via Cartesian products and optionally filters infeasible actions using **is_valid_action(action)**, allowing time-varying action spaces. The resulting continuous space is exposed as a **gym.Box**, aligned with the projection agent's action space, or appropriately wrapped for compatibility in case of other RL strategies.

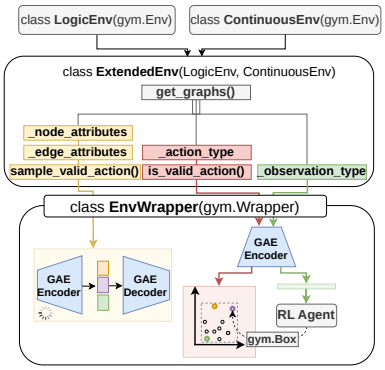


Figure 6: User-defined logic, attributes, and methods provided for GAE pretraining (left) and RL integration (right).

The *LaGCO-RL* library will be released under the GPL-3.0 License alongside this paper. The associated data repository contains the simulated graph instances, trained models, dataset splits, and hyperparameter configurations. The data is distributed under the CC BY 4.0 open license.
To facilitate reproducibility, the repository includes detailed `README` and `REPRODUCIBILITY` documents describing how to use the tool, reproduce the experiments, and access the data.
Figures 7 and 8 illustrate two concrete instantiations of the framework enabled by the *LaGCO-RL* library, showcasing two examples, respectively *cyber-attack path prediction* and *traffic engineering*, and how the corresponding environments can be instantiated in our framework. These examples

Traffic Engineering

```
class TrafficEngineeringExt(TrafficEngineering, ContinuousEnv):

def get_graphs(...):
    comm_G = ...
    traffic_G = ...
    return {"comm_G": comm_G, "traffic_G": traffic_G}

self._observation_type = {
    "comm": Graph(poolings=[Mean],
                  graph="comm_G"),
    "traffic": Graph(poolings=[Mean],
                  graph="traffic_G"),
    ...
}

self._action_type = {
    "traffic_link": Edge(graph="traffic_G", pooling=Concat),
    "communication_path": Path(graph="comm_G", pooling=Concat)
    ...
}

self._node_attributes = {
    "outgoing_traffic": Attribute(norm="min_max"),
    ...
}
self._edge_attributes = { ... }

def sample_valid_action(...): ...
def is_valid_action(...): ...
```

Figure 7: An instantiation of the automated framework with an example of the definition of the observation and action space, attribute specification, and proper methods to support the traffic engineering benchmark.

provides pseudo-code demonstrating how to define key components, including graph construction, observation and action spaces, attribute specifications, and supporting functions.

Figure 8 presents a potential choice of framework instantiation for the cybersecurity benchmark. The observation space combines graph-level representations with auxiliary features (e.g., number of nodes), while the action space is compositional, involving the selection of a source node, a target node, and an associated vulnerability. This example highlights the framework's ability to support structured actions over graph elements and its components in a conditional manner, as well as the integration of learned feature extractors (e.g., BERT-based embeddings) for semantic attributes such as vulnerabilities. In contrast, Figure 7 demonstrates a more complex setting in which multiple graph views are used to represent different aspects of the same environment instance. Specifically, a communication graph and a traffic graph are jointly defined, enabling the agent to reason over both topological connectivity and traffic flow dynamics from different graph representations. The observation space aggregates information independently from each graph, while the action space combines elements from both representations to form each action (i.e., selecting traffic links and communication paths). This highlights the flexibility of the framework in handling multi-graph environments and cross-graph interactions.

Additionally, the traffic engineering example illustrates the use of pooling operators for different graph components, as well as normalization strategies (e.g., min–max scaling) applied to node attributes to ensure consistent feature scaling across different instances.

I Observation space

Latent observation. The observation vector used in Section 4 for latent observation agents (G-Discrete, G-Discrete-M, Projection, Iterative) is defined as a fixed-dimensional representation that combines graph-level information and, for the projection agent, a summarized view of the action space. This design ensures sufficient expressiveness while maintaining a compact representation of both the environment structure and the decision space. The observation is constructed as follows:

- **Graph embedding:** Node embeddings are aggregated using multiple pooling strategies to capture complementary structural information:

Cyber-Path

```
class CyberAttackExt(CyberAttack, ContinuousEnv):

def get_graphs(...):
    G = ...
    return {"attack_G": G}

self._observation_type = {
    "graph": Graph(poolings=[Mean, Sum], graph="attack_G"),
    "nodes_num": Function(get_num_nodes, graph="attack_G"),
    ...
}

self._action_type = {
    "source": Node(graph="attack_G", spec={"controlled": true}),
    "target": Node(graph="attack_G"),
    "vuln": Object(extractor='bert', reference="target", key="vulns")
}

self._node_attributes = {
    "vulns": Attribute(extractor='bert', type='continuous')
    ...
}
self._edge_attributes = { ... }

def sample_valid_action(...): ...
def is_valid_action(...): ...
```

Figure 8: An instantiation of the automated framework with an example of the definition of the observation and action space, attribute specification, and proper methods to support the cyber-attack path prediction benchmark.

  - Mean pooling
  - Max pooling
  - Min pooling
  - Sum pooling
- **Structural descriptors:** Additional invariant features are included to explicitly encode global graph properties:
  - Number of nodes
  - Number of edges
  - Average node degree
  - Graph density
- **Action-space summarization (projection agent only):** For benchmarks with structured action spaces that cannot be fully captured through node embeddings alone (which are already aggregated in the graph embedding), the action space is further embedded and aggregated using:
  - Mean pooling
  - Max pooling
  - Min pooling
  - Sum pooling

  The resulting pooled representation is concatenated to the observation vector, providing a compact summary of the latent action space. This additional vector is provided only to the projection agent, as the iterative agent inherently iterates over all embeddings, resulting in inherent visibility over the latent action space.

Padding observation. For the padding-based discrete baselines, observations are explicitly constructed to encode all relevant information in a fixed-size vector. These representations are benchmark-specific and capture features at the node, edge, graph, or path level, depending on the task, and are chosen to align with the graph features used for GNN-based embeddings.

The observation design for each benchmark is detailed below:

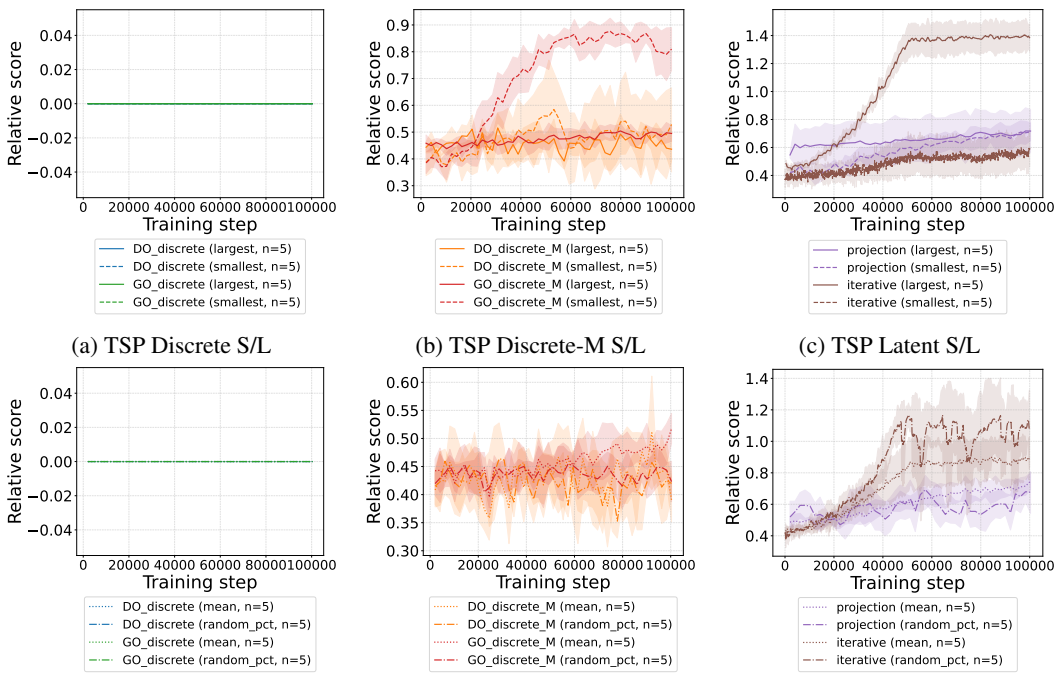
- **TSP.** The observation concatenates:
  - Flattened node coordinates
  - Binary visited status for each city
  - Flattened upper-triangular distance matrix (pairwise distances)
- **MinVertex.** The observation includes:
  - Node-level binary indicators specifying whether each node is selected
  - Edge-level binary indicators (flattened upper triangle) denoting whether edges are covered
- **MaxCut.** The observation consists of:
  - Node partition assignments
  - Flattened upper-triangular edge weights

  This representation captures both the graph structure and the current partitioning.
- **Placement.** The observation concatenates:
  - Physical machine (PM) features (e.g., resource capacities and usage)
  - Virtual machine (VM) features (e.g., resource demands)
- **Cyber-Path:** The observation is constructed as a concatenation of per-node feature vectors, encoding:
  - Visibility status (feature-level and graph-level)
  - Compromise state and privilege level
  - Service and vulnerability counts
  - Security-relevant flags (e.g., persistence, defense evasion, data exfiltration)
  - Aggregated vulnerability outcomes
  - Privilege requirements

  Features are partially masked depending on visibility, reflecting a partially observable setting, which is also done in the graph representation of the latent approaches.
- **OSPF.** The observation is built from edge-level features:
  - For communication links: capacity, routing weight, and used capacity
  - For traffic demands: traffic volume on each demand edge
- **Traffic.** The observation encodes:
  - Communication edge features: capacity and used capacity
  - Traffic demand values for each traffic edge

J LEARNING CURVES

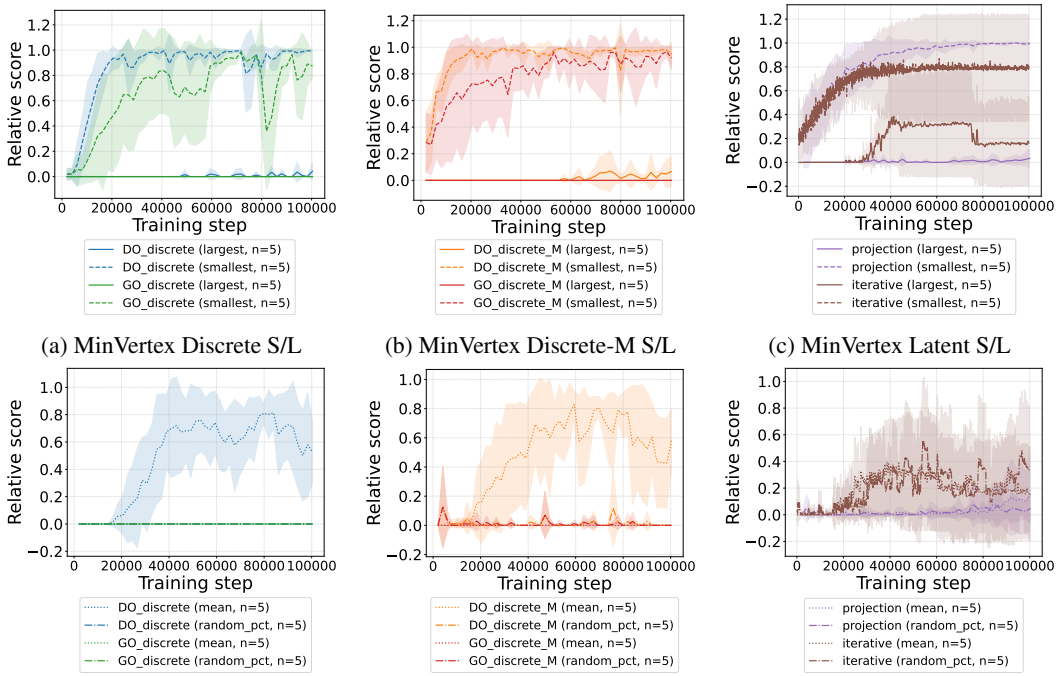
Figures 9, 10, 11, 12, 13, 14 and 15 show the training learning curves for the different approaches, reporting the mean and standard deviation computed over the five independent runs per training strategy. These curves also highlight that several runs fail to converge, particularly for iterative methods in real-world settings and for discrete approaches in some benchmark-strategy pairs when masking is not applied.
Figure 16 illustrates the evolution of the total loss during GAE training across benchmark environments, based on five representative 20-node samples and five training runs, with curves aggregated using mean and standard deviation. The results show higher variability on real-world benchmarks.



(a) TSP Discrete S/L (b) TSP Discrete-M S/L (c) TSP Latent S/L

(d) TSP Discrete M/V (e) TSP Discrete-M M/V (f) TSP Latent M/V

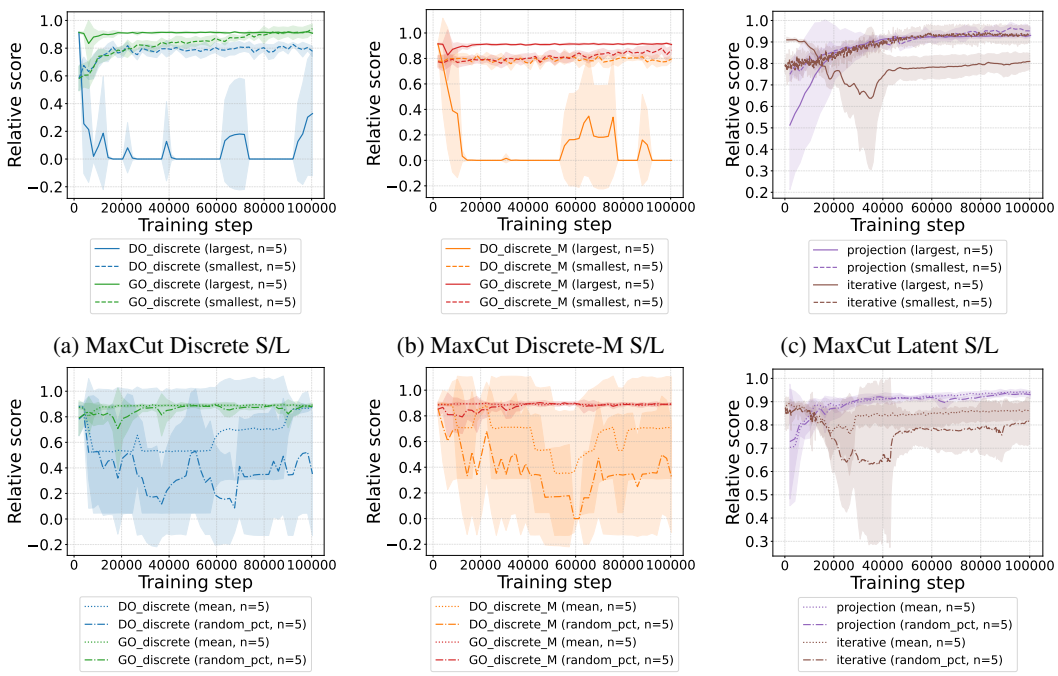
Figure 9: Average learning curves (mean ± standard deviation) across TSP benchmark environments under all training configurations.



(a) MinVertex Discrete S/L (b) MinVertex Discrete-M S/L (c) MinVertex Latent S/L

(d) MinVertex Discrete M/V (e) MinVertex Discrete-M M/V (f) MinVertex Latent M/V

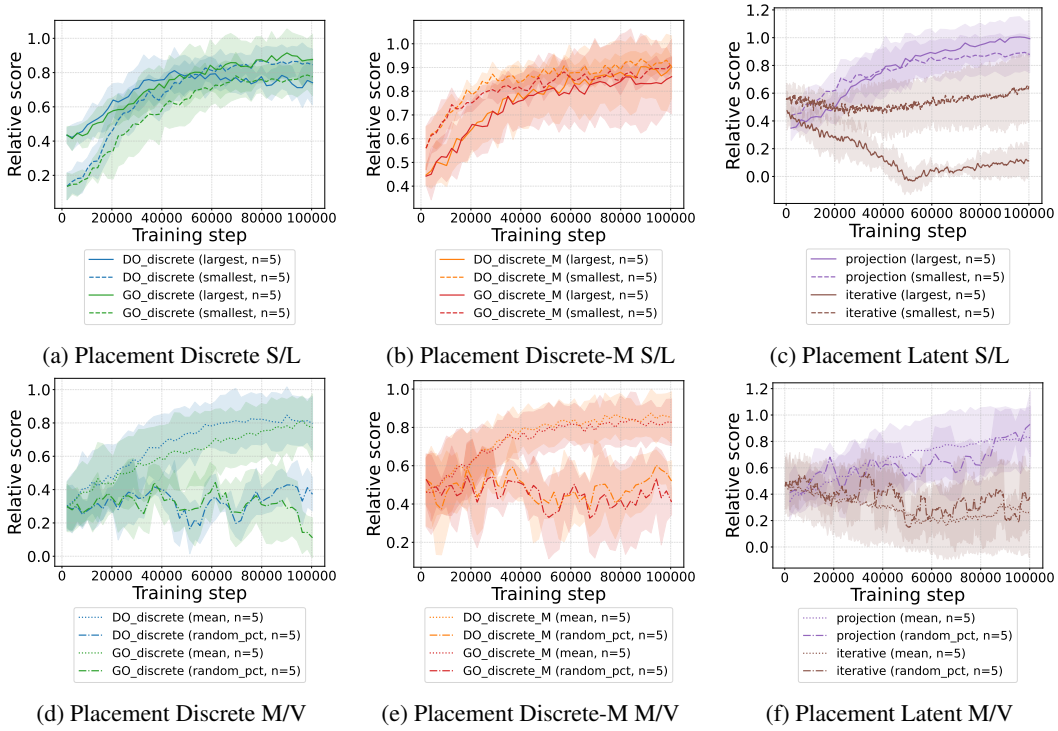
Figure 10: Average learning curves (mean ± standard deviation) across MinVertex benchmark environments under all training configurations.



(a) MaxCut Discrete S/L (b) MaxCut Discrete-M S/L (c) MaxCut Latent S/L

(d) MaxCut Discrete M/V (e) MaxCut Discrete-M M/V (f) MaxCut Latent M/V

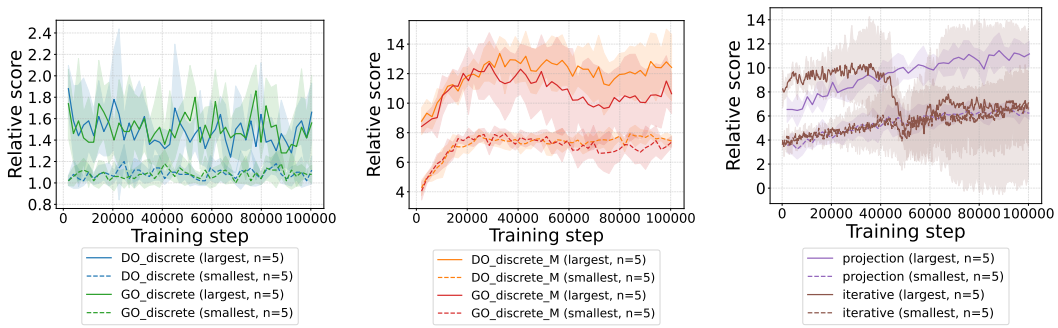
Figure 11: Average learning curves (mean ± standard deviation) across MaxCut benchmark environments under all training configurations.



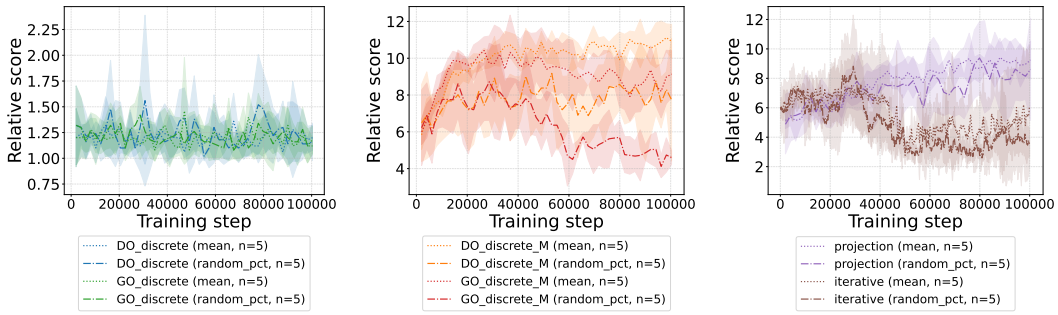
(a) Placement Discrete S/L (b) Placement Discrete-M S/L (c) Placement Latent S/L

(d) Placement Discrete M/V (e) Placement Discrete-M M/V (f) Placement Latent M/V

Figure 12: Average learning curves (mean ± standard deviation) across Placement benchmark environments under all training configurations.

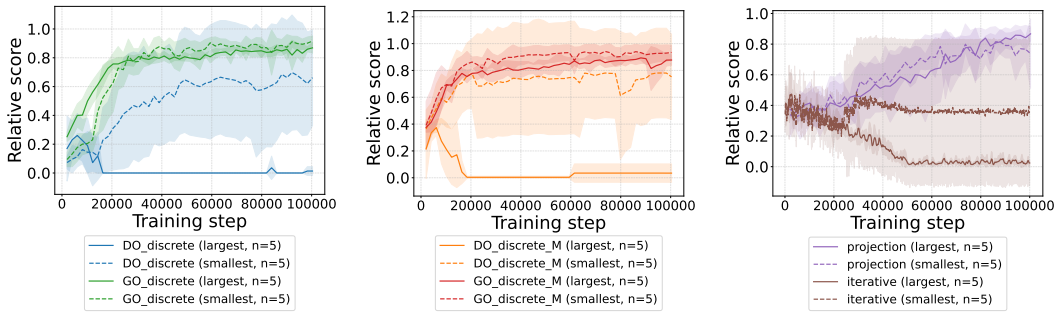


(a) Cyber-Path Discrete S/L (b) Cyber-Path Discrete-M S/L (c) Cyber-Path Latent S/L

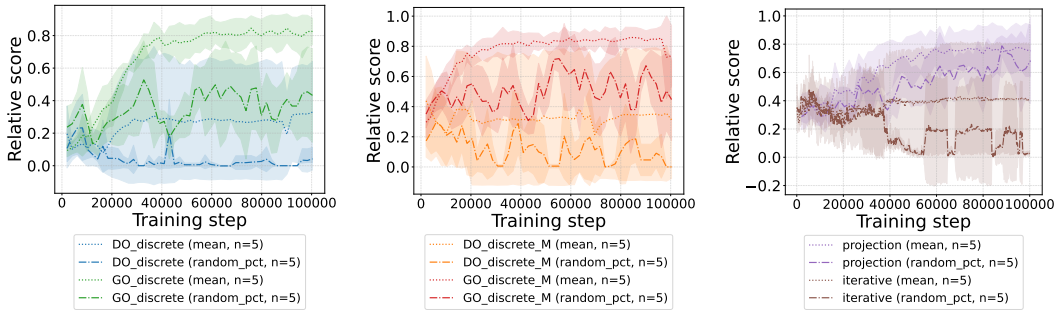


(d) Cyber-Path Discrete M/V (e) Cyber-Path Discrete-M M/V (f) Cyber-Path Latent M/V

Figure 13: Average learning curves (mean ± standard deviation) across Cyber-Path benchmark environments under all training configurations.

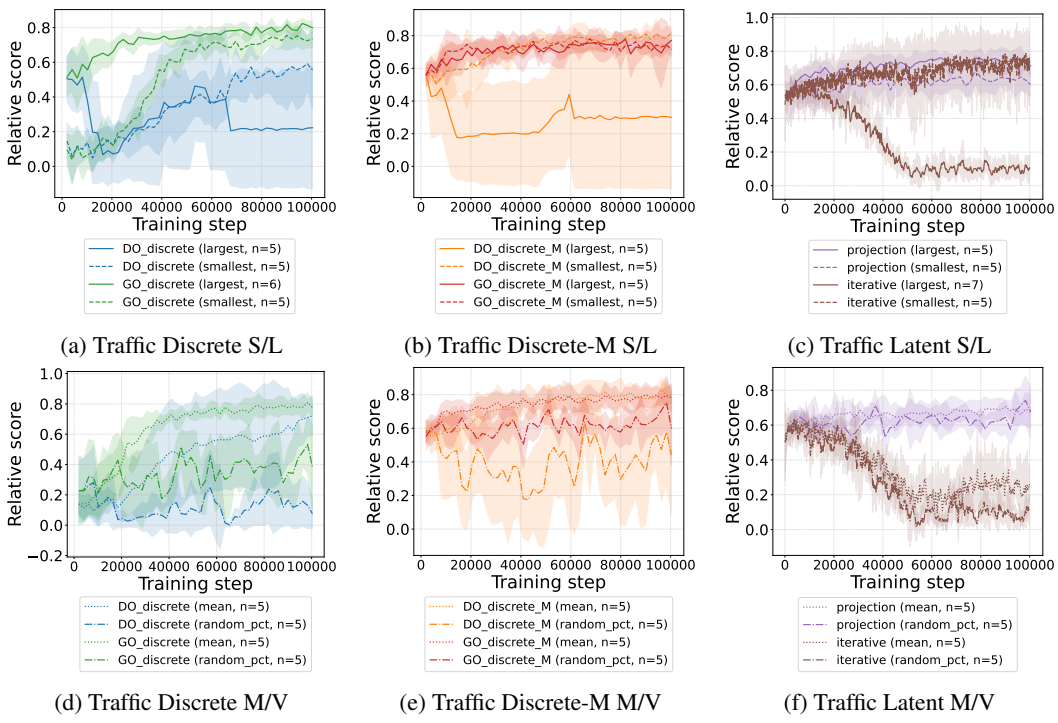


(a) OSPF Discrete S/L (b) OSPF Discrete-M S/L (c) OSPF Latent S/L



(d) OSPF Discrete M/V (e) OSPF Discrete-M M/V (f) OSPF Latent M/V

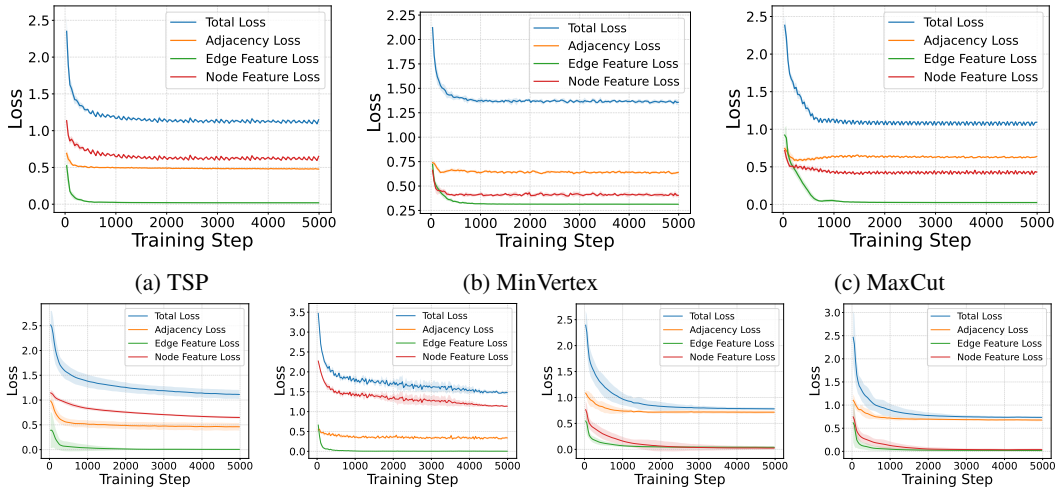
Figure 14: Average learning curves (mean ± standard deviation) across OSPF benchmark environments under all training configurations.



(a) Traffic Discrete S/L (b) Traffic Discrete-M S/L (c) Traffic Latent S/L

(d) Traffic Discrete M/V (e) Traffic Discrete-M M/V (f) Traffic Latent M/V

Figure 15: Average learning curves (mean ± standard deviation) across Traffic benchmark environments under all training configurations.



(a) TSP (b) MinVertex (c) MaxCut

(d) Placement (e) Cyber-Path (f) OSPF Engineering (g) Traffic Engineering

Figure 16: Graph autoencoder learning curves for five samples of 20 nodes across the seven benchmarks.

K TRAINING TIME

We report the computational training cost of the considered approaches on the most demanding benchmark, *Traffic*.

We report the comparison using the most demanding training strategy, the *varied* (V) training strategy, in which agents are exposed to multiple training environments, resulting in the most costly strategy in terms of training time.

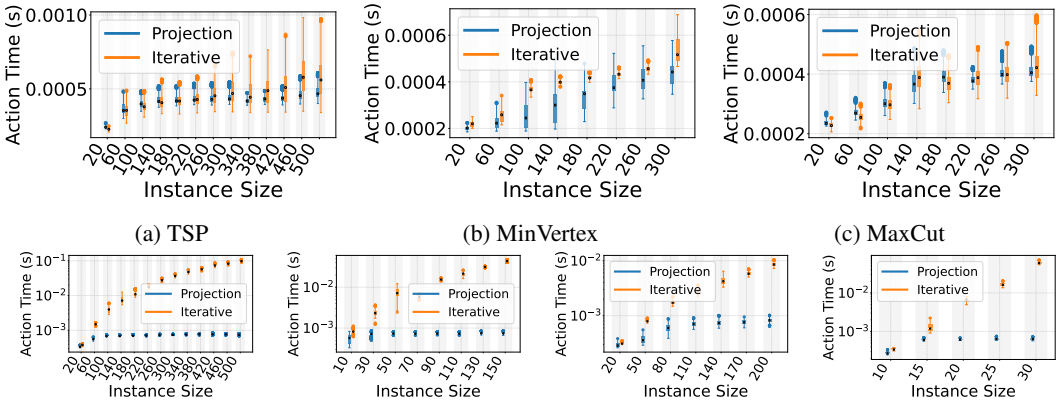
Table 10: Training time (seconds) across 5 runs on the traffic engineering benchmark.

Method	Run 1	Run 2	Run 3	Run 4	Run 5	Mean ± Std
P-discrete	671	678	675	675	684	676 ± 5
P-discrete-M	779	778	805	783	792	787 ± 10
G-discrete	1198	1084	3998	1300	1239	1764 ± 1100
G-discrete-M	1738	1614	5393	1759	1739	2449 ± 1600
Iterative	3776	2717	6718	2570	5095	4175 ± 1500
Projection	5844	3384	13417	5788	4869	7460 ± 3800

Results computed on the benchmark machine used for the generalization study[4] are reported in Table 10 over 5 independent runs, each using 20 training environments and 81 test environments.
In general, *projection* methods consistently exhibit the highest computational cost at training time, followed by the *iterative* approach, while *discrete* baselines remain significantly more efficient in the training stage, as expected.

L SCALABILITY CURVES

Figures 17 present the full evolution of the action-selection time experiments in Section 4.2 (evaluation time), including boxplots of the action time distribution across different graph sizes. The figures also indicate the sampled graph sizes for each benchmark, including the maximum size considered to perform the power-law estimation.



(a) TSP (b) MinVertex (c) MaxCut

(d) Placement (e) Cyber-Path (f) OSPF (g) Traffic

Figure 17: Scalability curve for seven experimental settings.

L.1 APPROXIMATE NEAREST-NEIGHBOR LOOKUP

To further assess the scalability of the *projection* approach, we evaluate FAISS IVF (Inverted File Index), an approximate NN method that partitions the embedding space into clusters and searches only a subset of relevant clusters, reducing retrieval cost compared with exhaustive search. IVF is primarily designed for static embedding sets; however, in our setting, action embeddings evolve throughout an episode because the current partial solution is encoded directly in the graph state (as discussed in Section 5). Consequently, graph updates modify the action embeddings, requiring the IVF index to be rebuilt or updated after each modification step. Thus, our evaluation does not represent the ideal use case for IVF, but nevertheless demonstrates a measurable speedup.
Table 11 reports the mean action-selection time for *TSP* across increasing graph sizes. Despite the additional index reconstruction overhead, IVF consistently reduces action time compared with the FAISS Flat Index. The advantage becomes increasingly pronounced at larger graph sizes, with IVF reducing the action time from 0.49 ms/action to 0.40 ms/action at 500 nodes.

[4] *Benchmark Machine (generalization study):* Dual Intel Xeon Gold 6258R (2 × 28 cores, 112 threads total) @ 2.70 GHz, 503 GiB RAM, NVIDIA RTX A6000 GPU, Ubuntu 22.04.3 LTS.

Table 11: Mean action-selection time (ms/action) for *TSP* using FAISS Flat and IVF indexes across graph sizes.

Size	20	60	100	140	180	220	260	300	340	380	420	460	500
Flat	0.25	0.37	0.41	0.44	0.44	0.45	0.45	0.46	0.42	0.43	0.46	0.48	0.49
IVF	0.23	0.26	0.30	0.33	0.35	0.35	0.36	0.37	0.37	0.38	0.38	0.38	0.40

These results suggest that approximate NN search can further improve the scalability of *Projection*, even when accounting for the additional cost of maintaining a dynamic index. A more effective use of sub-linear NN indexes could be enabled by decoupling the static graph representation from the evolving partial solution, such that action embeddings remain fixed throughout an episode. This would eliminate frequent index reconstruction and is a promising direction for future work, as discussed in the revised Limitations section.

M Extended generalization study

This section extends the generalization study by reporting additional statistical metrics in Tables 12–15. These include the mean performance, asymmetric spread estimates based on percentiles (lower spread: mean 16th percentile; upper spread: 84th percentile mean), Shapiro–Wilk normality test results, and bootstrapped confidence intervals (BCIs) of the normalized scores. These metrics provide a more reliable and robust estimate of performance across benchmarks and training strategies. The observed variability reflects differences in policy initialization, scenario splits, and the retraining of the GAE, which is performed independently for each training strategy.
In addition, Tables 16–19 provide further analysis logs with the complete evaluation setting without the best selection per environment, aggregating the results for all five runs to highlight the differences induced by the evaluation protocol when this selection step is removed, showcasing that the ranking and the conclusions among parameterization strategies remain consistent.

Regime	Metric	P-Discrete	P-Discrete-M	G-Discrete	G-Discrete-M	Iterative	Projection (ours)
				TSP			
S	*Mean*	**0.02**	**0.51**	**0.00**	**0.51**	**0.92**	**0.79**
	Std	-0.02/+-0.02	-0.06/+0.06	-0.00/+0.00	-0.06/+0.05	-0.26/+0.28	-0.18/+0.18
	IQM	0.00	0.51	0.00	0.51	0.91	0.79
	BCI	[0.01, 0.03]	[0.50, 0.51]	[0.00, 0.00]	[0.51, 0.52]	[0.90, 0.94]	[0.78, 0.81]
	Normal	$\times(p = 0.00)$	$\times(p = 0.01)$	✓(p=1.00)	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
L	*Mean*	**0.00**	**0.48**	**0.00**	**0.51**	**0.97**	**0.62**
	Std	-0.00/+0.00	-0.07/+0.06	-0.00/+-0.00	-0.06/+0.06	-0.26/+0.25	-0.11/+0.12
	IQM	0.00	0.48	0.00	0.51	0.98	0.61
	BCI	[0.00, 0.00]	[0.48, 0.49]	[0.00, 0.00]	[0.50, 0.51]	[0.95, 0.99]	[0.61, 0.63]
	Normal	✓(p=1.00)	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
M	*Mean*	**0.00**	**0.48**	**0.00**	**0.51**	**0.86**	**0.76**
	Std	-0.00/+0.00	-0.07/+0.07	-0.00/+0.00	-0.06/+0.06	-0.32/+0.39	-0.22/+0.20
	IQM	0.00	0.48	0.00	0.51	0.80	0.76
	BCI	[0.00, 0.00]	[0.47, 0.49]	[0.00, 0.00]	[0.51, 0.52]	[0.83, 0.89]	[0.74, 0.78]
	Normal	✓(p=1.00)	$\times(p = 0.01)$	✓(p=1.00)	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
V	*Mean*	**0.00**	**0.49**	**0.00**	**0.51**	**1.01**	**0.73**
	Std	-0.00/+0.00	-0.07/+0.06	-0.00/+0.00	-0.05/+0.04	-0.26/+0.26	-0.17/+0.16
	IQM	0.00	0.48	0.00	0.51	1.00	0.72
	BCI	[0.00, 0.00]	[0.48, 0.49]	[0.00, 0.00]	[0.51, 0.52]	[0.98, 1.03]	[0.71, 0.74]
	Normal	✓(p=1.00)	$\times(p = 0.00)$	✓(p=1.00)	$\times(p = 0.00)$	✓(p=0.07)	$\times(p = 0.00)$
				MinVertex			
S	*Mean*	**0.04**	**0.40**	**0.00**	**0.35**	**0.55**	**0.68**
	Std	-0.04/+-0.04	-0.15/+0.16	-0.00/+-0.00	-0.18/+0.15	-0.55/+0.45	-0.30/+0.32
	IQM	0.00	0.39	0.00	0.33	0.40	0.70
	BCI	[0.02, 0.05]	[0.39, 0.41]	[0.00, 0.01]	[0.33, 0.36]	[0.51, 0.59]	[0.66, 0.71]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
L	*Mean*	**0.09**	**0.41**	**0.24**	**0.39**	**0.15**	**0.17**
	Std	-0.09/+-0.07	-0.16/+0.17	-0.24/+0.33	-0.16/+0.17	-0.15/+0.32	-0.17/+0.33
	IQM	0.00	0.40	0.13	0.36	0.00	0.04
	BCI	[0.07, 0.11]	[0.40, 0.42]	[0.22, 0.27]	[0.37, 0.40]	[0.12, 0.18]	[0.15, 0.20]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
M	*Mean*	**0.06**	**0.39**	**0.15**	**0.37**	**0.26**	**0.20**
	Std	-0.06/+-0.06	-0.16/+0.17	-0.15/+0.35	-0.17/+0.18	-0.26/+0.61	-0.20/+0.55
	IQM	0.00	0.36	0.00	0.36	0.06	0.02
	BCI	[0.05, 0.08]	[0.37, 0.41]	[0.13, 0.17]	[0.36, 0.39]	[0.23, 0.30]	[0.17, 0.24]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
V	*Mean*	**0.27**	**0.42**	**0.18**	**0.40**	**0.35**	**0.17**
	Std	-0.27/+0.28	-0.15/+0.18	-0.18/+0.37	-0.15/+0.16	-0.35/+0.54	-0.17/+0.33
	IQM	0.17	0.40	0.03	0.39	0.16	0.03
	BCI	[0.25, 0.30]	[0.40, 0.43]	[0.15, 0.21]	[0.38, 0.41]	[0.31, 0.39]	[0.14, 0.19]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$

Table 12: Extended generalization results with confidence intervals and Shapiro-Wilk normality checks for TSP and MinVertex.

Regime	Metric	P-Discrete	P-Discrete-M	G-Discrete	G-Discrete-M	Iterative	Projection (ours)
				MaxCut			
S	*Mean*	**0.90**	**0.91**	**0.20**	**0.20**	**0.91**	**0.92**
	Std	-0.02/+0.02	-0.02/+0.02	-0.20/+0.22	-0.20/+0.19	-0.06/+0.05	-0.05/+0.04
	IQM	0.91	0.91	0.08	0.08	0.92	0.93
	BCI	[0.90, 0.90]	[0.91, 0.91]	[0.18, 0.23]	[0.18, 0.22]	[0.91, 0.92]	[0.92, 0.93]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
L	*Mean*	**0.27**	**0.88**	**0.90**	**0.91**	**0.48**	**0.91**
	Std	-0.27/+0.60	-0.04/+0.04	-0.02/+0.02	-0.02/+0.02	-0.48/+0.45	-0.03/+0.03
	IQM	0.07	0.88	0.90	0.91	0.33	0.91
	BCI	[0.24, 0.31]	[0.87, 0.88]	[0.90, 0.90]	[0.90, 0.91]	[0.45, 0.52]	[0.91, 0.91]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
M	*Mean*	**0.56**	**0.88**	**0.88**	**0.89**	**0.41**	**0.94**
	Std	-0.56/+0.35	-0.04/+0.04	-0.03/+0.04	-0.03/+0.03	-0.41/+0.53	-0.02/+0.02
	IQM	0.44	0.89	0.90	0.90	0.23	0.95
	BCI	[0.52, 0.59]	[0.88, 0.88]	[0.87, 0.88]	[0.89, 0.89]	[0.38, 0.45]	[0.94, 0.94]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
V	*Mean*	**0.69**	**0.87**	**0.87**	**0.90**	**0.39**	**0.93**
	Std	-0.69/+0.22	-0.04/+0.04	-0.02/+0.04	-0.02/+0.02	-0.39/+0.53	-0.02/+0.02
	IQM	0.88	0.88	0.89	0.90	0.21	0.94
	BCI	[0.66, 0.73]	[0.87, 0.88]	[0.87, 0.88]	[0.89, 0.90]	[0.35, 0.43]	[0.93, 0.93]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
				Placement			
S	*Mean*	**0.64**	**0.71**	**0.44**	**0.76**	**0.28**	**0.79**
	Std	-0.21/+0.22	-0.15/+0.15	-0.39/+0.39	-0.23/+0.22	-0.26/+0.26	-0.21/+0.20
	IQM	0.67	0.73	0.43	0.79	0.26	0.80
	BCI	[0.62, 0.66]	[0.70, 0.73]	[0.42, 0.47]	[0.73, 0.78]	[0.26, 0.31]	[0.77, 0.81]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.01)$
L	*Mean*	**0.58**	**0.70**	**0.63**	**0.73**	**0.26**	**0.76**
	Std	-0.17/+0.16	-0.12/+0.14	-0.19/+0.21	-0.16/+0.16	-0.24/+0.26	-0.18/+0.17
	IQM	0.58	0.71	0.66	0.74	0.21	0.77
	BCI	[0.57, 0.59]	[0.69, 0.71]	[0.61, 0.65]	[0.72, 0.75]	[0.24, 0.28]	[0.75, 0.78]
	Normal	✓(p=0.09)	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	✓(p=0.29)
M	*Mean*	**0.57**	**0.73**	**0.52**	**0.75**	**0.12**	**0.78**
	Std	-0.21/+0.21	-0.14/+0.14	-0.47/+0.31	-0.17/+0.18	-0.12/+0.17	-0.22/+0.21
	IQM	0.59	0.74	0.59	0.76	0.02	0.79
	BCI	[0.55, 0.59]	[0.72, 0.74]	[0.49, 0.55]	[0.73, 0.77]	[0.10, 0.14]	[0.76, 0.80]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.01)$
V	*Mean*	**0.60**	**0.71**	**0.67**	**0.72**	**0.35**	**0.78**
	Std	-0.16/+0.17	-0.13/+0.14	-0.21/+0.22	-0.14/+0.14	-0.34/+0.37	-0.19/+0.17
	IQM	0.60	0.72	0.69	0.72	0.30	0.79
	BCI	[0.58, 0.62]	[0.69, 0.72]	[0.65, 0.69]	[0.70, 0.73]	[0.32, 0.38]	[0.77, 0.80]
	Normal	✓(p=0.30)	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.03)$	$\times(p = 0.00)$	✓(p=0.09)

Table 13: Extended generalization results with confidence intervals and Shapiro-Wilk normality checks for MaxCut and Placement.

Regime	Metric	P-Discrete	P-Discrete-M	G-Discrete	G-Discrete-M	Iterative	Projection (ours)
				Cyber-Path			
S	*Mean*	**0.21**	**0.62**	**0.19**	**0.47**	**0.36**	**0.61**
	Std	-0.10/+0.10	-0.12/+0.13	-0.09/+0.08	-0.17/+0.18	-0.23/+0.20	-0.11/+0.11
	IQM	0.19	0.61	0.18	0.45	0.35	0.61
	BCI	[0.20, 0.22]	[0.61, 0.63]	[0.19, 0.20]	[0.45, 0.48]	[0.34, 0.38]	[0.60, 0.62]
	Normal	$\times(p = 0.00)$	$\times(p = 0.03)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	✓(p=0.23)
L	*Mean*	**0.21**	**0.61**	**0.21**	**0.52**	**0.30**	**0.61**
	Std	-0.09/+0.09	-0.11/+0.14	-0.10/+0.09	-0.15/+0.15	-0.22/+0.39	-0.11/+0.11
	IQM	0.19	0.61	0.18	0.51	0.20	0.61
	BCI	[0.20, 0.22]	[0.60, 0.62]	[0.20, 0.21]	[0.50, 0.53]	[0.27, 0.32]	[0.60, 0.62]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	✓(p=0.13)	$\times(p = 0.00)$	✓(p=0.09)
M	*Mean*	**0.21**	**0.64**	**0.20**	**0.47**	**0.31**	**0.64**
	Std	-0.10/+0.09	-0.11/+0.11	-0.09/+0.08	-0.17/+0.17	-0.20/+0.29	-0.09/+0.10
	IQM	0.19	0.64	0.19	0.46	0.25	0.64
	BCI	[0.20, 0.21]	[0.63, 0.65]	[0.19, 0.20]	[0.45, 0.48]	[0.29, 0.33]	[0.63, 0.65]
	Normal	$\times(p = 0.00)$	$\times(p = 0.01)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
V	*Mean*	**0.20**	**0.68**	**0.19**	**0.51**	**0.28**	**0.68**
	Std	-0.10/+0.11	-0.13/+0.13	-0.09/+0.10	-0.18/+0.15	-0.12/+0.12	-0.10/+0.10
	IQM	0.18	0.68	0.18	0.51	0.25	0.68
	BCI	[0.19, 0.21]	[0.67, 0.69]	[0.18, 0.20]	[0.49, 0.52]	[0.27, 0.29]	[0.67, 0.69]
	Normal	$\times(p = 0.00)$	✓(p=0.07)	$\times(p = 0.00)$	$\times(p = 0.01)$	$\times(p = 0.00)$	$\times(p = 0.00)$
				OSPF			
S	*Mean*	**0.24**	**0.86**	**0.37**	**0.50**	**0.92**	**0.94**
	Std	-0.24/+0.37	-0.07/+0.09	-0.37/+0.30	-0.38/+0.33	-0.09/+0.08	-0.06/+0.06
	IQM	0.08	0.86	0.37	0.51	0.94	0.95
	BCI	[0.21, 0.26]	[0.85, 0.87]	[0.35, 0.40]	[0.47, 0.52]	[0.91, 0.93]	[0.93, 0.94]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
L	*Mean*	**0.16**	**0.84**	**0.33**	**0.73**	**0.90**	**0.89**
	Std	-0.16/+0.24	-0.09/+0.11	-0.33/+0.33	-0.19/+0.18	-0.09/+0.10	-0.07/+0.11
	IQM	0.03	0.85	0.19	0.75	0.91	0.89
	BCI	[0.14, 0.18]	[0.83, 0.85]	[0.30, 0.35]	[0.71, 0.74]	[0.89, 0.91]	[0.88, 0.90]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
M	*Mean*	**0.27**	**0.84**	**0.33**	**0.56**	**0.91**	**0.92**
	Std	-0.27/+0.41	-0.10/+0.11	-0.33/+0.33	-0.33/+0.26	-0.09/+0.09	-0.07/+0.08
	IQM	0.12	0.86	0.20	0.59	0.93	0.92
	BCI	[0.24, 0.30]	[0.83, 0.86]	[0.31, 0.36]	[0.54, 0.59]	[0.90, 0.92]	[0.91, 0.92]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
V	*Mean*	**0.14**	**0.80**	**0.58**	**0.75**	**0.91**	**0.90**
	Std	-0.14/+0.30	-0.12/+0.12	-0.28/+0.25	-0.16/+0.15	-0.08/+0.09	-0.07/+0.10
	IQM	0.01	0.82	0.60	0.76	0.93	0.91
	BCI	[0.12, 0.16]	[0.78, 0.81]	[0.56, 0.61]	[0.73, 0.76]	[0.90, 0.92]	[0.90, 0.91]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$

Table 14: Extended generalization results with confidence intervals and Shapiro-Wilk normality checks for Cyber-Path and OSPF.

Regime	Metric	P-Discrete	P-Discrete-M	G-Discrete	G-Discrete-M	Iterative	Projection (ours)
				Traffic			
S	*Mean*	**0.24**	**0.82**	**0.30**	**0.64**	**0.63**	**0.77**
	Std	-0.24/+0.31	-0.07/+0.07	-0.28/+0.33	-0.25/+0.20	-0.26/+0.25	-0.11/+0.13
	IQM	0.12	0.81	0.24	0.69	0.67	0.79
	BCI	[0.22, 0.27]	[0.81, 0.82]	[0.28, 0.33]	[0.62, 0.66]	[0.60, 0.65]	[0.76, 0.79]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
L	*Mean*	**0.28**	**0.81**	**0.49**	**0.79**	**0.44**	**0.83**
	Std	-0.28/+0.28	-0.08/+0.08	-0.15/+0.17	-0.09/+0.08	-0.29/+0.31	-0.08/+0.08
	IQM	0.17	0.81	0.49	0.79	0.49	0.83
	BCI	[0.26, 0.31]	[0.80, 0.82]	[0.48, 0.51]	[0.78, 0.80]	[0.41, 0.47]	[0.82, 0.84]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	✓(p=0.05)	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
M	*Mean*	**0.45**	**0.80**	**0.55**	**0.64**	**0.52**	**0.85**
	Std	-0.37/+0.26	-0.09/+0.09	-0.26/+0.23	-0.33/+0.20	-0.25/+0.24	-0.08/+0.08
	IQM	0.49	0.80	0.57	0.72	0.55	0.85
	BCI	[0.43, 0.48]	[0.79, 0.81]	[0.53, 0.57]	[0.62, 0.66]	[0.49, 0.54]	[0.84, 0.86]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
V	*Mean*	**0.37**	**0.82**	**0.55**	**0.78**	**0.58**	**0.85**
	Std	-0.37/+0.30	-0.08/+0.07	-0.23/+0.23	-0.09/+0.09	-0.25/+0.25	-0.08/+0.08
	IQM	0.38	0.81	0.58	0.78	0.61	0.84
	BCI	[0.35, 0.40]	[0.81, 0.82]	[0.53, 0.58]	[0.77, 0.79]	[0.55, 0.61]	[0.84, 0.85]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$

Table 15: Extended generalization results with confidence intervals and Shapiro-Wilk normality checks for Traffic.

Regime	Metric	P-Discrete	P-Discrete-M	G-Discrete	G-Discrete-M	Iterative	Projection (ours)
				TSP			
S	*Mean*	**0.01**	**0.43**	**0.00**	**0.43**	**0.87**	**0.71**
	Std	-0.01/+-0.01	-0.08/+0.08	-0.00/+0.00	-0.08/+0.08	-0.27/+0.29	-0.19/+0.19
	IQM	0.00	0.43	0.00	0.43	0.86	0.71
	BCI	[0.01, 0.02]	[0.43, 0.43]	[0.00, 0.00]	[0.43, 0.43]	[0.86, 0.88]	[0.70, 0.72]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	✓(p=1.00)	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
L	*Mean*	**0.00**	**0.43**	**0.00**	**0.43**	**0.92**	**0.52**
	Std	-0.00/+0.00	-0.08/+0.07	-0.00/+-0.00	-0.07/+0.07	-0.27/+0.26	-0.13/+0.14
	IQM	0.00	0.43	0.00	0.43	0.93	0.51
	BCI	[0.00, 0.00]	[0.42, 0.43]	[0.00, 0.00]	[0.42, 0.43]	[0.91, 0.93]	[0.51, 0.52]
	Normal	✓(p=1.00)	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
M	*Mean*	**0.00**	**0.43**	**0.00**	**0.43**	**0.81**	**0.67**
	Std	-0.00/+0.00	-0.07/+0.07	-0.00/+0.00	-0.08/+0.07	-0.34/+0.42	-0.23/+0.21
	IQM	0.00	0.43	0.00	0.43	0.75	0.68
	BCI	[0.00, 0.00]	[0.42, 0.43]	[0.00, 0.00]	[0.42, 0.43]	[0.79, 0.82]	[0.66, 0.68]
	Normal	✓(p=1.00)	$\times(p = 0.00)$	✓(p=1.00)	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
V	*Mean*	**0.00**	**0.43**	**0.00**	**0.43**	**0.96**	**0.64**
	Std	-0.00/+0.00	-0.08/+0.08	-0.00/+0.00	-0.07/+0.08	-0.28/+0.27	-0.20/+0.19
	IQM	0.00	0.43	0.00	0.43	0.96	0.64
	BCI	[0.00, 0.00]	[0.42, 0.43]	[0.00, 0.00]	[0.43, 0.43]	[0.95, 0.97]	[0.63, 0.64]
	Normal	✓(p=1.00)	$\times(p = 0.00)$	✓(p=1.00)	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
				MinVertex			
S	*Mean*	**0.01**	**0.26**	**0.00**	**0.26**	**0.55**	**0.57**
	Std	-0.01/+-0.01	-0.15/+0.14	-0.00/+-0.00	-0.14/+0.14	-0.55/+0.45	-0.32/+0.31
	IQM	0.00	0.23	0.00	0.23	0.40	0.56
	BCI	[0.01, 0.02]	[0.26, 0.27]	[0.00, 0.00]	[0.26, 0.27]	[0.53, 0.57]	[0.56, 0.58]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
L	*Mean*	**0.03**	**0.27**	**0.10**	**0.27**	**0.15**	**0.09**
	Std	-0.03/+-0.03	-0.14/+0.15	-0.10/+0.23	-0.14/+0.13	-0.15/+0.32	-0.09/+0.16
	IQM	0.00	0.24	0.00	0.24	0.00	0.01
	BCI	[0.02, 0.03]	[0.26, 0.27]	[0.10, 0.11]	[0.26, 0.27]	[0.14, 0.16]	[0.09, 0.10]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
M	*Mean*	**0.02**	**0.27**	**0.06**	**0.27**	**0.26**	**0.13**
	Std	-0.02/+-0.02	-0.16/+0.17	-0.06/+-0.06	-0.15/+0.13	-0.26/+0.61	-0.13/+0.32
	IQM	0.00	0.25	0.00	0.24	0.06	0.00
	BCI	[0.02, 0.03]	[0.27, 0.28]	[0.05, 0.07]	[0.26, 0.28]	[0.25, 0.28]	[0.12, 0.14]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
V	*Mean*	**0.13**	**0.27**	**0.08**	**0.27**	**0.35**	**0.09**
	Std	-0.13/+0.25	-0.14/+0.13	-0.08/+-0.08	-0.14/+0.13	-0.35/+0.55	-0.09/+0.11
	IQM	0.02	0.24	0.00	0.24	0.16	0.00
	BCI	[0.12, 0.14]	[0.26, 0.27]	[0.07, 0.09]	[0.26, 0.27]	[0.33, 0.37]	[0.08, 0.09]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$

Table 16: Extended generalization results with confidence intervals and Shapiro-Wilk normality checks for TSP and MinVertex without the best selection per environment.

Regime	Metric	P-Discrete	P-Discrete-M	G-Discrete	G-Discrete-M	Iterative	Projection (ours)
				MaxCut			
S	*Mean*	**0.87**	**0.88**	**0.16**	**0.15**	**0.91**	**0.91**
	Std	-0.05/+0.04	-0.03/+0.04	-0.16/+0.19	-0.15/+0.16	-0.06/+0.05	-0.05/+0.05
	IQM	0.89	0.89	0.04	0.05	0.92	0.92
	BCI	[0.87, 0.87]	[0.88, 0.88]	[0.15, 0.16]	[0.15, 0.16]	[0.91, 0.91]	[0.91, 0.91]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
L	*Mean*	**0.24**	**0.86**	**0.85**	**0.87**	**0.48**	**0.88**
	Std	-0.24/+0.58	-0.05/+0.05	-0.05/+0.06	-0.04/+0.04	-0.48/+0.45	-0.05/+0.05
	IQM	0.05	0.88	0.87	0.88	0.33	0.89
	BCI	[0.23, 0.26]	[0.86, 0.87]	[0.85, 0.85]	[0.86, 0.87]	[0.47, 0.50]	[0.88, 0.88]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
M	*Mean*	**0.52**	**0.85**	**0.79**	**0.83**	**0.41**	**0.92**
	Std	-0.52/+0.37	-0.07/+0.06	-0.12/+0.10	-0.07/+0.08	-0.41/+0.53	-0.03/+0.03
	IQM	0.39	0.87	0.84	0.86	0.23	0.93
	BCI	[0.50, 0.53]	[0.85, 0.85]	[0.79, 0.80]	[0.82, 0.83]	[0.40, 0.43]	[0.92, 0.92]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
V	*Mean*	**0.65**	**0.84**	**0.79**	**0.84**	**0.39**	**0.91**
	Std	-0.65/+0.25	-0.06/+0.06	-0.08/+0.10	-0.06/+0.06	-0.39/+0.53	-0.04/+0.04
	IQM	0.82	0.86	0.84	0.87	0.21	0.92
	BCI	[0.63, 0.67]	[0.83, 0.84]	[0.79, 0.80]	[0.84, 0.85]	[0.37, 0.41]	[0.90, 0.91]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
				Placement			
S	*Mean*	**0.51**	**0.58**	**0.35**	**0.66**	**0.28**	**0.69**
	Std	-0.25/+0.24	-0.19/+0.19	-0.34/+0.35	-0.28/+0.26	-0.26/+0.26	-0.24/+0.23
	IQM	0.53	0.59	0.31	0.70	0.26	0.70
	BCI	[0.50, 0.52]	[0.57, 0.59]	[0.34, 0.37]	[0.65, 0.67]	[0.27, 0.29]	[0.68, 0.70]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
L	*Mean*	**0.43**	**0.56**	**0.49**	**0.61**	**0.26**	**0.62**
	Std	-0.20/+0.19	-0.18/+0.18	-0.25/+0.24	-0.19/+0.20	-0.24/+0.26	-0.21/+0.20
	IQM	0.42	0.56	0.51	0.61	0.21	0.62
	BCI	[0.42, 0.44]	[0.56, 0.57]	[0.48, 0.50]	[0.60, 0.61]	[0.25, 0.27]	[0.61, 0.63]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.04)$
M	*Mean*	**0.44**	**0.60**	**0.41**	**0.63**	**0.12**	**0.66**
	Std	-0.23/+0.23	-0.19/+0.19	-0.39/+0.30	-0.20/+0.22	-0.12/+0.17	-0.25/+0.25
	IQM	0.44	0.60	0.43	0.64	0.02	0.66
	BCI	[0.43, 0.45]	[0.59, 0.61]	[0.40, 0.42]	[0.62, 0.64]	[0.11, 0.13]	[0.65, 0.67]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
V	*Mean*	**0.45**	**0.57**	**0.55**	**0.58**	**0.35**	**0.65**
	Std	-0.20/+0.20	-0.18/+0.18	-0.25/+0.23	-0.18/+0.19	-0.34/+0.38	-0.20/+0.20
	IQM	0.45	0.58	0.56	0.58	0.30	0.65
	BCI	[0.44, 0.46]	[0.56, 0.58]	[0.54, 0.56]	[0.57, 0.59]	[0.34, 0.37]	[0.65, 0.66]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.01)$	$\times(p = 0.00)$	$\times(p = 0.01)$

Table 17: Extended generalization results with confidence intervals and Shapiro-Wilk normality checks for MaxCut and Placement without the best selection per environment.

Regime	Metric	P-Discrete	P-Discrete-M	G-Discrete	G-Discrete-M	Iterative	Projection (ours)
				Cyber-Path			
S	*Mean*	**0.11**	**0.44**	**0.11**	**0.29**	**0.26**	**0.47**
	Std	-0.05/+0.05	-0.19/+0.18	-0.05/+0.05	-0.17/+0.18	-0.19/+0.19	-0.15/+0.16
	IQM	0.08	0.44	0.08	0.25	0.21	0.48
	BCI	[0.11, 0.11]	[0.43, 0.44]	[0.10, 0.11]	[0.28, 0.29]	[0.25, 0.27]	[0.46, 0.48]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
L	*Mean*	**0.11**	**0.43**	**0.11**	**0.32**	**0.23**	**0.46**
	Std	-0.05/+0.06	-0.21/+0.18	-0.05/+0.06	-0.19/+0.18	-0.17/+0.24	-0.16/+0.15
	IQM	0.09	0.44	0.09	0.29	0.12	0.47
	BCI	[0.11, 0.11]	[0.43, 0.44]	[0.11, 0.11]	[0.31, 0.33]	[0.22, 0.24]	[0.46, 0.47]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
M	*Mean*	**0.11**	**0.46**	**0.11**	**0.29**	**0.23**	**0.49**
	Std	-0.05/+0.05	-0.17/+0.16	-0.05/+0.06	-0.18/+0.21	-0.16/+0.23	-0.16/+0.15
	IQM	0.08	0.47	0.09	0.26	0.16	0.51
	BCI	[0.10, 0.11]	[0.45, 0.47]	[0.11, 0.11]	[0.29, 0.30]	[0.22, 0.24]	[0.49, 0.50]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
V	*Mean*	**0.11**	**0.47**	**0.10**	**0.33**	**0.20**	**0.53**
	Std	-0.05/+0.05	-0.20/+0.19	-0.05/+0.05	-0.18/+0.17	-0.10/+0.10	-0.14/+0.15
	IQM	0.08	0.48	0.08	0.31	0.18	0.54
	BCI	[0.10, 0.11]	[0.47, 0.48]	[0.10, 0.11]	[0.32, 0.34]	[0.19, 0.20]	[0.53, 0.54]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
				OSPF			
S	*Mean*	**0.16**	**0.78**	**0.24**	**0.39**	**0.92**	**0.87**
	Std	-0.16/+0.21	-0.12/+0.12	-0.24/+0.25	-0.33/+0.33	-0.09/+0.08	-0.12/+0.13
	IQM	0.04	0.79	0.19	0.36	0.94	0.89
	BCI	[0.15, 0.17]	[0.77, 0.78]	[0.23, 0.25]	[0.37, 0.40]	[0.92, 0.92]	[0.86, 0.87]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
L	*Mean*	**0.08**	**0.77**	**0.19**	**0.57**	**0.90**	**0.79**
	Std	-0.08/+0.10	-0.12/+0.12	-0.19/+0.22	-0.24/+0.23	-0.09/+0.10	-0.13/+0.13
	IQM	0.00	0.78	0.08	0.58	0.91	0.80
	BCI	[0.07, 0.09]	[0.77, 0.78]	[0.18, 0.20]	[0.56, 0.58]	[0.90, 0.90]	[0.78, 0.79]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
M	*Mean*	**0.18**	**0.78**	**0.22**	**0.44**	**0.91**	**0.83**
	Std	-0.18/+0.27	-0.12/+0.13	-0.22/+0.26	-0.30/+0.29	-0.09/+0.09	-0.12/+0.13
	IQM	0.05	0.80	0.10	0.43	0.93	0.85
	BCI	[0.17, 0.19]	[0.78, 0.79]	[0.21, 0.23]	[0.43, 0.45]	[0.91, 0.92]	[0.83, 0.84]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
V	*Mean*	**0.08**	**0.77**	**0.42**	**0.60**	**0.91**	**0.80**
	Std	-0.08/+0.10	-0.12/+0.13	-0.28/+0.27	-0.21/+0.19	-0.08/+0.09	-0.12/+0.13
	IQM	0.00	0.79	0.40	0.61	0.93	0.82
	BCI	[0.07, 0.09]	[0.77, 0.78]	[0.41, 0.43]	[0.60, 0.61]	[0.91, 0.92]	[0.80, 0.81]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$

Table 18: Extended generalization results with confidence intervals and Shapiro-Wilk normality checks for Cyber-Path and OSPF without the best selection per environment.

Regime	Metric	P-Discrete	P-Discrete-M	G-Discrete	G-Discrete-M	Iterative	Projection (ours)
				Traffic			
S	*Mean*	**0.17**	**0.69**	**0.23**	**0.48**	**0.63**	**0.64**
	Std	-0.17/+0.23	-0.14/+0.13	-0.23/+0.26	-0.31/+0.26	-0.26/+0.25	-0.19/+0.18
	IQM	0.07	0.70	0.17	0.51	0.67	0.67
	BCI	[0.16, 0.18]	[0.68, 0.69]	[0.22, 0.24]	[0.47, 0.49]	[0.62, 0.64]	[0.63, 0.65]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
L	*Mean*	**0.15**	**0.69**	**0.28**	**0.65**	**0.44**	**0.71**
	Std	-0.15/+0.22	-0.14/+0.13	-0.25/+0.22	-0.16/+0.15	-0.29/+0.31	-0.13/+0.13
	IQM	0.05	0.70	0.27	0.66	0.49	0.72
	BCI	[0.14, 0.16]	[0.68, 0.69]	[0.27, 0.29]	[0.64, 0.65]	[0.43, 0.46]	[0.70, 0.72]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
M	*Mean*	**0.30**	**0.69**	**0.40**	**0.51**	**0.52**	**0.73**
	Std	-0.30/+0.29	-0.14/+0.13	-0.27/+0.26	-0.32/+0.24	-0.25/+0.24	-0.13/+0.14
	IQM	0.27	0.70	0.40	0.55	0.55	0.74
	BCI	[0.29, 0.31]	[0.68, 0.69]	[0.39, 0.41]	[0.50, 0.52]	[0.51, 0.53]	[0.72, 0.74]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$
V	*Mean*	**0.24**	**0.69**	**0.39**	**0.64**	**0.58**	**0.73**
	Std	-0.24/+0.27	-0.14/+0.13	-0.27/+0.26	-0.16/+0.15	-0.25/+0.26	-0.13/+0.13
	IQM	0.13	0.70	0.39	0.66	0.61	0.74
	BCI	[0.23, 0.25]	[0.68, 0.69]	[0.37, 0.40]	[0.64, 0.65]	[0.57, 0.59]	[0.72, 0.73]
	Normal	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$	$\times(p = 0.00)$

Table 19: Extended generalization results with confidence intervals and Shapiro-Wilk normality checks for Traffic without the best selection per environment.