Preference-Based Gradient Estimation for ML-Guided Approximate Combinatorial Optimization

Arman Mielke[1,2] **Uwe Bauknecht**[1] **Thilo Strauss**[3] **Mathias Niepert**[2]
[1]ETAS Research, Germany [2]Computer Science Department, University of Stuttgart, Germany
[3]School of AI and Advanced Computing, Xi'an Jiaotong-Liverpool University, China
Corresponding author: Arman Mielke <arman.mielke@etas.com>

Abstract

Combinatorial optimization (CO) problems arise across a broad spectrum of domains. While exact solutions are often computationally infeasible, many practical applications require high-quality solutions within a given time budget. To address this, we propose a learning-based approach that enhances existing non-learned heuristics for CO. Specifically, we parameterize these heuristics and train graph neural networks (GNNs) to predict parameter values that yield near-optimal solutions. Our method is trained end-to-end in a self-supervised fashion, using a novel gradient estimation scheme that treats the heuristic as a black box. This approach combines the strengths of learning and traditional algorithms: the GNN learns from data to guide the algorithm toward better solutions, while the heuristic ensures feasibility. We validate our method on two well-known CO problems: the travelling salesman problem and the minimum k-cut problem.

1 Introduction

The design and analysis of heuristics for combinatorial optimization (CO) often focuses on improving worst-case performance. However, such worst-case scenarios may rarely occur in real-world settings. Learning-based approaches can adapt to the distribution of problem instances encountered in practice. Graph neural networks (GNNs) are often the method of choice, since most CO problems are either defined on graphs or admit graph-based formulations. Since neural networks cannot directly predict solutions guaranteed to be feasible, generic algorithms such as Monte Carlo tree search and beam search are often used to decode their outputs. However, these methods are impractical to use during training because of their prohibitively long runtime. Omitting them during training introduces a discrepancy between training and inference, which can impact optimality gap and generalization.

To overcome these limitations, we propose to use GNNs to augment highly efficient heuristics that are commonly available for many CO problems. The GNNs are trained to predict parameters for the heuristics that influence their behavior. Specifically, we predict edge weights for the input graphs and apply the heuristics to this modified graph. By choosing fast heuristics, we can incorporate them into the training loop and use the exact same pipeline during training and inference. Once trained, the GNN-augmented heuristics serve as drop-in replacements, preserving efficiency and usability while improving solution quality. Indeed, we provide a theoretical guarantee that, for a broad class of heuristics, a suitably trained GNN can learn parameters that render the heuristic optimal.

Since the heuristics return discrete solutions, the gradients of their outputs with respect to their inputs are zero almost everywhere. We therefore need to apply gradient estimation techniques such as the score function and straight-through estimators (Williams, 1992; Bengio et al., 2013) to backpropagate through the heuristics. Since existing gradient estimators have high variance or cannot be used in a self-supervised setting, we propose preference-based gradient estimation (PBGE), which estimates gradients through a comparison of solutions sampled from the given heuristic. This approach enables

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

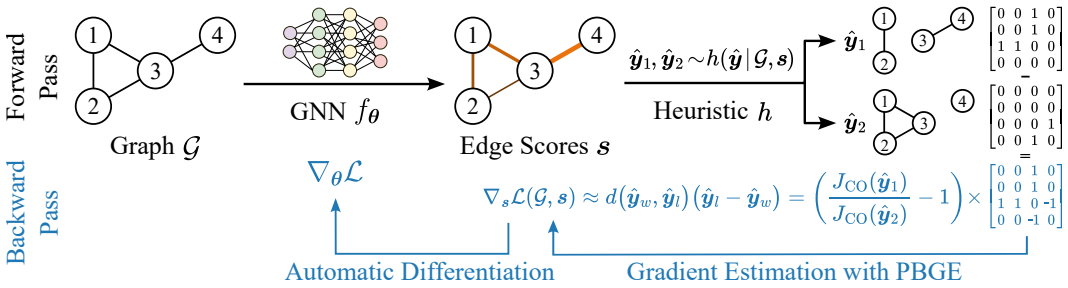


Figure 1: Overview of the proposed framework. In the forward pass, the GNN predicts edge scores that are used as input parameters for a CO heuristic. The heuristic operates on these by using them to scale the graph's edge weights and running an off-the-shelf heuristic on the modified graph to obtain a solution $\hat{\boldsymbol{y}}$. As the heuristic is not differentiable, a gradient estimator is used in the backward pass.

fully self-supervised training, eliminating the dependence on ground-truth labels, an often infeasible requirement for many CO problems. Figure 1 illustrates the proposed PBGE framework. We evaluate our approach on two well-known CO problems, the travelling salesman problem (TSP) and the minimum k-cut problem. Our approach improves the performance of a commonly used heuristic by an order of magnitude in the case of minimum k-cut while only minimally increasing the runtime.

In summary, our contributions include (1) a method for augmenting existing heuristics for CO using GNNs, (2) a self-supervised method for training a GNN for CO problems, (3) a novel gradient estimator, PBGE, for backpropagating through CO heuristics, and (4) an extensive experimental evaluation of (1-3) on common CO problems.

2 Related work

A simple way of training a model for CO is supervised learning: Nowak et al. (2017); Joshi et al. (2019) use a GNN to predict an approximate solution as a heatmap, which is decoded into a feasible solution using beam search. Sun & Yang (2023); Yu et al. (2024) employ graph-based denoising diffusion. However, supervised approaches are not applicable if exact solutions for the training problems are not available. Several approaches have used reinforcement learning (RL) to resolve this (Kwon et al., 2021; Qiu et al., 2022; Kim et al., 2024). A common approach is to formulate the CO problem as a Markov decision process (Bello et al., 2017; Deudon et al., 2018; Xu et al., 2020; Wu et al., 2022). Khalil et al. (2017); Kool et al. (2019) use the GNN autoregressively to predict which node is added to the solution set next and repeat that process until a valid solution is reached. However, these methods suffer from the large variance of the REINFORCE estimator. Methods that don't rely on RL can circumvent this problem, e.g. Sanokowski et al. (2024); Tönshoff et al. (2021); Zhang et al. (2023). A common approach is to formulate a surrogate loss consisting of a term that encourages high quality solutions and one that softly enforces constraints (Karalias & Loukas, 2020; Sun et al., 2022; Min et al., 2022, 2023; Wenkel et al., 2024). Concurrently to this work, Pan et al. (2025); Liao et al. (2025) also used preference learning for CO. Unlike these papers, our focus lies in using the GNN to augment existing CO heuristics by predicting parameters for the heuristic, allowing us to use the heuristic as decoder. We provide an extended discussion of related work in Appendix A.

3 Background and problem statement

Combinatorial optimization (CO) problems. A CO problem asks us, given a discrete set M and an objective function $J_{\text{CO}} : M \to \mathbb{R}$, to find $\min_{x \in M} J_{\text{CO}}(x)$. Since finding the exact optimum is often not required in practice, this paper focuses on efficiently finding approximate solutions.

The minimum k-cut problem asks, given a weighted, undirected graph and an integer $k \geq 2$, to find a set of edges with minimum total weight whose removal leaves the graph with exactly k connected components. In the travelling salesman problem (TSP), we are given a weighted, undirected graph, and are asked to find a minimum-weight Hamiltonian cycle. The variant of the TSP most commonly experimented on in related literature (Kool et al., 2019; Joshi et al., 2019, 2021) is the Euclidean TSP, where the graph is fully connected, the nodes represent points in the unit square and the edge weights are the distances between these points. Well-known probabilistic heuristics for these problems are the Karger–Stein algorithm (Karger & Stein, 1993) and the random insertion algorithm (Karg & Thompson, 1964), respectively. We give definitions for these problems and heuristics in Appendix B.

Problem statement. We consider CO problems on graphs with a linear objective function J_{CO} and a probabilistic heuristic. The heuristic takes as input a graph $\mathcal{G} = (V, E, w)$ with nodes V, edges E and edge weights $w : E \to \mathbb{R}_{>0}$, and returns (samples) a potentially suboptimal solution $\hat{\boldsymbol{y}}$. The heuristic, therefore, defines a probability distribution $h(\hat{\boldsymbol{y}} \mid \mathcal{G})$ over the solutions it outputs for a given graph $\mathcal{G}$. We now want to use a GNN $f_{\boldsymbol{\theta}}$ parameterized by $\boldsymbol{\theta}$ applied to the input graphs $\mathcal{G}$ to compute an updated graph $\mathcal{G}' = f_{\boldsymbol{\theta}}(\mathcal{G})$ such that the probabilistic heuristic when applied to this new graph is improved in expectation. Hence, we want to solve the following optimization problem:

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\hat{\boldsymbol{y}} \sim h(\hat{\boldsymbol{y}} | f_{\boldsymbol{\theta}}(\mathcal{G}))} \left[J_{\text{CO}}(\hat{\boldsymbol{y}}) \right].$$

For each input graph $\mathcal{G}$, $h(\hat{\boldsymbol{y}} | f_{\boldsymbol{\theta}}(\mathcal{G}))$ is a discrete probability distribution parameterized by $\boldsymbol{\theta}$. The main challenge in optimizing this is that discrete heuristics are typically not differentiable functions and that optimal solutions are prohibitively expensive to obtain as training data, making supervised training infeasible. Moreover, we assume that the heuristic is a black box—while we can sample from the probability distribution defined by it, we cannot compute a probability mass for a given sample.

4 Method

We introduce a novel approach to CO by deriving a new gradient estimator that enables backpropagation through probabilistic heuristics. This allows us to directly leverage solution quality rankings, generated via objective function-based comparisons, to learn to improve CO heuristics. We use graph neural networks (GNNs) to guide existing probabilistic heuristics for a given CO problem. The GNN receives the problem graph as input and produces a prior score for each edge. These scores are used as additional input alongside the graph for a parameterized version of an off-the-shelf CO heuristic, which then produces a solution to the CO problem. Since the heuristic is not differentiable in general, we use gradient estimation to obtain the gradients with respect to the GNN's output. Existing gradient estimators, such as REINFORCE, Gumbel softmax, or the implicit maximum likelihood estimator (I-MLE), either exhibit high bias or variance or require a differentiable loss function. We propose a gradient estimation scheme based on preference-based optimization, which we term preference-based gradient estimation (PBGE). Figure 1 shows an overview of our approach.

4.1 Parameterizing heuristics

A heuristic that takes a problem graph as input can be indirectly parameterized by modifying the graph before executing the algorithm. We modify an input graph $\mathcal{G}$ by using the GNN's output to change its edge weights. Assume there is an arbitrary but fixed ordering of edges. The model outputs for each edge a prior score $\boldsymbol{s} = f_{\boldsymbol{\theta}}(\mathcal{G}) \in \mathbb{R}^{|E|}$. A high score for a given edge is interpreted to mean that the respective edge should belong to the solution set with a higher probability mass. The heuristics we use prefer including edges of low weight in the solution set, so we scale down the weights of edges that received high scores. Specifically, the edge weights are multiplied with $1 - \sigma(\boldsymbol{s})$, where σ is the sigmoid function. By running the CO heuristic on this modified graph $\mathcal{G}'$, we parameterize the heuristic using the GNN's output scores $\boldsymbol{s}$. We use $h(\hat{\boldsymbol{y}} \mid \mathcal{G}, \boldsymbol{s}) = h(\hat{\boldsymbol{y}} \mid \mathcal{G}')$ to denote the probability distribution defined by a probabilistic CO heuristic parameterized in this way. It samples and outputs a vector $\hat{\boldsymbol{y}} \in \{0, 1\}^{|E|}$ that represents a solution to the CO problem, such as a TSP tour or k-cut. A value of 1 in $\hat{\boldsymbol{y}}$ means that the corresponding edge is in the solution set.

4.2 Preference-based gradient estimation (PBGE)

In preference learning, a training instance consists of an input and a pair of possible outputs. The supervision signal is an annotation indicating that one of the outputs $\boldsymbol{y}_w$ is of higher quality than the other output $\boldsymbol{y}_l$. We can construct a similar setup for CO by leveraging a pre-existing probabilistic CO heuristic h. Sampling from the heuristic multiple times likely yields two solutions $\hat{\boldsymbol{y}}_1, \hat{\boldsymbol{y}}_2 \sim h(\hat{\boldsymbol{y}} \mid \mathcal{G}, \boldsymbol{s})$ of different quality for a given problem instance $\mathcal{G} \sim \mathcal{D}$ from dataset $\mathcal{D}$. These solutions can easily be ranked by applying the CO problem's objective function J_{CO}. This means assigning $\hat{\boldsymbol{y}}_w$ and $\hat{\boldsymbol{y}}_l$ such that $J_{\text{CO}}(\hat{\boldsymbol{y}}_w) \leq J_{\text{CO}}(\hat{\boldsymbol{y}}_l)$. We propose the following preference-based loss function:

$$\mathcal{L}(\mathcal{D}, \boldsymbol{s}) = \mathbb{E}_{\hat{\boldsymbol{y}}_w, \hat{\boldsymbol{y}}_l \sim h(\hat{\boldsymbol{y}} | \mathcal{G}, \boldsymbol{s}),\, \mathcal{G} \sim \mathcal{D}} \left[d(\hat{\boldsymbol{y}}_w, \hat{\boldsymbol{y}}_l) \log \left(\frac{h(\hat{\boldsymbol{y}}_l \mid \mathcal{G}, \boldsymbol{s})}{h(\hat{\boldsymbol{y}}_w \mid \mathcal{G}, \boldsymbol{s})} \right) \right]. \tag{1}$$

$d(\hat{\boldsymbol{y}}_w, \hat{\boldsymbol{y}}_l)$ is a scaling factor (d also depends on $\mathcal{G}$, but we omit this for clarity). As we will see later, its purpose is to scale the gradients based on the distance between the objective values of $\hat{\boldsymbol{y}}_w$ and $\hat{\boldsymbol{y}}_l$. Since we treat the CO heuristic as a black box, we cannot calculate the probabilities $h(\hat{\boldsymbol{y}}_w \mid \mathcal{G}, \boldsymbol{s})$ and $h(\hat{\boldsymbol{y}}_l \mid \mathcal{G}, \boldsymbol{s})$ directly. We therefore introduce a proxy distribution $\pi(\hat{\boldsymbol{y}} \mid \mathcal{G}, \boldsymbol{s}) \approx h(\hat{\boldsymbol{y}} \mid \mathcal{G}, \boldsymbol{s})$ for which we can obtain probabilities directly. For all heuristics used in this paper, a high prior score in $\boldsymbol{s}$ for a certain edge increases the probability of this edge being included in the output $\hat{\boldsymbol{y}}$. This motivates the use of an exponential family distribution to model the proxy distribution π for h:

$$\pi(\hat{\boldsymbol{y}} \mid \mathcal{G}, \boldsymbol{s}) = \frac{\exp(\langle \hat{\boldsymbol{y}}, \boldsymbol{s} \rangle)}{\sum_{\boldsymbol{y}' \in \mathcal{C}} \exp(\langle \boldsymbol{y}', \boldsymbol{s} \rangle)}, \tag{2}$$

where $\langle \cdot, \cdot \rangle$ is the inner product and $\mathcal{C}$ is the set of all solutions to the CO problem. Replacing h with π in Eq. 1 and inserting Eq. 2 simplifies the loss function to

$$\mathcal{L}(\mathcal{D}, \boldsymbol{s}) = \mathbb{E}_{\hat{\boldsymbol{y}}_w, \hat{\boldsymbol{y}}_l \sim h(\hat{\boldsymbol{y}} \mid \mathcal{G}, \boldsymbol{s}),\, \mathcal{G} \sim D} \Big[d(\hat{\boldsymbol{y}}_w, \hat{\boldsymbol{y}}_l) \big(\langle \hat{\boldsymbol{y}}_l, \boldsymbol{s} \rangle - \langle \hat{\boldsymbol{y}}_w, \boldsymbol{s} \rangle \big) \Big]. \tag{3}$$

Now, the gradient of this expectation with respect to $\boldsymbol{s}$ and its single-sample Monte Carlo estimate are

$$\nabla_{\boldsymbol{s}} \mathcal{L}(\mathcal{D}, \boldsymbol{s}) = \mathbb{E}_{\hat{\boldsymbol{y}}_w, \hat{\boldsymbol{y}}_l \sim h(\hat{\boldsymbol{y}} \mid \mathcal{G}, \boldsymbol{s}),\, \mathcal{G} \sim D} \big[d(\hat{\boldsymbol{y}}_w, \hat{\boldsymbol{y}}_l)(\hat{\boldsymbol{y}}_l - \hat{\boldsymbol{y}}_w) \big], \tag{4}$$

$$\approx d(\hat{\boldsymbol{y}}_w, \hat{\boldsymbol{y}}_l)(\hat{\boldsymbol{y}}_l - \hat{\boldsymbol{y}}_w), \quad \text{where } \hat{\boldsymbol{y}}_w, \hat{\boldsymbol{y}}_l \sim h(\hat{\boldsymbol{y}} \mid \mathcal{G}, \boldsymbol{s}). \tag{5}$$

Intuitively, the gradient is negative at a certain edge if that edge is in the better solution, but not in the worse solution. A negative gradient raises the GNN's output score, so the GNN will be nudged towards including this edge in its solution. Similarly, a positive gradient means that the corresponding edge was only in the worse solution, and it pushes the GNN towards not including this edge.

The form of the gradient is reminiscent of those used for preference learning with large language models (LLMs) (Rafailov et al., 2023; Meng et al., 2024). Unlike the preference learning setting used with LLMs, we not only know which solution in a pair is better, but we can measure the quality of each solution exactly using the objective function. This eliminates the need for human annotators to rank pairs of examples. Moreover, we can leverage the objective function to more easily compute a suitable scaling factor.

If the solutions $\hat{\boldsymbol{y}}_w$ and $\hat{\boldsymbol{y}}_l$ are of similar quality, we do not want to strongly move the GNN towards either solution. We therefore scale the gradient with the relative optimality gap between the two:

$$d(\hat{\boldsymbol{y}}_w, \hat{\boldsymbol{y}}_l) = \frac{J_{\text{CO}}(\hat{\boldsymbol{y}}_l)}{J_{\text{CO}}(\hat{\boldsymbol{y}}_w)} - 1, \qquad \nabla_{\boldsymbol{s}} \mathcal{L}(\mathcal{G}, \boldsymbol{s}) \approx \left(\frac{J_{\text{CO}}(\hat{\boldsymbol{y}}_l)}{J_{\text{CO}}(\hat{\boldsymbol{y}}_w)} - 1 \right) (\hat{\boldsymbol{y}}_l - \hat{\boldsymbol{y}}_w).$$

d is always non-negative, since, by definition, $J_{\text{CO}}(\hat{\boldsymbol{y}}_l) \geq J_{\text{CO}}(\hat{\boldsymbol{y}}_w)$. Using this scaling factor places more weight on pairs of solutions where the difference in quality is large. If the two solutions are of the same quality, the gradient is set to zero, so we do not move the GNN towards either solution.

The variance of the gradient can be reduced by estimating the expectation in Eq. 4 by creating a pool of solutions with the heuristic and constructing pairs from this pool. We form pairs by combining the best solution from the pool with each of the weaker solutions. In practice, the accuracy of the gradients depends heavily on the quality of the best found solution $\hat{\boldsymbol{y}}_w$. At the beginning of training, the GNN cannot yet output good enough scores to consistently find reasonable $\hat{\boldsymbol{y}}_w$. To remedy this, we also run the heuristic on the unmodified graph and add the resulting solutions to the pool from which the pairs are generated. The complete training procedure is described in Algorithm 1.

Algorithm 1 Training iteration with PBGE

$\boldsymbol{s} \leftarrow f_{\boldsymbol{\theta}}(\mathcal{G})$

▷ Sample n solutions from h guided by $\boldsymbol{s}$, and m unguided solutions

$\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_n \sim h(\hat{\boldsymbol{y}} \mid \mathcal{G}, \boldsymbol{s})$

$\hat{\boldsymbol{y}}_{n+1}, \ldots, \hat{\boldsymbol{y}}_{n+m} \sim h(\hat{\boldsymbol{y}} \mid \mathcal{G})$

$\hat{\mathcal{Y}} \leftarrow \{\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_{n+m}\}$

$\hat{\boldsymbol{y}}_w \leftarrow \arg\min_{\hat{\boldsymbol{y}} \in \hat{\mathcal{Y}}} J_{\text{CO}}(\hat{\boldsymbol{y}})$

▷ Estimate gradient using each pair involving $\hat{\boldsymbol{y}}_w$

$$\nabla_{\boldsymbol{s}} \mathcal{L}(\mathcal{G}, \boldsymbol{s}) \approx \sum_{\hat{\boldsymbol{y}}_l \in \hat{\mathcal{Y}}} \left(\frac{J_{\text{CO}}(\hat{\boldsymbol{y}}_l)}{J_{\text{CO}}(\hat{\boldsymbol{y}}_w)} - 1 \right) (\hat{\boldsymbol{y}}_l - \hat{\boldsymbol{y}}_w)$$

Backpropagate gradient $\nabla_{\boldsymbol{s}} \mathcal{L}(\mathcal{G}, \boldsymbol{s})$

At test time, the model's output needs to be converted (decoded) to a solution to the CO problem. This can simply be done using the heuristic, as described in Section 3. The solution can be improved by running the probabilistic heuristic repeatedly and using the best solution found as final output.

4.3 Theoretical analysis

We prove the following theoretical results, which characterize under what conditions we can turn a heuristic into an exact algorithm if we find an optimal modified input graph. Theorem 1 proves this

Table 1: Minimum k-cut optimality gaps on graphs with 100 nodes, using Karger–Stein as decoder. Mean $\pm$ standard deviation were calculated over ten evaluation runs on the same model parameters. In the supervised and self-supervised rows, a GNN trained with the indicated method is used to guide the Karger–Stein algorithm. In the columns labeled "Best out of 3 runs", the Karger–Stein algorithm is run three times on the same GNN outputs, and the best result is used.

Method	**Unweighted graphs**		**NOIgen+**	
	Single run	Best out of 3 runs	Single run	Best out of 3 runs
Non-learned				
Karger–Stein	3.61%±0.21 (330ms)	0.62%±0.07 (971ms)	11.29%±0.71 (352ms)	0.43%±0.11 (1.00s)
Supervised				
BCE loss	0.27%±0.06 (402ms)	0.03%±0.02 (1.05s)	0.41%±0.07 (415ms)	0.06%±0.04 (1.04s)
I-MLE	1.67%±0.10 (415ms)	0.14%±0.06 (1.03s)	2.53%±0.12 (435ms)	0.28%±0.06 (1.08s)
Self-supervised				
I-MLE	3.39%±0.16 (402ms)	0.52%±0.05 (1.01s)	7.63%±0.41 (444ms)	0.41%±0.07 (1.10s)
PBGE (**ours**)	0.38%±0.05 (398ms)	0.06%±0.05 (1.01s)	0.58%±0.06 (439ms)	0.09%±0.05 (1.08s)

for the Karger–Stein algorithm, and Theorem 2 considers a wide class of CO problems and heuristics. More rigorous formulations of these theorems and their proofs can be found in Appendix C.

Theorem 1. *Let $h(\hat{\boldsymbol{y}} \mid \mathcal{G}, \boldsymbol{s})$ be the probability distribution defined by the Karger–Stein algorithm guided by scores $\boldsymbol{s} \in \mathbb{R}^{|E|}$. Let $\boldsymbol{y} \in \{0,1\}^{|E|}$ be a minimum k-cut on graph $\mathcal{G} = (V, E)$. Then,*

$$\lim_{\sigma(\boldsymbol{s}) \to \boldsymbol{y}} h(\boldsymbol{y} \mid \mathcal{G}, \boldsymbol{s}) = 1.$$

Theorem 2. *Let J_{CO} be the objective of a minimization problem on graphs with $J_{CO}(\hat{\boldsymbol{y}}) = \sum_{e \in \hat{\boldsymbol{y}}} w(e)$, $w(e) > 0$. Let $\boldsymbol{y} \in \{0,1\}^{|E|}$ be an optimal solution on graph $\mathcal{G}$. Let $h(\hat{\boldsymbol{y}} \mid \mathcal{G})$ be the probability distribution defined by a probabilistic heuristic, where for any input graph, the heuristic's error is bounded by a function α such that for all $\hat{\boldsymbol{y}} \sim h(\hat{\boldsymbol{y}} \mid \mathcal{G})$, $J(\hat{\boldsymbol{y}}) \leq \alpha(\mathcal{G}) J(\boldsymbol{y})$. Let $h(\hat{\boldsymbol{y}} \mid \mathcal{G}, \boldsymbol{s})$ be the parameterized version of that heuristic, using parameters $\boldsymbol{s} \in \mathbb{R}^{|E|}$. Then,*

$$\lim_{\sigma(\boldsymbol{s}) \to \boldsymbol{y}} h(\boldsymbol{y} \mid \mathcal{G}, \boldsymbol{s}) = 1.$$

In other words, in the limit, the heuristic guided by $\boldsymbol{s}$ always finds optimal solution $\boldsymbol{y}$. From this, the following result regarding insertion algorithms on TSP immediately follows:

Corollary 3. *Let $\mathcal{G} = (V, E, w)$ be an undirected graph, and let $h(\hat{\boldsymbol{y}} \mid \mathcal{G})$ be a probabilistic insertion algorithm, such as random insertion. Let $\boldsymbol{y} \in \{0,1\}^{|E|}$ be a minimal length TSP tour on $\mathcal{G}$. Then,*

$$\lim_{\sigma(\boldsymbol{s}) \to \boldsymbol{y}} h(\boldsymbol{y} \mid \mathcal{G}, \boldsymbol{s}) = 1.$$

5 Experiments

We validate our approach on two well-known combinatorial optimization problems: the travelling salesman problem (TSP) and the minimum k-cut problem. For both problems, we synthetically generate problem instances and establish baselines as reference. We use residual gated graph convnets (Bresson & Laurent, 2017), but adapt them to include edge features $\boldsymbol{e}_{ij}^{l}$ and a dense attention map $\boldsymbol{\eta}_{ij}^{l}$ following Joshi et al. (2019). Please see Appendix D for details.

Graph generation and baselines. For minimum k-cut, we use a more challenging modification of the established graph generator *NOIgen* (Nagamochi et al., 1994), which we call *NOIgen+*. We also use unweighted graphs that only rely on graph structure to prevent trivial solutions. Graphs for the TSP are generated using the established method in Kool et al. (2019); Joshi et al. (2019, 2021). Please refer to Appendix F for details. As we assume no ground truth solutions, our primary baselines are gradient estimators for unsupervised training. We set the loss to $\mathcal{L}(\mathcal{D}, \boldsymbol{s}) = \mathbb{E}_{\hat{\boldsymbol{y}} \sim h(\hat{\boldsymbol{y}} \mid \mathcal{G}, \boldsymbol{s}),\, \mathcal{G} \sim D}\left[J_{\mathrm{CO}}(\hat{\boldsymbol{y}})\right]$ and estimate $\nabla_{\boldsymbol{s}} \mathcal{L}(\mathcal{D}, \boldsymbol{s})$ using I-MLE (Niepert et al., 2021), and, in the case of TSP, REINFORCE (Williams, 1992). For minimum k-cut, we also train supervised baselines using an edge-level binary cross entropy (BCE) loss. Additionally, we train models using I-MLE in a supervised fashion, comparing the heuristic's output $\hat{\boldsymbol{y}}$ with $\boldsymbol{y}$ using a Hamming loss. See Appendix E for details.

Table 2: TSP optimality gaps, with mean ± standard deviation, calculated over ten evaluation runs on the same model parameters. We indicate the decoder used in parentheses. Values with * are obtained from these papers and therefore do not include standard deviations. We provide an extended version of this table with more baselines in Appendix G.3.

Method (decoder in parentheses)	$n = 20$	$n = 50$	$n = 100$
Deudon et al. (greedy)	0.66%* (2m)	3.98%* (5m)	8.41%* (8m)
Deudon et al. (sampling 1280 times)	0.11%* (5m)	1.28%* (17m)	12.70%* (56m)
Deudon et al. (sampling 1280 times + 2OPT)	0.09%* (6m)	1.00%* (32m)	4.64%* (5h)
Kool et al. (greedy)	0.34%*	1.76%* (2s)	4.53%* (6s)
Kool et al. (sampling 1280 times)	0.08%* (5m)	0.52%* (24m)	2.26%* (1h)
REINFORCE (random ins., 20 runs)	7.98%±0.08 (769ms)	23.84%±0.10 (11.62s)	52.83%±0.47 (1.38m)
REINFORCE (random ins., 100 runs)	4.78%±0.03 (3.64s)	11.24%±0.08 (57.85s)	31.06%±0.25 (7.53m)
I-MLE (random ins., 20 runs)	10.54%±0.15 (761ms)	35.87%±0.34 (11.14s)	61.42%±0.23 (1.37m)
I-MLE (random ins., 100 runs)	6.55%±0.10 (3.68s)	19.40%±0.24 (56.63s)	44.42%±0.14 (7.51m)
PBGE **(ours)** (random ins., 20 runs)	0.18%±0.01 (763ms)	2.37%±0.02 (11.11s)	5.12%±0.06 (1.43m)
PBGE **(ours)** (random ins., 100 runs)	0.05%±0.01 (3.73s)	1.13%±0.03 (53.98s)	3.67%±0.05 (7.44m)
Random Insertion	4.46%±0.08 (41ms)	7.57%±0.08 (575ms)	9.63%±0.11 (4.34s)

5.1 Results

Minimum k-Cut. We evaluate our method on the minimum k-cut problem, using the Karger–Stein algorithm as a base. Table 1 shows optimality gaps of the unmodified Karger–Stein algorithm, as well as several versions of our method. Each version augments the Karger–Stein algorithm with a GNN, and they differ by how the GNN was trained. Note that when augmenting the Karger–Stein algorithm with a GNN trained with PBGE, the optimality gap improves by an order of magnitude. On top of this, even though it didn't use any ground truth solutions during training, the GNN trained with PBGE comes close to matching the GNN trained supervised with a BCE loss. The experiments presented here use $k = 2$, experiments for larger values of k can be found in Appendix G.1. Also, an analysis of how our approach reduces the number of Karger–Stein runs required to find the exact minimum k-cut is presented in Appendix G.2.

Travelling salesman problem (TSP). Table 2 shows the optimality gaps of our approach and its variants on TSP. All of our models were trained using random insertion as the CO heuristic. For PBGE, we sampled 10 solutions from $h(\hat{\boldsymbol{y}} \mid \mathcal{G}, \boldsymbol{s})$ and 10 solutions from $h(\hat{\boldsymbol{y}} \mid \mathcal{G})$. The decoder used at test time is listed after the name of the respective method in parentheses. Greedy search repeatedly follows the edge with the highest score, until each node has been visited once. Sampling simply refers to sampling multiple solutions and using the best one. "+ 2OPT" refers to improving the decoded solution using 2OPT local search (Croes, 1958). Figure 2 shows the Pareto frontier of self-supervised methods on TSP-100. PBGE achieves Pareto-optimality when decoding with 100 random insertion runs.

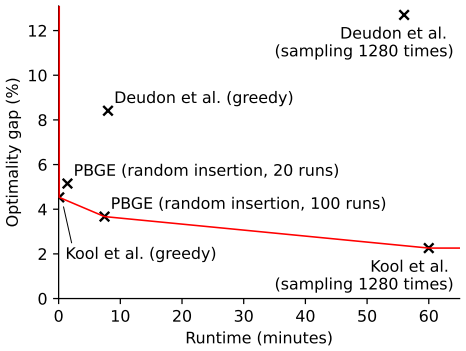


Figure 2: The Pareto frontier of self-supervised methods on TSP-100. Non-Pareto-optimal baselines with very high optimality gaps or long runtimes are omitted for clarity.

6 Conclusion

We introduced a method to improve existing heuristics for CO using GNNs. The GNN predicts parameters, which are used as input for the non-learned heuristic to produce a high-quality solution to the CO problem. The GNN is trained based on the CO problem's downstream objective, without the need for labelled data. To achieve this, we used gradient estimation to backpropagate through the heuristic. We proposed a novel gradient estimation scheme for this purpose, which we called preference-based gradient estimation (PBGE). Incorporating a CO heuristic during training incurs some limitations. Firstly, the training process is more computationally intensive compared to competing approaches. Secondly, an existing heuristic is required for our approach. Additionally, we only experimented on CO problems for which solutions can be represented in terms of the graph's edges. We leave extending our approach to other kinds of CO problems to future work.

Acknowledgments and Disclosure of Funding

The authors would like to thank Aneesh Barthakur, Matteo Palmas, Roman Freiberg and Jiaqi Wang for fruitful discussions. The authors thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Arman Mielke. The contributions of Thilo Strauss were carried out while employed with ETAS Research.

References

Antoniadis, A., Eliáš, M., Polak, A., and Venzin, M. Approximation algorithms for combinatorial optimization with predictions. In *International Conference on Learning Representations*, 2025.

Applegate, D. L., Bixby, R. E., Chvatál, V., and Cook, W. J. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.

Badrinath, A., Agarwal, P., and Xu, J. Hybrid preference optimization: Augmenting direct preference optimization with auxiliary objectives. *arXiv e-prints*, pp. arXiv–2405, 2024.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. Neural combinatorial optimization with reinforcement learning. In *International Conference on Learning Representations*, 2017.

Bengio, E., Jain, M., Korablyov, M., Precup, D., and Bengio, Y. Flow network based generative models for non-iterative diverse candidate generation. In *Advances in Neural Information Processing Systems*, volume 34, pp. 27381–27394. Curran Associates, Inc., 2021.

Bengio, Y., Léonard, N., and Courville, A. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013.

Bresson, X. and Laurent, T. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553v2*, 2017.

Bu, F., Jo, H., Lee, S. Y., Ahn, S., and Shin, K. Tackling prevalent conditions in unsupervised combinatorial optimization: Cardinality, minimum, covering, and more. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 4696–4729. PMLR, Jul 2024.

Chekuri, C. S., Goldberg, A. V., Karger, D. R., Levine, M. S., and Stein, C. Experimental study of minimum cut algorithms. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '97, pp. 324–333. Society for Industrial and Applied Mathematics, 1997.

Cheng, H., Zheng, H., Cong, Y., Jiang, W., and Pu, S. Select and optimize: Learning to solve large-scale TSP instances. In *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, volume 206 of *Proceedings of Machine Learning Research*, pp. 1219–1231. PMLR, Apr 2023.

Corsini, A., Porrello, A., Calderara, S., and Dell' Amico, M. Self-labeling the job shop scheduling problem. In *Advances in Neural Information Processing Systems*, volume 37, pp. 105528–105551. Curran Associates, Inc., 2024.

Croes, G. A. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, Dec 1958.

Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., and Rousseau, L.-M. Learning heuristics for the TSP by policy gradient. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 15th International Conference, CPAIOR*, pp. 170–181. Springer-Verlag, Jun 2018.

Duan, H., Vaezipoor, P., Paulus, M. B., Ruan, Y., and Maddison, C. Augment with care: Contrastive learning for combinatorial problems. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 5627–5642. PMLR, Jul 2022.

Elmachtoub, A. N. and Grigas, P. Smart "predict, then optimize". *Management Science*, 68(1):9–26, Jan 2022.

Gasse, M., Chetelat, D., Ferroni, N., Charlin, L., and Lodi, A. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Georgiev, D. G., Numeroso, D., Bacciu, D., and Lio, P. Neural algorithmic reasoning for combinatorial optimisation. In *Proceedings of the Second Learning on Graphs Conference*, volume 231 of *Proceedings of Machine Learning Research*, pp. 28:1–28:15. PMLR, Nov 2023.

Joshi, C. K., Laurent, T., and Bresson, X. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

Joshi, C. K., Cappart, Q., Rousseau, L.-M., and Laurent, T. Learning TSP requires rethinking generalization. In *International Conference on Principles and Practice of Constraint Programming*, 2021.

Kahng, A., Nerem, R., Wang, Y., and Yang, C.-Y. NN-Steiner: A mixed neural-algorithmic approach for the rectilinear Steiner minimum tree problem. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38:13022–13030, Mar 2024.

Karalias, N. and Loukas, A. Erdős goes neural: An unsupervised learning framework for combinatorial optimization on graphs. In *Advances in Neural Information Processing Systems*, volume 33, pp. 6659–6672. Curran Associates, Inc., 2020.

Karg, R. L. and Thompson, G. L. A heuristic approach to solving travelling salesman problems. *Management Science*, 10(2):225–248, 1964. doi: 10.1287/mnsc.10.2.225.

Karger, D. R. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, pp. 21–30. Society for Industrial and Applied Mathematics, 1993.

Karger, D. R. and Stein, C. An $\tilde{\mathrm{O}}(n^2)$ algorithm for minimum cuts. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pp. 757–765. Association for Computing Machinery, 1993. doi: 10.1145/167088.167281.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., and Song, L. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Kim, H., Kim, M., Ahn, S., and Park, J. Symmetric replay training: Enhancing sample efficiency in deep reinforcement learning for combinatorial optimization. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 24110–24136. PMLR, Jul 2024.

Kool, W., van Hoof, H., and Welling, M. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

Kwon, Y.-D., Choo, J., Yoon, I., Park, M., Park, D., and Gwon, Y. Matrix encoding networks for neural combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 34, pp. 5138–5149. Curran Associates, Inc., 2021.

Liao, Z., Chen, J., Wang, D., Zhang, Z., and Wang, J. BOPO: Neural combinatorial optimization via best-anchored and objective-guided preference optimization. *arXiv preprint arXiv:2503.07580*, Mar 2025.

Mandi, J., Kotary, J., Berden, S., Mulamba, M., Bucarey, V., Guns, T., and Fioretto, F. Decision-focused learning: Foundations, state of the art, benchmark and future opportunities. *J. Artif. Int. Res.*, 80:1623–1701, Sep 2024.

McCarty, E., Zhao, Q., Sidiropoulos, A., and Wang, Y. NN-Baker: A neural-network infused algorithmic framework for optimization problems on geometric intersection graphs. In *Advances in Neural Information Processing Systems*, volume 34, pp. 23023–23035. Curran Associates, Inc., 2021.

Meng, Y., Xia, M., and Chen, D. SimPO: Simple preference optimization with a reference-free reward. In *Advances in Neural Information Processing Systems*, volume 37, pp. 124198–124235. Curran Associates, Inc., 2024.

Min, Y., Wenkel, F., Perlmutter, M., and Wolf, G. Can hybrid geometric scattering networks help solve the maximum clique problem? In *Advances in Neural Information Processing Systems*, volume 35, pp. 22713–22724. Curran Associates, Inc., 2022.

Min, Y., Bai, Y., and Gomes, C. P. Unsupervised learning for solving the travelling salesman problem. In *Advances in Neural Information Processing Systems*, volume 36, pp. 47264–47278. Curran Associates, Inc., 2023.

Minervini, P., Franceschi, L., and Niepert, M. Adaptive perturbation-based gradient estimation for discrete latent variable models. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence*. AAAI Press, 2023. doi: 10.1609/aaai.v37i8.26103.

Nagamochi, H., Ono, T., and Ibaraki, T. Implementing an efficient minimum capacity cut algorithm. *Mathematical Programming*, 67:325–341, 1994.

Niepert, M., Minervini, P., and Franceschi, L. Implicit MLE: Backpropagating through discrete exponential family distributions. In *Advances in Neural Information Processing Systems*, volume 34, pp. 14567–14579. Curran Associates, Inc., 2021.

Nowak, A., Villar, S., Bandeira, A., and Bruna, J. A note on learning algorithms for quadratic assignment with graph neural networks. *arXiv preprint arXiv:1706.07450*, Jun 2017.

Pan, M., Lin, G., Luo, Y.-W., Zhu, B., Dai, Z., Sun, L., and Yuan, C. Preference optimization for combinatorial optimization problems. *arXiv preprint arXiv:2505.08735*, May 2025.

Pan, X., Jin, Y., Ding, Y., Feng, M., Zhao, L., Song, L., and Bian, J. H-TSP: Hierarchically solving the large-scale traveling salesman problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 9345–9353, Jun 2023.

Qiu, R., Sun, Z., and Yang, Y. Dimes: A differentiable meta solver for combinatorial optimization problems. In *Advances in Neural Information Processing Systems*, volume 35, pp. 25531–25546. Curran Associates, Inc., 2022.

Rafailov, R., Sharma, A., Mitchell, E., Manning, C. D., Ermon, S., and Finn, C. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pp. 53728–53741. Curran Associates, Inc., 2023.

Rosenkrantz, D. J., Stearns, R. E., and Lewis II, P. M. An analysis of several heuristics for the traveling salesman problem. *SIAM journal on computing*, 6(3):563–581, Sep 1977.

Sanokowski, S., Hochreiter, S., and Lehner, S. A diffusion model framework for unsupervised neural combinatorial optimization. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 43346–43367. PMLR, Jul 2024.

Schuetz, M. J. A., Brubaker, J. K., and Katzgraber, H. G. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4(4):367–377, Apr 2022.

Sun, H., Guha, E. K., and Dai, H. Annealed training for combinatorial optimization on graphs. *arXiv preprint arXiv:2207.11542*, 2022.

Sun, Z. and Yang, Y. DIFUSCO: Graph-based diffusion solvers for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 36, pp. 3706–3731. Curran Associates, Inc., 2023.

Tönshoff, J., Ritzert, M., Wolf, H., and Grohe, M. Graph neural networks for maximum constraint satisfaction. *Frontiers in artificial intelligence*, 3:580607, 2021.

Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

Vlastelica, M., Paulus, A., Musil, V., Martius, G., and Rolínek, M. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, May 2020.

Wenkel, F., Cantürk, S., Horoi, S., Perlmutter, M., and Wolf, G. Towards a general recipe for combinatorial optimization with multi-filter GNNs. In *The Third Learning on Graphs Conference*, 2024.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, May 1992. doi: 10.1007/BF00992696.

Wu, Y., Song, W., Cao, Z., Zhang, J., and Lim, A. Learning improvement heuristics for solving routing problems. *IEEE Transactions on Neural Networks and Learning Systems*, 33(9):5057–5069, 2022.

Xia, Y., Yang, X., Liu, Z., Liu, Z., Song, L., and Bian, J. Position: Rethinking post-hoc search-based neural approaches for solving large-scale traveling salesman problems. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 54178–54190. PMLR, Jul 2024.

Xu, Z., Abe, K., Sato, I., and Sugiyama, M. Solving NP-hard problems on graphs with extended AlphaGo Zero. *arXiv preprint arXiv:1905.11623*, 2020.

Yau, M., Karalias, N., Lu, E., Xu, J., and Jegelka, S. Are graph neural networks optimal approximation algorithms? In *Advances in Neural Information Processing Systems*, volume 37, pp. 73124–73181. Curran Associates, Inc., 2024.

Yu, K., Zhao, H., Huang, Y., Yi, R., Xu, K., and Zhu, C. DISCO: Efficient diffusion solver for large-scale combinatorial optimization problems. *arXiv preprint arXiv:2406.19705*, 2024.

Zhang, D., Dai, H., Malkin, N., Courville, A. C., Bengio, Y., and Pan, L. Let the flows tell: Solving graph combinatorial problems with GFlowNets. In *Advances in Neural Information Processing Systems*, volume 36, pp. 11952–11969. Curran Associates, Inc., 2023.

A Extended related work

The easiest way of training a model for CO is to assume the existence of ground truth solutions to the CO problems and train in a supervised fashion. Nowak et al. (2017); Joshi et al. (2019) use a GNN to predict an approximate solution as a heatmap, which is then decoded into a feasible solution to the CO problem using beam search. Vinyals et al. (2015) introduce pointer networks, which leverage the fact that many CO problems ask to identify a subset or a permutation of the input. Georgiev et al. (2023) follow a neural algorithmic reasoning approach to learn to imitate CO solvers. Gasse et al. (2019); Kahng et al. (2024) replace components of an existing algorithm that would otherwise be expensive to compute with neural networks. Antoniadis et al. (2025) augment existing algorithms using discrete predictions. Finally, Sun & Yang (2023); Yu et al. (2024) use graph-based denoising diffusion to generate high-quality solutions. However, these supervised approaches aren't applicable to such cases where calculating exact solutions for the training problems is not feasible.

Several approaches have used reinforcement learning (RL) to remove this dependence on a labeled dataset (Kwon et al., 2021; Qiu et al., 2022; Kim et al., 2024). A common approach is to formulate the CO problem as a Markov decision process (Bello et al., 2017; Deudon et al., 2018; Xu et al., 2020; Wu et al., 2022). Khalil et al. (2017); Kool et al. (2019) use the GNN autoregressively to predict which node should be added next to the solution set and repeat that process until a valid solution is reached. Cheng et al. (2023); Pan et al. (2023); McCarty et al. (2021) tackle large-scale instances by locally optimizing sub-parts of the instance individually. However, RL is often sample inefficient and difficult to train due to the high variance of the gradient estimations.

A common self-supervised approach that does not rely on RL is to formulate a surrogate loss consisting of a term that encourages high quality solutions and one that softly enforces the constraints (Karalias & Loukas, 2020; Sun et al., 2022; Min et al., 2022, 2023; Bu et al., 2024; Wenkel et al., 2024). Duan et al. (2022) train self-supervised using a contrastive loss instead. Schuetz et al. (2022); Sanokowski et al. (2024) focus on quadratic and polynomial unconstrained binary optimization, allowing them to formulate self-supervised loss functions for these problem families. Tönshoff et al. (2021) use an LSTM-based architecture to solve binary maximum constraint satisfaction problems, which many CO problems can be formulated as. Corsini et al. (2024) train a pointer network-based model in a self-supervised fashion. Zhang et al. (2023) tackle CO using GFlowNets (Bengio et al., 2021). Yau et al. (2024) propose a GNN architecture that can capture message passing algorithms with provable approximation guarantees for a large class of CO problems.

Joshi et al. (2021) compare some of the paradigms introduced in other papers in structured experiments. Xia et al. (2024) argue that the common pattern of training the GNN using a surrogate loss, but testing it using a decoder, means that the GNN is applied inconsistently, leading to uncertain performance during testing. Our approach addresses this problem by incorporating the decoder during training. There have been two lines of work on backpropagating through CO problems. Firstly, if we have a set of optimal solutions given as training data, we can use supervised learning to train the GNN to output adjacency matrices as close as possible to the optimal solutions (Elmachtoub & Grigas, 2022). This is often called "predict, then optimize". Secondly, there are several methods to backpropagate through a non-differentiable CO algorithm, such as Niepert et al. (2021); Minervini et al. (2023); Vlastelica et al. (2020). Related to our work is decision-focused learning, which has developed several methods to backpropagate through CO solvers (Mandi et al., 2024). Our approach follows this paradigm. The I-MLE (Niepert et al., 2021) is another such method; we compare against it in our experiments and show that our approach improves over I-MLE. Preference learning has previously been used outside CO for fine-tuning LLMs (Badrinath et al., 2024; Rafailov et al., 2023; Meng et al., 2024). For instance, Badrinath et al. (2024) propose a hybrid approach between preference learning and reinforcement learning from human feedback.

Concurrently to this work, Pan et al. (2025); Liao et al. (2025) also applied preference learning to CO. Similarly to our approach, these methods use a decoder during training to collect a pool of CO solutions for each training instance, then build pairs of solutions from this pool. These pairs are each ranked by preference and used to construct a gradient. Two major differences to our work lie in the decoder and the construction of the solution pool. Pan et al. (2025) decode solutions by modelling the CO problem as a Markov decision process and constructing the solution autoregressively, an approach commonly used in RL-based methods. Solution pairs are simply constructed from all combinations of solutions in the pool, including many low-quality pairs. In contrast, our approach of using only pairs containing the best found solution emphasizes learning from high-quality solutions,

while disregarding less useful pairs where both solutions are low-quality. Liao et al. (2025) use a sampling decoder, and additionally add one solution from a greedy decoder to the pool. The solution pool is pre-filtered by sorting the solutions by quality and only retaining every n-th solution, then the pairs are constructed from the remaining solutions. Importantly, unlike Pan et al. (2025); Liao et al. (2025), our focus lies in using the GNN to augment existing CO heuristics by predicting parameters for the heuristics. This also allows us to use these heuristics as decoder.

B Combinatorial optimization problems and heuristics

B.1 Minimum k-cut problem

B.1.1 Problem Definition

We are given a connected, undirected graph $\mathcal{G} = (V, E, w)$ with edge weights $w : E \to \mathbb{R}_{>0}$, as well as a desired number of connected components $k \in \mathbb{N}, 2 \leq k \leq |V|$. The goal is to find a set of edges $C \subseteq E$ with minimal total weight whose removal leaves k connected components. This set is called a *minimum k-cut*. Formally, we are optimizing

$$\min_{C \subseteq E} \sum_{e \in C} w(e) \quad \text{such that graph } (V, E \backslash C) \text{ has } k \text{ connected components.}$$

B.1.2 Karger's algorithm

Karger's algorithm (Karger, 1993) is a Monte Carlo algorithm for the minimum k-cut problem.

The algorithm is based on the *contraction operation*: An edge $e = \{x, y\}$ is contracted by merging its nodes x and y into a new node xy. For clarity, we will call the node that results from this merger a meta-node. Every edge that was incident to exactly one of the two merged nodes is now altered to instead be incident to the meta-node xy: An edge $\{x, z\}$ or $\{y, z\}$ becomes $\{xy, z\}$. This may result in parallel edges, meaning that the resulting graph is a multigraph. All edges $\{x, y\}$ are removed, so that the resulting multigraph contains no self-loops.

Karger's algorithm works by repeatedly sampling an edge, where the probability of each edge is proportional to its weight, then contracting that edge. This is repeated until there are only k nodes left. Each of these remaining k meta-nodes represents a connected component in the original graph, with each node of the original graph that was subsequently been merged into that meta-node belonging to this connected component. Any edge in the original graph that spans between two connected components is cut.

Since this algorithm is not guaranteed to find the minimum k-cut, a common strategy is to run the algorithm repeatedly and use the smallest found cut as the final result.

B.1.3 Karger–Stein algorithm

The Karger–Stein algorithm (Karger & Stein, 1993) is a recursive version of Karger's algorithm, shown in Algorithm 2.

B.2 Travelling Salesman Problem (TSP)

B.2.1 Problem Definition

We are given a complete directed or undirected graph $\mathcal{G} = (V, E, w)$ with edge weights $w : E \longrightarrow \mathbb{R}$. Our goal is to find a minimum weight Hamiltonian cycle in $\mathcal{G}$. A Hamiltonian cycle, also called a tour, is a sequence of nodes where each node in the graph appears exactly once, each node in the sequence is adjacent to the previous node, and the first node is adjacent to the last node. The weight of the cycle is the sum of all edge weights of edges between nodes that appear next to each other in the cycle, including the edge from the last to the first node. Instead of representing a tour as a sequence of nodes, we will represent it as the set of edges between nodes that appear next to each other in the sequence. Formally, we are optimizing

$$\min_{T \subseteq E} \sum_{e \in T} w(e) \quad \text{such that } T \text{ forms a Hamiltonian cycle.}$$

Algorithm 2 Karger–Stein algorithm

KARGER–STEIN
Input: connected, undirected graph $\mathcal{G} = (V, E, w)$
if $|V| \leq 6$ **then**
  return CONTRACT$(\mathcal{G}, 2)$
else
  target $t \leftarrow \left\lceil \frac{|V|}{\sqrt{2}} + 1 \right\rceil$
  $\mathcal{G}_1 \leftarrow$ CONTRACT$(\mathcal{G}, t)$
  $\mathcal{G}_2 \leftarrow$ CONTRACT$(\mathcal{G}, t)$
  return $\min\{$KARGER–STEIN$(\mathcal{G}_1),$ KARGER–STEIN$(\mathcal{G}_2)\}$ ▷ Return the lower-weight cut
end if

CONTRACT
Input: connected, undirected graph $\mathcal{G} = (V, E, w)$, target number of nodes t
while $|V| > t$ **do**
  $\mathcal{G} \leftarrow$ sample edge in $\mathcal{G}$ and contract it
end while
return $\mathcal{G}$

The nodes are commonly called cities, and the weight of an edge is commonly called the distance between the two cities. The TSP is NP complete.

Metric TSP. The metric version of TSP additionally assumes that the distances between the cities form a metric. This means that

- the graph is undirected (or $w(x, y) = w(y, x)$ for all $x, y \in V$),
- distances between cities at different locations are positive, and
- the edge weights satisfy the triangle inequality, i.e. $w(x, y) + w(y, z) \geq w(x, z)$ for all $x, y, z \in V$.

Euclidean TSP. Euclidean TSP is a special case of metric TSP in which the cities are located at points in the unit square and the distances between the cities are the Euclidean distances between the respective points. Research on solving the TSP using neural networks often focuses on this version of the problem (Kool et al., 2019; Joshi et al., 2019, 2021).

B.2.2 Random insertion algorithm

The random insertion algorithm (Karg & Thompson, 1964) is a Monte Carlo algorithm for the TSP.

Since a Hamiltonian cycles to a given graph $\mathcal{G} = (V, E)$ is required to contain every $v \in V$ exactly once, it is a straightforward approach to iteratively sample and remove nodes from V until it is empty. The random insertion algorithm, as suggested by Karg and Thompson, begins by selecting two nodes $s, t \in V$ at random and adds the edges (s, t) and (t, s) to an initial cycle. In order to extend the cycle to include all nodes, the algorithm now samples a node $v \in V \setminus \{s, t\}$ and selects the edges (x, v) and (v, y) such that x and y are already part of the partial cycle with $x \neq y$ and such that the sum of the metric distances of (x, v) and (v, y) is minimal.

The cycles obtained in this way are at most $(\lceil \log_2 |V| \rceil + 1)$ times longer than the optimal cycle (Rosenkrantz et al., 1977).

The algorithm is summarized in Algorithm 3.

C Proofs

We assume that the edges E of a graph $\mathcal{G} = (V, E, w)$ are in an arbitrary but fixed order. This means that the scores assigned to the edges by a GNN can be represented as a vector $\boldsymbol{s} \in \mathbb{R}^{|E|}$. We use $\boldsymbol{s}[e]$

Algorithm 3 Random insertion

Input: connected, undirected graph $\mathcal{G} = (V, E, w)$
$\mathcal{T} \leftarrow$ TSP tour consisting of one random node
$v \leftarrow$ sample node in V that is not part of $\mathcal{T}$ yet
$\mathcal{T} \leftarrow$ insert v into $\mathcal{T}$ to form a loop of two nodes
for $i \in \{1, \dots, |V|\}$ **do**
  $v \leftarrow$ sample node in V that is not part of $\mathcal{T}$ yet
  $\mathcal{T} \leftarrow$ insert v into $\mathcal{T}$ at the point in the tour $\mathcal{T}$ where it increases the tour's length by the least amount
end for
return $\mathcal{G}$

to denote the score assigned to a specific edge $e \in E$. A subset of edges $\hat{\boldsymbol{y}}$ can be represented as $\hat{\boldsymbol{y}} \in \{0, 1\}^{|E|}$, where a 1 indicates that the respective edge is in the set. For notational simplicity, we will still write $e \in \hat{\boldsymbol{y}}$ for edges that are in this subset.

A probabilistic CO heuristic for an edge subset problem defines a probability distribution over the subsets of edges. We use $h(\hat{\boldsymbol{y}} \mid \mathcal{G}, \boldsymbol{s})$ to denote the probability that the output is edge subset $\hat{\boldsymbol{y}}$ given input graph $\mathcal{G}$ and edge scores $\boldsymbol{s}$, as described in Section 4.

We use σ to denote the element-wise sigmoid function $\sigma(x) = \frac{e^x}{1+e^x}$.

C.1 Probability of finding the optimal minimum k-cut

Let $\mathcal{G} = (V, E)$ be an undirected graph. During the first $i - 1$ iterations of the modified Karger's algorithm, some edges have been merged away. Let E_i be the set of edges that are left at iteration i (this means $E_1 = E$). The probability that a specific edge e is selected for contraction at iteration i is

$$p_i(e) = \frac{1 - \sigma(\boldsymbol{s}[e])}{\sum_{e' \in E_i} \big(1 - \sigma(\boldsymbol{s}[e'])\big)}.$$

Theorem 1. *Let $\mathcal{G} = (V, E)$ be an undirected graph, and let $\boldsymbol{y} \in \{0, 1\}^{|E|}$ be a minimum k-cut on $\mathcal{G}$. Moreover, let $h(\hat{\boldsymbol{y}} \mid \mathcal{G}, \boldsymbol{s})$ be the probability distribution defined by Karger's algorithm guided by parameters $\boldsymbol{s} \in \mathbb{R}^{|E|}$, and let σ be the element-wise sigmoid function. Then,*

$$\lim_{\sigma(\boldsymbol{s}) \to \boldsymbol{y}} h(\boldsymbol{y} \mid \mathcal{G}, \boldsymbol{s}) = 1.$$

In other words, in the limit, Karger's algorithm guided by $\boldsymbol{s}$ always finds optimal solution $\boldsymbol{y}$.

Proof. Assuming that no edge in $\boldsymbol{y}$ has been contracted yet, the probability that the edge that is selected for contraction in iteration i is in $\boldsymbol{y}$ is

$$p_i(e \in \boldsymbol{y}) = \sum_{e \in \boldsymbol{y}} p_i(e) = \frac{\sum_{e \in \boldsymbol{y}} \big(1 - \sigma(\boldsymbol{s}[e])\big)}{\sum_{e' \in E_i} \big(1 - \sigma(\boldsymbol{s}[e'])\big)}$$

Karger's algorithm outputs a given cut $\boldsymbol{y}$ if and only if no edge in $\boldsymbol{y}$ is contracted by the algorithm (see Karger & Stein (1993), lemma 2.1). Let k be the parameter for minimum k-cut, i.e. the number of desired connected components. Karger's algorithm will always terminate after $|V| - k$ contraction steps. The probability that no edge in $\boldsymbol{y}$ is contracted during the $|V| - k$ contraction steps is

$$h(\boldsymbol{y} \mid \mathcal{G}, \boldsymbol{s}) = \prod_{i=1}^{|V|-k} \big(1 - p_i(e \in \boldsymbol{y})\big) = \prod_{i=1}^{|V|-k} \left(1 - \frac{\sum_{e \in \boldsymbol{y}} \big(1 - \sigma(\boldsymbol{s}[e])\big)}{\sum_{e' \in E_i} \big(1 - \sigma(\boldsymbol{s}[e'])\big)}\right)$$

In the numerator, $\lim_{\sigma(s) \to \boldsymbol{y}} \sum_{e \in \boldsymbol{y}} \big(1 - \sigma(\boldsymbol{s}[e])\big) = 0$, since all $\sigma(\boldsymbol{s}[e])$ go to 1. The denominator is greater than zero, because there is at least one edge left that is not in $\boldsymbol{y}$, otherwise the algorithm

would be finished. Since the fraction goes towards zero, all terms of the product go towards 1 and $\lim_{\sigma(\boldsymbol{s})\to\boldsymbol{y}} h(\boldsymbol{y} \mid \mathcal{G}, \boldsymbol{s}) = 1$. □

The result can trivially be extended to the Karger–Stein algorithm, a variant of Karger's algorithm, and to weighted graphs $\mathcal{G} = (V, E, w)$.

C.2 Probability of finding optimal solutions using algorithms with relative error bounds

Let $\mathcal{G} = (V, E, w)$ be a weighted graph with edge weights $w : E \to \mathbb{R}_{>0}$. Consider any CO problem that asks to find a set of edges of minimum total weight that fits the constraints (e.g. minimum k-cut or TSP). This means that the CO problem's objective function takes the form $J(\hat{\boldsymbol{y}}) = \sum_{e\in\hat{\boldsymbol{y}}} w(e)$. Remember that we allow ourselves the notational convenience of treating $\hat{\boldsymbol{y}}$ as both a set of edges and a vector in $\{0, 1\}^{|E|}$.

Theorem 2. *Let $\mathcal{G} = (V, E, w)$ be an undirected, weighted graph with $w(e) > 0$. Let J be the objective of a minimization problem on graphs with $J(\hat{\boldsymbol{y}}) = \sum_{e\in\hat{\boldsymbol{y}}} w(e)$, and let $\boldsymbol{y} \in \{0, 1\}^{|E|}$ be an optimal solution on $\mathcal{G}$. Moreover, let $h(\hat{\boldsymbol{y}} \mid \mathcal{G})$ be the probability distribution defined by a probabilistic heuristic. For every input graph with arbitrary edge weights, the algorithm returns an approximate solution whose error is bounded by a function α such that for all $\hat{\boldsymbol{y}} \sim h(\hat{\boldsymbol{y}} \mid \mathcal{G})$, $J(\hat{\boldsymbol{y}}) \leq \alpha(\mathcal{G})J(\boldsymbol{y})$. Finally, let $h(\hat{\boldsymbol{y}} \mid \mathcal{G}, \boldsymbol{s})$ be the parameterized version of that heuristic, using parameters $\boldsymbol{s} \in \mathbb{R}^{|E|}$, and let σ be the element-wise sigmoid function. Then,*

$$\lim_{\sigma(\boldsymbol{s})\to\boldsymbol{y}} h(\boldsymbol{y} \mid \mathcal{G}, \boldsymbol{s}) = 1.$$

Proof. Let $\mathcal{G}'$ be the graph that the heuristic receives as input after modification through $\boldsymbol{s}$, i.e. $h(\hat{\boldsymbol{y}} \mid \mathcal{G}, \boldsymbol{s}) = h(\hat{\boldsymbol{y}} \mid \mathcal{G}')$. Let $J_{\mathcal{G}'}(\boldsymbol{y})$ denote the total weight of a solution in the modified graph $\mathcal{G}'$. As $\sigma(\boldsymbol{s})$ approaches $\boldsymbol{y}$, the weights in $\mathcal{G}'$ of edges in $\boldsymbol{y}$ approach 0. This means that the total weight of the optimal solution $\boldsymbol{y}$ in the modified graph $\mathcal{G}'$ approaches 0:

$$\lim_{\sigma(\boldsymbol{s})\to\boldsymbol{y}} J_{\mathcal{G}'}(\boldsymbol{y}) = 0.$$

Any other solution $\boldsymbol{y}'$ cannot be a strict subset of the edges included in $\boldsymbol{y}$, since that would mean that it has a lower total weight than $\boldsymbol{y}$. $\boldsymbol{y}'$ therefore includes at least one edge that is not included in $\boldsymbol{y}$. Since this edge's original edge weight is greater than 0, its scaled weight remains greater than 0 as $\sigma(\boldsymbol{s})$ approaches $\boldsymbol{y}$. The total weight in $\mathcal{G}'$ of any solution other than $\boldsymbol{y}$ therefore approaches a value greater than 0:

$$\lim_{\sigma(\boldsymbol{s})\to\boldsymbol{y}} J_{\mathcal{G}'}(\boldsymbol{y}') > 0 \quad \forall \boldsymbol{y}' \neq \boldsymbol{y}.$$

In the limit, the right hand side of the relative error bound $J_{\mathcal{G}'}(\hat{\boldsymbol{y}}) \leq \alpha(G)J_{\mathcal{G}'}(\boldsymbol{y})$ approaches 0. This means that the maximum total weight of solutions that the heuristic can find approaches 0, and, as soon as the lengths of the suboptimal solutions in $\mathcal{G}'$ are far enough away from 0, $\boldsymbol{y}$ is the only solution left that the heuristic can find. □

Note that this proof can trivially be adapted to CO problems that asks to find a set of *nodes* of minimum total weight that fits the constraints.

We know that the TSP tour produced by any insertion algorithm $h(\hat{\boldsymbol{y}} \mid \mathcal{G})$, such as random insertion (Karg & Thompson, 1964), is within $\lceil \log(|V|) \rceil + 1$ of the optimal tour (Rosenkrantz et al., 1977). Formally, this means that $J(\hat{\boldsymbol{y}}) \leq \left(\lceil \log(|V|) \rceil + 1\right) J(\boldsymbol{y})$ for $\hat{\boldsymbol{y}} \sim h(\hat{\boldsymbol{y}} \mid \mathcal{G})$. From that, the following result immediately follows:

Corollary 3. *Let $\mathcal{G} = (V, E, w)$ be an undirected, weighted graph, and let $\boldsymbol{y} \in \{0, 1\}^{|E|}$ be a minimal length TSP tour on $\mathcal{G}$. Let $h(\hat{\boldsymbol{y}} \mid \mathcal{G}, \boldsymbol{s})$ be a probabilistic insertion algorithm, guided by parameters $\boldsymbol{s} \in \mathbb{R}^{|E|}$. Let σ be the element-wise sigmoid function. Then,*

$$\lim_{\sigma(\boldsymbol{s})\to\boldsymbol{y}} h(\boldsymbol{y} \mid \mathcal{G}, \boldsymbol{s}) = 1.$$

D Implementation details

D.1 Residual gated graph convnets

We use residual gated graph convnets (Bresson & Laurent, 2017), but adapt them to include edge features $\boldsymbol{e}_{ij}^l$ and a dense attention map $\boldsymbol{\eta}_{ij}^l$ following Joshi et al. (2019). The input node features $\boldsymbol{x}_i^0$ and edge features $\boldsymbol{e}_{ij}^0$ are first pre-processed using a single-layer MLP (multi-layer perceptron) for each of the two. Each further layer is computed as follows:

$$\boldsymbol{x}_i^{l+1} = \boldsymbol{x}_i^l + \mathrm{ReLU}\left(\mathrm{BN}\left(W_1^l \boldsymbol{x}_i^l + \sum_{j \in \mathcal{N}_i} \boldsymbol{\eta}_{ij}^l \odot W_2^l \boldsymbol{x}_j^l\right)\right) \quad \text{with } \boldsymbol{\eta}_{ij}^l = \frac{\sigma\big(\boldsymbol{e}_{ij}^l\big)}{\sum_{j' \in \mathcal{N}_i} \sigma\big(\boldsymbol{e}_{ij'}^l + \varepsilon\big)} \in \mathbb{R}^d,$$

$$\boldsymbol{e}_{ij}^{l+1} = \boldsymbol{e}_{ij}^l + \mathrm{ReLU}\left(\mathrm{BN}\left(W_3^l \boldsymbol{e}_{ij}^l + W_4^l \boldsymbol{x}_i^l + W_5^l \boldsymbol{x}_j^l\right)\right),$$

where $W_1^l, \ldots, W_5^l \in \mathbb{R}^{d \times d}$ are learnable weights, d is the hidden dimension, ReLU is the rectified linear unit, BN is batch normalization, $\sigma = \frac{e^x}{1+e^x}$ is the element-wise sigmoid function, and ε is an arbitrary small value. $\odot$ denotes the Hadamard product, and $\mathcal{N}_i$ denotes the set of nodes that are adjacent to i.

The final edge-level output is calculated from the last layer's edge features $\boldsymbol{e}_{ij}^l$ using another MLP. $f(\mathcal{G}) \in \mathbb{R}^{|E|}$ refers to applying this GNN on a graph $\mathcal{G}$.

D.2 Decoding using the Karger–Stein algorithm

In the case of the Karger–Stein algorithm, we noticed empirically that simply modifying the input graph can lead to degenerate behavior during testing. The Karger–Stein algorithm uses the graph's edge weights in two places: (1) when sampling an edge for contraction and (2) when comparing the cuts that resulted from different recursion arms. We noticed that the performance of our overall method can be improved when using a model trained with the setting described in Section 4.2 by using the modified edge weights for the first case and the original edge weights for the second case. Intuitively, if the GNN makes a mistake when scaling the edge weights, using the original edge weights for comparing cuts can allow the Karger–Stein algorithm to find the optimal cut regardless.

E Details regarding baselines

E.1 Supervised training with binary cross entropy loss

The task is treated as an edge-level binary classification task. The network is trained using a BCE loss:

$$\hat{y} = \sigma\big(f_\theta(\mathcal{G})\big)$$

$$\mathcal{L}_{\text{supervised BCE}}(\mathcal{G}, y) = \mathrm{BCE}(\hat{y}, y) = \sum_{i=1}^{|E|} y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)$$

where f_θ is a GNN, $\mathcal{G}$ is the input graph, $y \in \{0, 1\}^{|E|}$ is the ground truth solution and $\hat{y} \in (0, 1)^{|E|}$ is the predicted solution.

A ground truth label of 1 represents that an edge belongs to a minimum k-cut or a TSP tour.

E.2 REINFORCE

Recall that the REINFORCE algorithm (Williams, 1992), also known as the score function estimator, calculates

$$\nabla_\theta \mathbb{E}_{y \sim p_\theta(x)}\big[J(y)\big] = \mathbb{E}_{y \sim p_\theta(x)}\big[J(y) \nabla_\theta \log p_\theta(x)\big]$$

where J is an objective function, and $p_\theta(x)$ is a probability distribution parameterized by θ.

We assume that p_θ is a discrete constrained exponential family distribution, i.e.

$$p_\theta(x) = \begin{cases} \frac{\exp\big(\langle x, \theta\rangle\big)}{\sum_{x'} \exp\big(\langle x', \theta\rangle\big)} & \text{if } x \text{ satisfies the constraints} \\ 0 & \text{otherwise} \end{cases}$$

For valid x,

$$\log p_\theta(x) = \langle x, \theta\rangle - A(\theta),$$

where $A(\theta)$ is the log-partition function

$$A(\theta) = \log\left(\sum_{x' \in \mathcal{C}} \exp\big(\langle x', \theta\rangle\big)\right).$$

Since $\nabla_\theta A(\theta) = \mathbb{E}_{y \sim p_\theta(x)}[y]$, we get

$$\nabla_\theta \log p_\theta(x) = x - \mathbb{E}_{y \sim p_\theta(x)}[y].$$

Inserting this into the REINFORCE formula gives us

$$\nabla_\theta \mathbb{E}_{y \sim p_\theta(x)}\big[J(y)\big] = \mathbb{E}_{y \sim p_\theta(x)}\left[J(y)\Big(y - \mathbb{E}_{y' \sim p_\theta(x)}[y']\Big)\right].$$

Using the Gumbel-max trick, we sample from p_θ by sampling $\varepsilon \sim \mathrm{Gumbel}(0, 1)$, then calculating $y := h(\theta + \varepsilon)$.

Estimating the outer expectation by sampling once and the inner expectation by sampling N times, we arrive at Algorithm 4.

Algorithm 4 REINFORCE

Input: distribution parameter θ
$\varepsilon \sim \mathrm{Gumbel}(0, 1)$
$y \leftarrow h(\theta + \varepsilon)$
$\nabla_\theta J(y) \leftarrow J(y)\left(y - \frac{1}{N}\sum_{i=1}^{N} y_i\right)$
where $\varepsilon_i \sim \mathrm{Gumbel}(0, 1)$
$y_i \leftarrow h(\theta + \varepsilon_i)$
return $\nabla_\theta J(y)$

E.3 Implicit maximum likelihood estimator (I-MLE)

I-MLE (Niepert et al., 2021) allows estimating gradients with respect to the parameters of discrete exponential family distributions. This can be used to backpropagate through CO solvers as follows. In the forward pass, perturb the input $\boldsymbol{\theta} \in \mathbb{R}^n$ to the CO solver using noise $\boldsymbol{\epsilon} \sim \rho(\boldsymbol{\epsilon})$ sampled from a suitable noise distribution. Then run the CO solver on the perturbed input $\boldsymbol{\theta} + \boldsymbol{\epsilon}$, obtaining output $\boldsymbol{z}$.

In the backward pass, assume we know the gradient of the loss w.r.t. to $\boldsymbol{z}$, $\nabla_{\boldsymbol{z}}\mathcal{L}$. First, obtain a modified input $\boldsymbol{\theta}'$ for which we can expect better outputs compared to $\boldsymbol{\theta}$. One generally applicable option suggested by Niepert et al. (2021) is $\boldsymbol{\theta}' = \boldsymbol{\theta} - \lambda\nabla_{\boldsymbol{z}}\mathcal{L}$, where λ is a hyperparameter. Using the same noise $\boldsymbol{\epsilon}$ as in the forward pass, perturb $\boldsymbol{\theta}'$ and run the CO solver on $\boldsymbol{\theta}' + \boldsymbol{\epsilon}$, obtaining $\boldsymbol{z}'$. Finally, return the estimated gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L} \approx \boldsymbol{z} - \boldsymbol{z}'$.

This produces biased gradient estimates, but with much smaller variance than REINFORCE. To further reduce variance, this procedure can be repeated S times, sampling new noise $\boldsymbol{\epsilon}_i \sim \rho(\boldsymbol{\epsilon}_i)$ each time and averaging the results.

We used three noise samples from the Sum-of-Gamma noise distribution suggested by Niepert et al. (2021), using $\kappa = 5$ and 100 iterations.

E.3.1 Supervised training with I-MLE

The outputs of the GNN are used to guide a CO heuristic. This heuristic outputs a solution to the CO problem, which can be compared to the ground truth solution using a Hamming loss. During backpropagation, I-MLE (Niepert et al., 2021) is used to estimate the gradient of the loss with respect to the GNN's output. This setting has no practical benefit over the simple supervised training using a BCE loss, but it serves to measure the effectiveness of I-MLE.

The training procedure works as follows.

$$
\begin{aligned}
s &= \sigma\big(f_\theta(\mathcal{G})\big) \\
\hat{y} &= h(\mathcal{G}, 1 - s) \\
\mathcal{L}_{\text{supervised I-MLE}}(\mathcal{G}, y) &= \frac{1}{|E|} \sum_{i=1}^{|E|} \hat{y}_i(1 - y_i) + (1 - \hat{y}_i) y_i
\end{aligned}
$$

Note that here, $\hat{y} \in \{0, 1\}^{|E|}$ is guaranteed to be a valid solution to the given CO problem. I-MLE is used to estimate $\frac{\mathrm{d}\hat{\mathcal{L}}}{\mathrm{d}(1-s)}$.

E.3.2 I-MLE target distribution

We're using a custom target distribution for I-MLE in the supervised setting. This target distribution is similar to the target distribution for CO problems presented in (Niepert et al., 2021). The idea behind it is to recover the ground truth label from the loss, which is possible when using the Hamming loss:

$$
\begin{aligned}
\ell(\hat{y}, y) &= \hat{y}(1 - y) + (1 - \hat{y}) y \\
\frac{\mathrm{d}}{\mathrm{d}\hat{y}} \ell(\hat{y}, y) &= 1 - 2y \\
y &= \frac{1 - \frac{\mathrm{d}}{\mathrm{d}\hat{y}} \ell(\hat{y}, y)}{2}
\end{aligned}
$$

The best value for θ is $1 - y$ (we have to invert it because the input to the heuristic is inverted).[1] With this we arrive at the following target distribution:

$$
\theta' = 1 - y = \frac{1 + \frac{\mathrm{d}}{\mathrm{d}\hat{y}} \mathcal{L}}{2}
$$

E.3.3 Self-supervised training with I-MLE

Since we're using a CO heuristic that guarantees that its outputs are valid solutions to the CO problem, we can use the CO problem's objective function as a loss directly instead of the supervised Hamming loss. The ground truth labels are therefore no longer required.

In the case of minimum k-cut, the size of the cut is used as loss function. For TSP, the length of the tour is used.

$$
\begin{aligned}
s &= \sigma\big(f_\theta(\mathcal{G})\big) \\
\hat{y} &= h(\mathcal{G}, 1 - s) \\
\mathcal{L}_{\text{self-supervised I-MLE}}(\mathcal{G}) &= J_{\text{CO}}(\hat{y})
\end{aligned}
$$

where J_{CO} is the objective function of the CO problem. Note that the CO problem's constraints do not explicitly appear here, because the CO heuristic already guarantees that the constraints are met.

As before, I-MLE is used to estimate $\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}(s-1)}$. In this setting, the general-purpose target distribution for I-MLE presented in (Niepert et al., 2021) is used, setting $\lambda = 20$ as suggested by that paper.

[1] Using θ in the same sense as Niepert et al. (2021). In our case, $\theta = 1 - s$

F Graph generation

F.1 Minimum k-cut

Many commonly used graph generators create graphs with low-degree nodes. These graphs contain trivial solutions to the minimum k-cut problem in which $k-1$ connected components only contain one node, and one connected component contains all of the remaining nodes. When creating a dataset for minimum k-cut, care must therefore be taken to avoid graphs with low-degree nodes.

Graphs without edge weights. A simple method to generate graphs with meaningful solutions to the minimum k-cut problem is as follows. Create k fully connected subgraphs of random sizes within a given range. Then, add a random number of edges between random nodes of different subgraphs while ensuring that the resulting graph is connected. An additional benefit of this method is that, if the number of edges added between subgraphs is smaller than the number of nodes in the smallest subgraph by at least two, then the minimum k-cut is known from the construction: the cut consists of exactly the edges that were added between subgraphs. However, since all graphs generated this way consist of fully connected subgraphs, these problem instances are limited in diversity.

The range of possible problem instances can be improved by generating graphs of varying density. Start by assigning nodes to k subgraphs of random sizes within a given range. Then, add a random number of edges that connect nodes of different subgraphs. For each subgraph, add edges between random nodes within the same subgraph until all nodes have a higher degree than the number of edges between subgraphs. As long as there are enough edges between subgraphs, the minimum k-cut very likely consists of the edges between subgraphs. The minimum node degree and hence the density of the graph depends on the number of edges between subgraphs and therefore on the size of the minimum k-cut.

Graphs with edge weights. For minimum k-cut graphs with edge weights, a graph generator commonly called NOIgen (Nagamochi et al., 1994) (named after the initials of the authors) is often used. NOIgen works by first creating a specified number of nodes and adding edges between random nodes until a specified density is reached (sometimes, a Hamilton path is created first to ensure that the graph is connected). The weights of the edges are chosen uniformly at random. Finally, the nodes are randomly divided into k subgraphs. The weights of edges that connect nodes of different subgraphs are scaled down by a fixed factor.

When testing traditional, non-learned algorithms, the scaling factor is sometimes chosen to be very small, such that the minimum k-cut is very likely to consist of the edges between subgraphs (Chekuri et al., 1997). However, this makes the problem trivially easy for GNNs, which can learn that a very low edge weight corresponds to an edge belonging to the minimum k-cut. This allows the GNN to disregard the graph structure and therefore circumvent the challenging part of the problem. On the other hand, if the weights of edges between subgraphs are not scaled down enough, the generated graph might have a trivial solution that simply cuts out $k-1$ nodes.

To combat this problem, we modify NOIgen by controlling not just the weights of edges between subgraphs, but also the number of edges between subgraphs. We add a parameter that specifies which fraction of edges is generated between subgraphs (as opposed to within the same subgraph). Ensuring that there are few enough edges between subgraphs allows for a milder downscaling of their edge weights without introducing a trivial solution. This in turn prevents the GNN from inferring whether an edge belongs to the minimum k-cut simply from its weight.

The minimum k-cut in these graphs usually consists of the edges between subgraphs, but this is not guaranteed. The ground truth solution is therefore calculated separately to make sure that it reflects the optimal cut, as described in Section F.3. Another benefit of calculating them separately is that we can set the number of subgraphs to a different number than k, generating more interesting graphs.

F.2 Travelling salesman problem

Instances of Euclidean TSP are commonly generated with this simple algorithm:

1. Create a fully connected graph with n nodes

Table 3: Minimum k-cut optimality gaps on unweighted graphs with 100 nodes, for $k = 3$ and $k = 4$. The row labelled "PBGE" refers to a GNN trained self-supervised with PBGE, using the Karger–Stein algorithm as decoder. In the columns labeled "Best out of 3 runs", the Karger–Stein algorithm is run three times on the same GNN outputs, and the best result is used.

Method	$k = 3$		$k = 4$	
	Single run	Best out of 3 runs	Single run	Best out of 3 runs
Karger–Stein	32.95%	4.93%	45.92%	9.16%
PBGE (**ours**)	3.71%	0.04%	9.72%	0.81%

Table 4: Number of Karger–Stein runs required to find the minimum k-cut with probability 0.99 on unweighted graphs with 100 nodes. The row labelled "PBGE" refers to a GNN trained self-supervised with PBGE, using the Karger–Stein algorithm as decoder. The reported times refer to the total execution time of the indicated number of Karger–Stein runs, including the time required to run the GNN once (if present).

Method	$k = 3$	$k = 4$
Karger–Stein	7 runs (1.56s)	7 runs (1.42s)
PBGE (**ours**)	2 runs (552ms)	3 runs (671ms)

2. For each node, draw a position in the unit square uniformly at random and assign it as node features
3. Calculate the distances between the nodes and assign them as edge features

F.3 Calculating ground truth labels for supervised training

In general, ground truth labels are generated using a traditional (i.e. non-learned) algorithm. In some settings, the graph can be constructed such that the ground truth solution can be obtained simultaneously from the same construction process, in which case running the traditional algorithm is not necessary.

- Minimum k-cut: For graphs without edge weights, the graphs can be constructed with known ground truth solutions. See Section F.1 for details. For graphs with edge weights, the Karger–Stein algorithm (Karger & Stein, 1993) is run 100 times, and the smallest cut found is treated as the ground truth minimum cut.
- TSP: The well-established Concorde solver (Applegate et al., 2006), which guarantees optimal solutions, is used to generate ground truth labels.

G Extended experiments

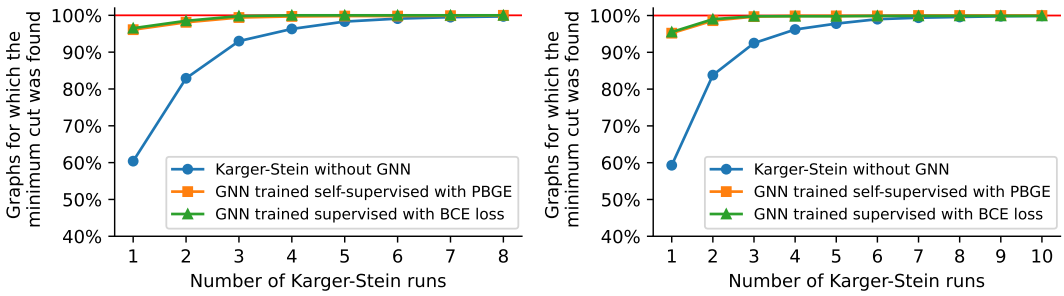
G.1 Minimum k-cut with $k > 2$

Table 3 shows optimality gaps of PBGE and Karger–Stein on minimum k-cut with $k = 3$ and $k = 4$. In both cases, augmenting the Karger–Stein algorithm using a GNN trained self-supervised with PBGE substantially improves the optimality gap.

It is common to run the Karger–Stein algorithm repeatedly until the minimum k-cut is found with sufficiently large probability. Table 4 shows the number of Karger-Stein runs required to find the optimal cut with probability 0.99. The reported number is the lowest number of Karger–Stein runs for which the minimum k-cut is found for 99% of the graphs in the validation set. For both $k = 3$ and $k = 4$, our approach reduces the number of required Karger–Stein runs by more than half, thus providing a considerable speed up.

G.2 Number of Karger–Stein runs to find the minimum k-cut

As stated before, it is common to run the Karger–Stein algorithm repeatedly until the minimum k-cut is found with sufficiently large probability. Therefore, the an important metric is the number of runs



(a) On graphs without edge weights and 100 nodes.

(b) On graphs without edge weights and 200 nodes.

Figure 3: The number of graphs for which the minimum k-cut was found after a given number of Karger–Stein runs, for $k = 2$. For example, for Karger–Stein without a GNN on graphs with 100 nodes (Figure 3a), two Karger–Stein runs suffice to find the minimum k-cut for 83% of graphs.

for Karger–Stein to find the optimal k-cut. If this number is low, we can run Karger–Stein a small number of times and be reasonably certain that the minimum k-cut was found. Figure 3 shows for how many graphs the minimum k-cut is found in a set number of runs, comparing the unmodified Karger–Stein algorithm with two versions that were augmented using a GNN. On both datasets, the augmented Karger–Stein algorithm needs much fewer runs to find the minimum k-cut, almost always finding it on the first attempt. Again, the GNN trained self-supervised with PBGE comes close to matching supervised performance.

G.3 Extended comparison for TSP

Table 5 compares PBGE with more baselines, including ones that make use of supervised learning.

We also compare against a simple self-supervised baseline that runs random insertion on the input graph 20 times and treats the best solution found as ground truth for a BCE loss. We call this baseline "Best-of-20".

The beam search decoder works similarly to the greedy decoder. It starts with an arbitrary node, then explores the b edges with the highest scores. This gives us b partial solutions. In each iteration, each partial solution is expanded at its last node, and out of the resulting paths, the b best partial solutions are kept. Edges that would lead to invalid tours are ignored. The parameter b is called the *beam width*, and beam search with $b = 1$ corresponds to greedy search.

H Hyperparameters and other details

Table 6 and Table 7 detail the hyperparameters used for experiments on minimum k-cut and TSP, respectively.

Our training sets for minimum k-cut contain 10,000 graphs, and the validation sets contain 1,000 graphs. For TSP, our training sets consist of 1,000,000 graphs, and our validation sets contain 1,000 graphs.

I Hardware resources used for experiments

We used an AMD EPYC 7313 16-Core processor and an NVIDIA H100 GPU in our experiments. Runtime measurements were taken on an 11th Gen Intel Core i7-11850H processor with a base speed of 2.50 GHz, using a single CPU core and no GPU, with a batch size of 1.

Table 5: TSP optimality gaps with mean ± standard deviation calculated over ten evaluation runs on the same model parameters. We group methods by their original paper and indicate the decoder used in parentheses. Results marked with * are values obtained from the indicated papers and therefore do not include standard deviations.

Method (decoder in parentheses)	$n = 20$	$n = 50$	$n = 100$
Self-supervised			
Bello et al. (2017)			
(greedy)	1.42%*	4.46%*	6.90%*
Deudon et al. (2018)			
(greedy)	0.66%* (2m)	3.98%* (5m)	8.41%* (8m)
(greedy + 2OPT)	0.42%* (4m)	2.77%* (26m)	5.21%* (3h)
(sampling 1280 times)	0.11%* (5m)	1.28%* (17m)	12.70%* (56m)
(sampling 1280 times + 2OPT)	0.09%* (6m)	1.00%* (32m)	4.64%* (5h)
Khalil et al. (2017)			
(greedy)	1.42%*	5.16%*	7.03%*
Kool et al. (2019)			
(greedy)	0.34%*	1.76%* (2s)	4.53%* (6s)
(sampling 1280 times)	0.08%* (5m)	0.52%* (24m)	2.26%* (1h)
REINFORCE (random ins., 20 runs)	7.98%±0.08 (769ms)	23.84%±0.10 (11.62s)	52.83%±0.47 (1.38m)
REINFORCE (random ins., 100 runs)	4.78%±0.03 (3.64s)	11.24%±0.08 (57.85s)	31.06%±0.25 (7.53m)
I-MLE (random ins., 20 runs)	10.54%±0.15 (761ms)	35.87%±0.34 (11.14s)	61.42%±0.23 (1.37m)
I-MLE (random ins., 100 runs)	6.55%±0.10 (3.68s)	19.40%±0.24 (56.63s)	44.42%±0.14 (7.51m)
Best-of-20 (random insertion, 20 runs)	0.40%±0.03 (787ms)	11.94%±0.11 (11.03s)	15.82%±0.09 (1.47m)
Best-of-20 (random insertion, 100 runs)	0.10%±0.01 (3.70s)	5.21%±0.15 (52.82s)	9.82%±0.09 (7.38m)
PBGE **(ours)** (random ins., 20 runs)	0.18%±0.01 (763ms)	2.37%±0.02 (11.11s)	5.12%±0.06 (1.43m)
PBGE **(ours)** (random ins., 100 runs)	0.05%±0.01 (3.73s)	1.13%±0.03 (53.98s)	3.67%±0.05 (7.44m)
Supervised, not directly comparable			
Joshi et al. (2019)			
(greedy)	0.60%* (6s)	3.10%* (55s)	8.38%* (6m)
(beam search, beam width 1280)	0.10%* (20s)	0.26%* (2m)	2.11%* (10m)
(beam search, width 1280 + heuristic)	0.01%* (12m)	0.01%* (18m)	1.39%* (40m)
Sun & Yang (2023)			
(greedy)		0.10%*	0.24%*
(sampling 16 times)		0.00%*	0.00%*
BCE loss (random insertion, 20 runs)	0.15%±0.01 (787ms)	0.95%±0.03 (10.92s)	2.86%±0.04 (1.47m)
BCE loss (random insertion, 100 runs)	0.04%±0.00 (3.75s)	0.59%±0.02 (54.79s)	1.75%±0.03 (7.35m)
Non-learned heuristics			
Christofides	8.72%±0.00 (45ms)	11.07%±0.00 (685ms)	11.86%±0.00 (4.45s)
Random Insertion	4.46%±0.08 (41ms)	7.57%±0.08 (575ms)	9.63%±0.11 (4.34s)
Farthest Insertion	2.38%±0.00 (57ms)	5.50%±0.00 (909ms)	7.58%±0.00 (5.68s)
LKH3	0.00%* (18s)	0.00%* (5m)	0.00%* (21m)
Exact solvers, not directly comparable			
Concorde (Applegate et al., 2006)	0.00%* (1m)	0.00%* (2m)	0.00%* (3m)
Gurobi	0.00%* (7s)	0.00%* (2m)	0.00%* (17m)

Table 6: Hyperparameters used for minimum k-cut.

Hyperparameter	Value
GNN layers	4
GNN hidden channels	32
MLP prediction head layers	2
Optimizer	AdamW
Weight Decay	0.01
Learning rate scheduler	ReduceLROnPlateau
Initial learning rate	0.001
Learning rate scheduler patience	4
Batch size	64 graphs

Table 7: Hyperparameters used for TSP.

Hyperparameter	Value
GNN layers	12
GNN hidden channels	100
MLP prediction head layers	3
Optimizer	AdamW
Weight Decay	0.01
Learning rate scheduler	ReduceLROnPlateau
Initial learning rate	0.0001
Learning rate scheduler patience	4
Batch size	64 graphs

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims made in the abstract and introduction are supported by the paper.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitations are discussed in Section 6.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.

- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Proofs are provided in Appendix C.

   Guidelines:

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The code as well as the generators for the datasets are provided.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code as well as the generators for the datasets are provided.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Implementation details are described in Appendix D. Hyperparameters and other details are listed in Appendix H.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: All optimality gaps from our experiments are reported as mean±standard deviation.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The compute resources used in the experiments are described in Appendix I.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the code of ethics and adhere to it.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [NA]

Justification: The goal of our work is to advance the fields of machine learning and combinatorial optimization. There are many potential societal consequences of any work in these fields, but it is impossible to foresee or mitigate particular impacts for general methods such as ours. There is no direct path to any negative applications.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: The method presented in this paper poses no immediate risk for misuse, as it tackles general combinatorial optimization problems.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The datasets used in this paper are synthetic; we generated all datasets ourselves. Where code from other projects is used, the original authors are credited and a copy of the original license is included.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Documentation comments are provided as part of the code.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: No crowdsourcing experiments and no research with human subjects were performed.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: No crowdsourcing experiments and no research with human subjects were performed.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: LLMs were not used in this work, apart from programming aids (which do not need to be documented according to the LLM policy).

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.