SELF-SUPERVISED COMBINATORIAL OPTIMIZATION WITH CONSTRAINTS VIA FRANK–WOLFE

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-supervised learning for combinatorial optimization has emerged as a promising paradigm for solving discrete optimization problems with neural networks, but a central challenge remains: handling hard combinatorial constraints within continuous, gradient-based training. Continuously extending combinatorial objectives to convex domains is a powerful technique, yet existing approaches often require projection steps that constrain neural network outputs to lie inside the feasible polytope and rely on ad-hoc and problem-specific constructions. We propose a general framework in which the neural network is allowed to predict arbitrary continuous vectors that could potentially lie outside of the feasible polytope. These predictions are then approximated by sparse convex combinations of feasible solutions using a geometric decomposition algorithm based on Frank–Wolfe methods and approximate Carathéodory results. This decomposition induces an a.e.-differentiable, self-supervised loss defined as the expected value of the discrete objective. The same procedure provides an automatic rounding guarantee at inference time. We demonstrate strong empirical performance across multiple combinatorial problems, including the Quadratic Assignment Problem, Maximum Coverage, and the Traveling Salesperson Problem.

1 INTRODUCTION

Combinatorial optimization (CO) forms a central pillar of optimization theory and practice, encompassing a broad class of problems in which one seeks to optimize an objective over a discrete, often exponentially large, feasible set. Such problems arise ubiquitously across science and engineering, from routing and resource allocation to learning and inference. Canonical examples include the Traveling Salesperson Problem (TSP), Maximum Coverage (MC), and Quadratic Assignment Problem (QAP), among many others. What unifies these problems is not only their computational hardness, but also the rich algorithmic structure induced by discrete constraints that describe large-scale discrete combinatorial spaces of configurations.

Despite this shared structure, successful approaches to CO have historically been case-specific, often reflecting deep problem-dependent insights; ”stroke-of-genius” heuristics. A common algorithmic template, especially in approximation algorithms, relaxes the discrete problem into a continuous or convex surrogate, solves for a fractional solution, and then applies a bespoke rounding or improvement procedure to recover feasibility. While this relax-and-round paradigm has led to remarkable practical results and theoretical guarantees, adapting this approach to gradient-based data-driven settings can be challenging. This limitation and challenge is especially pronounced when problem instances are drawn from structured distributions, increasingly common in real world applications. Self-supervised learning (SSL) approaches for combinatorial optimization offer an appealing alternative: unlike static approximation algorithms and heuristics, they can leverage latent patterns in data, train efficiently without requiring large amounts of labeled solutions, and incorporate algorithmic priors that facilitate learning and generalization.

In this work, we bridge the gap between discrete optimization and deep learning by building on work that used continuous extensions of discrete functions as losses for neural CO Karalias et al. (2022; 2025); Nerem et al. (2025). The central idea in this line of work is to use extensions to embed discrete constraints and objectives directly into a learning pipeline, allowing learning and optimization to proceed end-to-end. By smoothing the discrete landscape, these approaches enable

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

an unsupervised learning paradigm where models learn to exploit patterns and discover high-quality solutions without requiring expensive labels.

More specifically, we propose a generic learning-based pipeline that integrates hard combinatorial constraints directly into training and inference. The neural network is allowed to output an arbitrary continuous vector, without any feasibility requirement. This output is then passed through a geometric decomposition algorithm inspired by approximate Carathéodory and Frank–Wolfe methods Barman (2015); Mirrokni et al. (2017); Combettes & Pokutta (2023), which produces a sparse convex combination of feasible solutions. The resulting decomposition may differ substantially from the network output but is always expressed entirely in terms of feasible points. This effectively shifts the burden of constraint enforcement from tuning penalty terms to capturing constraints algorithmically through our decomposition algorithm. The decomposition induces a (a.e. differentiable) distribution over feasible solutions, and the expected value of the discrete objective under this distribution is used directly as a self-supervised training loss. In addition, the discrepancy between the network output and its decomposition can be quantified and included as an auxiliary regularization term, with an associated weight. While not required by the framework, we find empirically that including this term improves optimization stability and solution quality. Gradients propagate through this objective using automatic differentiation. At inference time, the same procedure yields a small set of feasible candidate solutions from which the best one is selected. See Figure 1.

Our pipeline is projection-free. The decomposition in Karalias et al. (2025) requires the neural prediction to lie inside the feasible polytope, with feasibility ensured separately through problem-specific constructions. For example, it uses distinct projections for the hypersimplex (Proposition 4.4) and the spanning-tree polytope (Theorem E.7). Our method removes the need for projection and accepts arbitrary ambient-space outputs and uses a Frank–Wolfe decomposition to construct a sparse convex combination of feasible vertices.

Moreover, our decomposition has an explicit run time certificate and uses fewer Linear Maximization Oracle (LMO) calls. It requires exactly one exact or approximate LMO call per iteration, so T iterations use exactly T LMO calls. In contrast, each iteration of the generic GLS (Groetschel/Lovasz/Schrijver) decomposition in Karalias et al. (2025) invokes minimal-face and ray-boundary optimization routines that themselves require solving convex optimization problems using the oracle; see the proof of Theorem 4.1. Consequently, a single GLS iteration may require many LMO calls. For example, a standard ellipsoid-based implementation requires approximately $O(n^2L)$ oracle calls per iteration and $O(n^3L)$ calls overall, where L is a precision parameter.

Although specialized GLS decompositions can be more efficient for particular polytopes, deriving them requires nontrivial, problem-specific techniques. Given an efficient exact or approximate LMO, our procedure applies unchanged across different polytopes, with only the LMO changing. It thus provides a unified oracle-based method for hypersimplex, Birkhoff, spanning-tree, and matching polytopes, subsuming the specialized settings in Karalias et al. (2025); Nerem et al. (2025).

Contributions. We make the following contributions:

- **A projection-free, self-supervised framework for CO.** We introduce a unified learning framework that enables neural networks to optimize combinatorial objectives under hard discrete constraints by operating over convex polytopes of feasible solutions. Unlike prior geometric-extension methods, our framework does *not* require network outputs to lie in the polytope during training and provides a common formulation across a broad class of combinatorial structures.
- **An oracle-based extension layer using Frank–Wolfe decomposition.** To realize this framework, we propose a Frank–Wolfe–style extension layer that maps an arbitrary continuous prediction to a sparse convex combination of feasible solutions, yielding a distribution over feasible discrete solutions. Given an efficient exact or approximate linear maximization oracle (LMO), the same procedure applies across different polytopes; only the implementation of the LMO changes. It requires one LMO call per decomposition iteration and avoids problem-specific projections and specialized decomposition algorithms.
- **A differentiable, self-supervised loss with built-in rounding guarantees.** The induced convex decomposition allows us to define a differentiable loss as the expected discrete objective, with approximation error explicitly incorporated. At inference time, selecting the best solution in the support yields a feasible solution whose objective value is no worse than the expectation. This

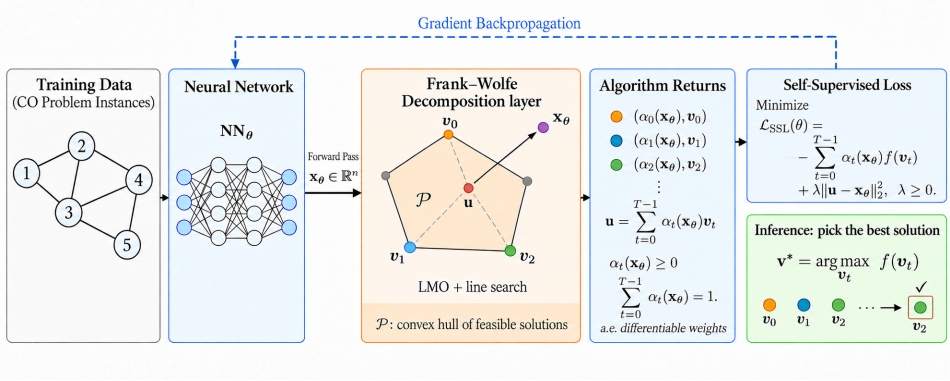


Figure 1: Overview of our framework. During training, the model with parameters θ outputs a point $\mathbf{x}_\theta$, which may lie outside $\mathcal{P}$, the convex hull of feasible solutions. Our Frank–Wolfe decomposition layer constructs a feasible proxy $\mathbf{u}$ as a convex combination of at most T feasible vertices $\mathbf{v}_t$, with weights $\alpha_t(\mathbf{x}_\theta)$. The self-supervised loss $\mathcal{L}_{\mathrm{SSL}}$ is computed from the expected discrete objective f and a reconstruction term weighted by $\lambda \geq 0$. The weights $\alpha_t(\mathbf{x}_\theta)$ are a.e. differentiable w.r.t $\mathbf{x}_\theta$, allowing gradients from the loss to propagate through the decomposition layer back to the neural network. During inference (green box), the same procedure generates feasible candidates, and the best solution $\mathbf{v}^*$ is selected. Arrows show the forward pass (solid) and backpropagation (dashed).

provides a direct rounding guarantee and unifies learning and solution recovery within a single pipeline.

- **Empirical validation across diverse combinatorial domains.** We demonstrate the effectiveness of our framework on a range of classical CO problems—including MC, TSP, and QAP—showing strong empirical performance and broad applicability across diverse constraint families.

2 Problem Formulation and Learning Setup

We consider CO problems defined over a discrete feasible set. Each problem instance is specified by a collection of variables $V = \{v_1, \ldots, v_n\}$, each assigned a value from a discrete domain $D = \{1, \ldots, d\}$, a finite set of feasible solutions $\mathcal{C} \subseteq D^n$, and a real-valued objective function $f : \mathcal{C} \to \mathbb{R}$. The goal is to find an optimal feasible solution $\mathbf{x}^* \in \mathcal{C}$ that optimizes (either minimizes or maximizes) the objective function:

$$\max f(\mathbf{x}), \quad \text{s.t.} \quad \mathbf{x} \in \mathcal{C}. \tag{1}$$

This abstract formulation accommodates a broad class of CO problems, including those defined over binary vectors, general discrete assignments, and permutations.

Many examples in this paper focus on problems with Boolean decision variables, where solutions can be represented by indicator vectors encoding subset or structure selection under constraints. Two representative running examples are Maximum Coverage, which selects a subset of elements subject to a budget constraint, and the TSP, where binary variables indicate whether edges are included in a Hamiltonian tour. Our framework also applies to CO problems over permutations, e.g., the QAP, and to non-Boolean discrete domains, e.g., k-submodular maximization.

A standard geometric perspective on CO is to consider the convex hull of all feasible solutions,

$$\mathcal{P} = \mathrm{conv}(\mathcal{C}).$$

For CO problems with Boolean domain, we have $\mathcal{P} = \mathrm{conv}(\mathcal{C}) \subseteq [0,1]^n$. As an example, when $\mathcal{C}$ consists of all subsets of size k, $\mathcal{P}$ is the associated cardinality polytope, a.k.a hypersimplex. Each $S \in \mathcal{C}$ is represented by its indicator vector $\mathbf{1}_S \in \{0,1\}^n$, where $(\mathbf{1}_S)_i = 1$ if element i belongs to S and $(\mathbf{1}_S)_i = 0$ otherwise. The polytope associated with $\mathcal{C}$ is then $\mathcal{P} := \mathrm{conv}\{\mathbf{1}_S : S \in \mathcal{C}\} \subseteq [0,1]^n$, whose vertices, i.e., extreme points $\mathrm{ext}(\mathcal{P})$, coincide with the indicator vectors of subsets of size k.

Although $\mathcal{P}$ may have an exponentially large facet description, many important combinatorial polytopes admit efficient linear optimization via polynomial-time separation or optimization oracles.

Self-Supervised Learning for CO. In the self-supervised learning setting, each problem instance $\mathcal{I}$ is described by input features $\mathbf{Z}_{\mathcal{I}}$ possibly together with instance-specific structure. These features are processed by a neural network NN_θ, which outputs a representation $\mathbf{x}_\theta$. Importantly, this output is not required to lie in the convex polytope of feasible solutions.

Rather than training the network to imitate optimal solutions, learning is guided by a loss function defined on an a.e. differentiable extension of the combinatorial objective. This extension is constructed using the geometry of the polytope $\mathcal{P}$, allowing the objective to be meaningfully evaluated on fractional solutions and optimized using gradient-based methods.

The network is trained over a collection of instances without access to ground-truth optimal solutions, making the design of the loss function central. At inference time, the continuous output $\mathbf{x}$ is converted into a feasible discrete solution using a principled, geometry-based procedure, rather than heuristic rounding. Moreover, because no labeled data is required, the same objective can be optimized directly on test instances, allowing test-time computation to improve solution quality.

3 Proposed Method

We now describe the overall pipeline. Consider a polytope $\mathcal{P} = \mathrm{conv}(\mathcal{C})$, where $\mathcal{C} \subset \mathbb{R}^n$ denotes the set of feasible solutions for a CO problem. Let $\mathrm{ext}(\mathcal{P}) \subseteq \mathcal{C}$ denote the set of extreme points i.e., corners, of $\mathcal{P}$ which correspond to feasible combinatorial solutions.

Given a target point $\mathbf{x} \in \mathbb{R}^n$, which can be thought of as the output of a neural network, our goal is to find a point $\mathbf{u} \in \mathcal{P}$ that serves as a good proxy for $\mathbf{x}$ and satisfies desirable properties. In particular, we want $\mathbf{u}$ to be a *sparse* convex combination of T points $\mathbf{v}_0, \ldots, \mathbf{v}_{T-1} \in \mathrm{ext}(\mathcal{P})$. That is, for a small T, we have

$$\mathbf{u}(\mathbf{x}) = \sum_{t=0}^{T-1} \alpha_t(\mathbf{x})\, \mathbf{v}_t \in \mathcal{P} \quad \text{with} \quad \begin{cases} \mathbf{v}_t \in \mathrm{ext}(\mathcal{P}), \\ \sum_{t=0}^{T-1} \alpha_t(\mathbf{x}) = 1;\ \alpha_t \geq 0. \end{cases} \tag{2}$$

The coefficients $\alpha_t(\mathbf{x})$ define a distribution $\mathcal{D}(\mathbf{x})$ on feasible solutions, with FW-induced objective

$$F(\mathbf{x}) = \sum_{t=0}^{T-1} \alpha_t(\mathbf{x})\, f(\mathbf{v}_t) = \mathbb{E}_{\mathbf{v}\sim\mathcal{D}(\mathbf{x})}[f(\mathbf{v})]. \tag{3}$$

Training. In the learning pipeline, let $\mathbf{x}_\theta$ denote the prediction of a neural network with parameters θ. For a maximization problem (1), we train the network by minimizing the self-supervised loss

$$\mathcal{L}_{\mathrm{SSL}}(\theta) := -F(\mathbf{x}_\theta) + \lambda \|\mathbf{x}_\theta - \mathbf{u}(\mathbf{x}_\theta)\|_2^2\,, \qquad \lambda \geq 0. \tag{4}$$

In order to optimize the loss function using gradient-based methods, the weights $\alpha_t(\mathbf{x}_\theta)$ must be differentiable functions of $\mathbf{x}_\theta$. Under this condition, derivatives of the loss with respect to the network parameters can be computed via the chain rule: $\frac{\partial F}{\partial \theta} = \frac{\partial F}{\partial \boldsymbol{\alpha}(\mathbf{x}_\theta)} \cdot \frac{\partial \boldsymbol{\alpha}(\mathbf{x}_\theta)}{\partial \mathbf{x}_\theta} \cdot \frac{\partial \mathbf{x}_\theta}{\partial \theta}$, where $\frac{\partial F}{\partial \alpha_t(\mathbf{x}_\theta)} = f(\mathbf{v}_t)$ and $\frac{\partial \boldsymbol{\alpha}(\mathbf{x}_\theta)}{\partial \mathbf{x}_\theta}$ comes from differentiating the steps our FW decomposition algorithm, Algorithm 1. Note that under the stable-oracle decision assumption introduced in Section 3.1, the selected vertices $\mathbf{v}_t$ are locally constant functions of $\mathbf{x}_\theta$ almost everywhere. Hence, $\frac{\partial \mathbf{v}_t}{\partial \mathbf{x}_\theta} = 0$ wherever the derivative exists, and the gradient propagates only through the coefficients. The gradient of the term $\lambda\|\mathbf{x}_\theta - \mathbf{u}(\mathbf{x}_\theta)\|_2^2$ is computed similarly by differentiating through $\mathbf{u}(\mathbf{x}_\theta) = \sum_{t=0}^{T-1} \alpha_t(\mathbf{x}_\theta)\mathbf{v}_t$.

In Section 3.1, we present our algorithm, which given $\mathbf{x}$ computes $\mathbf{u}(\mathbf{x}) \in \mathcal{P}$ satisfying the properties in (2). Moreover, we show that the weights $\alpha_t(\mathbf{x})$ returned by our algorithm are a.e. differentiable w.r.t. $\mathbf{x}$ and provide a convergence rate in T, the number of iterations of the algorithm. For notational simplicity, throughout Section 3.1 we suppress the dependence on the target point $\mathbf{x}$, writing $\mathbf{u}$ and α_t in place of $\mathbf{u}(\mathbf{x})$ and $\alpha_t(\mathbf{x})$.

Remark 3.1 (Extension property). The FW-induced objective F defines a surrogate on $\mathbb{R}^n$. For F to be an extension of $f : \mathcal{C} \to \mathbb{R}$, it must agree with f on $\mathcal{C}$. We therefore adopt the convention that Algorithm 1 returns the singleton decomposition $(1, \mathbf{x})$ whenever $\mathbf{x} \in \mathcal{C}$. Consequently, $F(\mathbf{c}) = f(\mathbf{c})$ for every $c \in \mathcal{C}$. Since $\mathcal{C}$ is finite, this convention modifies F only on a set of Lebesgue measure zero and therefore does not affect its almost-everywhere differentiability.

3.1 FROM NN OUTPUT TO POLYTOPE: FRANK-WOLFE DECOMPOSITION

In this section, we describe Algorithm 1. Given $\mathbf{x}$, define the following quadratic objective:

$$J(\mathbf{u}) = \tfrac{1}{2}\|\mathbf{x} - \mathbf{u}\|_2^2, \qquad \mathbf{u} \in \mathcal{P}. \tag{5}$$

Starting at an initial point $\mathbf{u}_0$ in the polytope our goal is to iteratively move towards $\mathbf{u}_1, \mathbf{u}_2, .., \mathbf{u}_{T-1}$ so that at each iteration we decrease $J(\mathbf{u}_t)$, and at the end $\mathbf{u}_{T-1}$ has the desired aforementioned properties. More specifically, we start with an initial point $\mathbf{u}_0$ which is a corner of the polytope $\mathcal{P}$, i.e. $\mathbf{v}_0 = \mathbf{u}_0 \in \mathrm{ext}(\mathcal{P})$. Then, in the first iteration, we find another corner of the polytope $\mathcal{P}$, say $\mathbf{v}_1$ and set $\mathbf{u}_1 = (1-\gamma_1)\mathbf{u}_0 + \gamma_1\mathbf{v}_1$ with an appropriate value $\gamma_1 \in [0,1]$, to be determined later. By construction, $\mathbf{u}_1 \in \mathcal{P}$ as it is a convex combination of two corners of the polytope. This iterative process is repeated and at every iteration t we have

$$\mathbf{u}_t = (1-\gamma_t)\mathbf{u}_{t-1} + \gamma_t\mathbf{v}_t \tag{6}$$

Algorithm 1 FW DECOMPOSITION $(\mathbf{x}, \mathcal{P}, T; \mathrm{LMO})$

Input: Target $\mathbf{x} \in \mathbb{R}^n$, polytope $\mathcal{P}$, support budget T, and an exact or approximate LMO.
Initialize: $\mathbf{v}_0 = \mathbf{u}_0 = \arg\max_{\mathbf{v}\in\mathcal{P}}\langle \mathbf{x}, \mathbf{v}\rangle$, $\alpha_0 \leftarrow 1$.
for $t = 1$ to $T-1$ **do**
  $\mathbf{r}_t \leftarrow \mathbf{x} - \mathbf{u}_{t-1}$ {Residual}
  $\mathbf{v}_t \leftarrow \arg\max_{\mathbf{v}\in\mathcal{P}}\langle \mathbf{r}_t, \mathbf{v}\rangle$ {Find a corner via LMO}
  $\mathbf{d}_t \leftarrow \mathbf{v}_t - \mathbf{u}_{t-1}$ {Descent direction}
  $\gamma_t \leftarrow \mathrm{Clip}(0, 1, \langle \mathbf{r}_t, \mathbf{d}_t\rangle / \|\mathbf{d}_t\|_2^2)$ {Line search step}
  $\alpha_j \leftarrow (1-\gamma_t)\alpha_j$ for all existing weight α_j, $j \in \{0, \ldots, t-1\}$.
  $\alpha_t \leftarrow \gamma_t$
  $\mathbf{u}_t \leftarrow (1-\gamma_t)\mathbf{u}_{t-1} + \gamma_t\mathbf{v}_t$
end for
Output: Weights and vertices $\{(\alpha_t, \mathbf{v}_t)\}_{t=0}^{T-1}$ such that $\mathbf{u}_{T-1} = \sum_{t=0}^{T-1}\alpha_t\mathbf{v}_t$.

with an appropriate value for $\gamma_t \in [0,1]$ and $\mathbf{v}_t \in \mathrm{ext}(\mathcal{P})$. The algorithm terminates after $T-1$ iterations and returns $\mathbf{u}_{T-1}$, which will lie in $\mathcal{P}$ by construction. At this point, $\mathbf{u}_{T-1}$ is a convex combination of $\mathbf{u}_0 = \mathbf{v}_0, \ldots, \mathbf{v}_{T-1}$ with weights $\alpha_0, \ldots, \alpha_{T-1}$ that are calculated recursively with respect to γ_ts. Specifically, $\sum_{t=0}^{T-1}\alpha_t = 1$, with $\alpha_0 = \prod_{i=1}^{T-1}(1-\gamma_i)$ and $\alpha_t = \gamma_t\prod_{i=t+1}^{T-1}(1-\gamma_i)$.

The main two points to address here are which corner $\mathbf{v}_t$ to choose and what is a good value for step size γ_t.

Note that the function $J(\mathbf{u})$ is convex and its gradient is $\nabla J(\mathbf{u}) = \mathbf{u} - \mathbf{x}$. We define the **residual** at iteration t as $\mathbf{r}_t := \mathbf{x} - \mathbf{u}_{t-1} = -\nabla J(\mathbf{u}_{t-1})$, which points from our current estimate toward the target. Frank–Wolfe is a "projection-free" method. At every iteration, instead of moving toward the target in the gradient direction and then projecting back into $\mathcal{P}$ (which can be computationally expensive), it finds the vertex $\mathbf{v}_t$ of the polytope that lies furthest in the direction of the residual $\mathbf{r}_t$. It then takes a small step γ_t from the current point $\mathbf{u}_{t-1}$ toward that vertex $\mathbf{v}_t$. Since both $\mathbf{u}_{t-1}$ and $\mathbf{v}_t$ are in $\mathcal{P}$, their convex combination is guaranteed to remain inside $\mathcal{P}$. The exact procedure is shown in Algorithm 1, and in order to find $\mathbf{v}_t$ we require what is known as LMO.

Definition 3.2 (Linear Maximization Oracle (LMO))**.** Given a residual $\mathbf{r} \in \mathbb{R}^n$, an exact LMO returns a vertex $\mathbf{v}^\star$ that maximizes the alignment with $\mathbf{r}$:

$$\mathbf{v}^\star = \arg\max\langle \mathbf{r}, \mathbf{v}\rangle \quad \text{s.t.} \quad \mathbf{v} \in \mathcal{P} \tag{7}$$

A fundamental property of linear optimization is that a linear objective over a polytope attains an optimum at a vertex. Hence, an LMO returns a (integral) vertex of the polytope. When exact maximization is difficult, the convergence statement below uses a relative Frank–Wolfe gap approximation Jaggi (2013); Lacoste-Julien et al. (2013); Locatello et al. (2017); Zhou & Sun (2022). Formally, at iteration t, a δ-relative FW-gap oracle returns $\mathbf{v}_t \in \mathrm{ext}(\mathcal{P})$ satisfying

$$\langle \mathbf{r}_t, \mathbf{v}_t - \mathbf{u}_{t-1}\rangle \;\geq\; \delta \max_{\mathbf{v}\in\mathcal{P}}\langle \mathbf{r}_t, \mathbf{v} - \mathbf{u}_{t-1}\rangle, \tag{8}$$

where $\mathbf{r}_t = \mathbf{x} - \mathbf{u}_{t-1}$ and $\delta \in (0,1]$. This is the approximation notion used in our convergence statement. Exact LMOs satisfy (8) with $\delta = 1$.

For the step size γ_t, several choices are possible. We compute it using a simple line-search:

$$\gamma_t = \mathrm{Clip}(0, 1, \langle \mathbf{r}_t, \mathbf{d}_t\rangle / \|\mathbf{d}_t\|_2^2),$$

with the convention $\gamma_t = 0$ when $\mathbf{d}_t = 0$. This allows for efficient computation and supports branchwise differentiation of the weights $\{\alpha_t\}$.

Assumption (stable oracle decisions). Fix the support budget T. We assume that the deterministic oracle and tie-breaking rule do not switch pathologically. The set of inputs $\mathbf{x}$ at which an arbitrarily small perturbation can change one of the selected vertices, or change whether a line-search step is clipped, has zero volume. Equivalently, for almost every input $\mathbf{x}$, there is a small neighborhood around $\mathbf{x}$ on which Algorithm 1 selects the same vertices $\mathbf{v}_0, \ldots, \mathbf{v}_{T-1}$ and uses the same line-search clipping cases. This assumption rules out artificial rules that switch between equally valid approximate vertices on dense sets. It is satisfied by the standard comparison-based oracles used in combinatorial optimization, including top-k selection, matroid greedy algorithms such as Kruskal's algorithm, and exact assignment or matching oracles with fixed tie-breaking.

All these results together yield the following theorem. The convergence guarantee is the standard exact Frank–Wolfe rate, and the a.e. differentiability proof is given in Section B.

Theorem 3.3 (Exact Frank–Wolfe Decomposition)**.** *Let $\mathbf{x} \in \mathbb{R}^n$, and let $\mathcal{P} \subset \mathbb{R}^n$ be a polytope with diameter $D := \operatorname{diam}(\mathcal{P}) = \max_{\mathbf{u},\mathbf{v}\in\mathcal{P}} \|\mathbf{u}-\mathbf{v}\|_2$. Run Algorithm 1 with an exact LMO and support budget $T \geq 2$, so that it performs $T-1$ Frank–Wolfe updates. Then the algorithm outputs $\mathbf{u}_{T-1} = \sum_{t=0}^{T-1} \alpha_t \mathbf{v}_t \in \mathcal{P}$, where $\mathbf{v}_t \in \operatorname{ext}(\mathcal{P})$, $\alpha_t \geq 0$, and $\sum_{t=0}^{T-1} \alpha_t = 1$. Moreover, if $\mathbf{u}^\star = \arg\min_{\mathbf{u}\in\mathcal{P}} J(\mathbf{u})$, then $J(\mathbf{u}_{T-1}) - J(\mathbf{u}^\star) \leq \frac{2D^2}{T+1}$. The coefficients α_t are almost everywhere differentiable functions of $\mathbf{x}$.*

Corollary 3.4 (Approximate Frank–Wolfe Decomposition)**.** *If the exact LMO in Theorem 3.3 is replaced by a δ-relative FW-gap oracle satisfying* (8)*, then the same convex-combination conclusion holds. Let $D = \operatorname{diam}(\mathcal{P})$, $\mathbf{u}^\star = \arg\min_{\mathbf{u}\in\mathcal{P}} J(\mathbf{u})$, and $h_0 = J(\mathbf{u}_0) - J(\mathbf{u}^\star)$. Then $J(\mathbf{u}_{T-1}) - J(\mathbf{u}^\star) \leq \frac{2\left(D^2/\delta + h_0\right)}{\delta(T-1)+2}$. The coefficient map remains almost everywhere differentiable.*

Discussion on projection onto the polytope. In the above discussion we did not require neural network output to lie in the polytope, yielding a generic projection-free framework. Nevertheless, when this occurs—either by design of the neural network or because it is empirically advantageous—our framework aligns with *approximate Carathéodory* results. When $\mathbf{x} \in \mathcal{P}$, minimizing $J(\mathbf{u}) = \frac{1}{2}\|\mathbf{x}-\mathbf{u}\|_2^2$ via Algorithm 1 produces a sparse convex combination of vertices of $\mathcal{P}$ approximating $\mathbf{x}$. Constructing such sparse convex combinations is the goal of approximate Carathéodory results. Unlike the exact Carathéodory theorem, the approximate variants relax exact representation in favor of an additive error to achieve sparser convex combinations Pisier (1981); Barman (2015); Mirrokni et al. (2017). In the exact-LMO setting, Theorem 3.3 yields the approximate Carathéodory bound $\|\mathbf{x}-\mathbf{u}_{T-1}\|_2^2 \leq \frac{4D^2}{T+1}$.

4 Applications and Experiments

We evaluate our framework on three core CO problems—MC, QAP, and TSP—and show that the same decomposition and extension apply across all cases. Our method is competitive with and often outperforms SOTA approaches. Compared to SOTA approaches, which rely on problem-specific and novel engineered constructions, this generality is a key advantage. For testing methods and choice of baselines, we keep a strict one-shot setting (e.g., without Test Time Optimization (TTO)) to ensure fairness, detailed in Section F. Code and data are available in link.

4.1 Maximum Coverage

Let $\mathcal{U}$ be a ground set of elements with weights $w_u \geq 0$ for $u \in \mathcal{U}$, and let $\{S_1, \ldots, S_n\}$ be subsets $S_i \subseteq \mathcal{U}$. The value of $\mathbf{x} \in \{0,1\}^n$ is the total weight of covered elements, $f_{\text{cov}}(\mathbf{x}) = \sum_{u\in\mathcal{U}} w_u \, \mathbf{1}[\exists\, i \in [n] \text{ with } u \in S_i \text{ and } \mathbf{x}(i) = 1]$. The MC problem under cardinality constraint is $\max_{\mathbf{x}\in\{0,1\}^n} f_{\text{cov}}(\mathbf{x})$ subject to $\sum_{i=1}^n \mathbf{x}(i) = k$, where k denotes the prescribed cardinality.

Neural model and continuous targets. This problem can naturally be represented as a bipartite graph where the goal is to select k nodes from one part so that we maximize the total number of their neighbors in the other part. We apply a GAT-based encoder to obtain node embeddings. This $\mathbf{x}_\theta$ is the input to our FW Decomposition algorithm to obtain a small list of subsets $\{S^{(t)}\}_{t=0}^{T-1}$, of size k, and coefficients $\{\alpha_t\}_{t=0}^{T-1}$. For training, we minimize the differentiable surrogate loss $\mathcal{L}_{\text{COV}} = -\sum_{t=0}^{T-1} \alpha_t \, f_{\text{cov}}(\mathbf{1}_{S^{(t)}}) + \lambda \|\mathbf{x}_\theta - \sum_{t=0}^{T-1} \alpha_t \mathbf{1}_{S^{(t)}}\|_2^2$ and backpropagate through the FW

Table 1: MC performance comparison across three datasets, reporting mean inference time and coverage across k. Among neural methods, ■ indicates ranking the 1st, ■ the 2nd in each column.

	Random500				Random1000				Rail			
	$k = 10$		$k = 50$		$k = 20$		$k = 100$		$k = 20$		$k = 50$	
Method	Time (s)	Coverage	Time (s)	Coverage	Time (s)	Coverage	Time (s)	Coverage	Time (s)	Coverage	Time (s)	Coverage
Random (240s)	240.00	13372.76	240.00	36786.89	240.00	24133.50	240.00	70527.31	240.00	5291.67	240.00	7367.00
Gurobi (120s)	11.574	15714.90	120.065	44880.59	37.298	31347.62	120.139	89696.83	121.059	5631.67	121.033	7604.67
Greedy	0.057	15640.99	0.121	44597.56	0.190	31105.89	0.460	88685.40	0.727	5617.00	1.320	7630.00
Ucom2-short	1.041	15253.35	0.739	44208.95	1.734	29791.66	1.621	88472.61	2.299	5512.67	2.454	7594.67
GeoNCO	0.579	15177.77	1.108	41247.80	0.825	29810.12	1.908	81357.29	1.856	5343.00	2.084	7411.67
CardNN-noTTO-S	1.688	9231.54	1.721	33055.87	1.869	18458.92	1.881	65793.40	1.929	5074.33	2.189	7193.00
EGN-naive	53.041	15262.76	120.136	41272.68	120.010	29968.04	120.356	81166.12	120.676	5234.67	121.335	7408.33
RL (GNN+Actor)	0.069	14741.26	0.247	39510.96	0.102	29912.14	0.465	81158.54	0.219	5137.00	0.219	7456.67
FWNCO (ours)	0.426	15192.64	0.555	42172.44	0.647	29926.63	0.913	83559.22	1.432	5344.67	1.931	7393.00

steps into the GAT parameters. Following Karalias et al. (2025) (**GeoNCO**), we train on synthetic random graphs and test on real-world graphs. For evaluation, Table 1, we report $\max_t f_{\text{cov}}(\mathbf{1}_{S^{(t)}})$ without TTO and compare with the baselines without TTO. Our method is faster than **GeoNCO** and it often outperforms the learning baselines. Specifically, it consistently stays on the pareto front over the learning baselines and is also competitive against non-learning baselines considering the quality-efficiency tradeoff. (See Section C.4 for adversarial instances where greedy fails.)

Ablation on projection and T for MC. Table 4 shows that removing the polytope projection preserves solution quality while improving runtime. Although the model converges to a point outside the hypersimplex, the solution quality from our decomposition is slightly better than the projected version. This suggests the projection-free variant allows greater exploration by avoiding hard constraints. See Appendix C.5 for an ablation on T.

Ablation on comparison with Greedy. Greedy is a strong baseline for MC: because the coverage objective is monotone submodular and the constraint is a cardinality budget, Greedy guarantees at least a $1 - 1/e$ fraction of the optimal coverage. To test FWNCO on instances where Greedy struggles, we use GreedyTrap instances designed to bring Greedy's performance close to this worst-case bound. Table 5 shows that FWNCO achieves an average coverage improvement of 21.3% over Greedy across the four settings.

4.2 Quadratic Assignment Problem

There are n facilities and n locations. Distances between locations are given by a matrix $\mathbf{B} \in \mathbb{R}^{n\times n}$, and flows between facilities by a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$. The goal is to assign each facility to a distinct location so as to minimize the total flow-weighted distance. Let Π_n denote the set of $n \times n$ permutation matrices. The objective is $\min_{\mathbf{P}\in\Pi_n} f_{\text{qap}}(\mathbf{P})$ where $f_{\text{qap}}(\mathbf{P}) = \text{trace}(\mathbf{APBP}^\top)$.

Neural model and continuous targets. Given the output of encoder network, GraphSAGE, $\mathbf{X}_\theta \in [0,1]^{n\times n}$, we apply our FW Decomposition algorithm using a deterministic greedy algorithm for weighted maximum matching. After T iterations, it returns a short list of permutation matrices $\{\mathbf{P}_t\}_{t=0}^{T-1}$ and coefficients $\{\alpha_t\}_{t=0}^{T-1}$. We train directly against the discrete QAP objective and using the following $\mathcal{L}_{\text{QAP}}(\mathbf{X}_\theta) = \sum_{t=0}^{T-1} \alpha_t f_{\text{qap}}(\mathbf{P}_t) + \lambda\|\mathbf{X}_\theta - \sum_{t=0}^{T-1} \alpha_t \mathbf{P}_t\|_F^2$. We backpropagate through FW to the parameters producing $\mathbf{X}_\theta$. At evaluation we report $\min_t f_{\text{qap}}(\mathbf{P}_t)$.

We train on synthetic instances following Tan & Mu (2024). For size n, we sample locations $\mathbf{L} \sim \text{Uniform}(0,1)^2$ and a symmetric flow matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ with zero diagonal, upper-triangular entries sampled from $[0,1]$, and entries independently set to zero with probability p (symmetrically). We test on **QAPLIB** benchmark Burkard et al. (1997) (134 instances, 15 classes; see Section D.1).

Consistent with prior studies, we report per-class and aggregated gap statistics on instances of size 12–64, where baseline methods are tractable. Recent learning-based QAP methods, such as **RGM** Liu et al. (2023) and **SAWT** Tan & Mu (2024), rely on iterative Learn-to-Construct or Learn-to-Improve pipelines which could be time consuming and prevent scalability. Our approach, on the other hand, produces a permutation in a single forward pass, which compared to the learning baselines, yields better efficiency, while achieving a better out of distribution (OOD) generalization.

Table 2: Mean gaps and inference time (s) for models pre-trained on QAP32 and applied to QAPLIB instances. **Std.** is the standard deviation of the class-wise mean gaps across the 14 data classes (bur–wil). In each column, ■ indicates ranking the 1st, ■ the 2nd.

Method	bur	chr	esc	had	kra	lipa	nug	rou	scr	sko	ste	tai	tho	wil	Avg.	Std.	Time (s)
SM	22.3	460.1	301.6	17.4	65.3	19.0	45.5	35.8	123.4	29.0	475.5	180.5	55.0	13.8	181.2	163.3	0.01
RRWM	23.1	616.0	63.9	25.1	58.8	20.9	67.8	51.2	173.5	48.5	539.4	197.2	80.6	18.2	169.5	192.9	0.15
SK-JA	4.7	38.5	364.8	25.8	41.4	0.0	25.3	13.7	48.6	18.3	120.4	25.2	32.9	8.8	93.2	93.9	563.4
NGM	3.4	121.3	126.7	8.2	31.6	16.2	21.0	30.9	55.5	25.2	101.7	61.4	27.5	10.8	62.4	41.9	15.72
RGM	7.1	112.4	32.8	6.2	15.0	13.3	9.7	13.4	45.5	10.6	134.1	17.3	20.7	8.1	35.8	40.4	75.53
SAWT	2.8	110.7	13.5	3.8	30.1	0.4	9.2	10.8	28.5	17.7	93.5	16.5	24.8	8.1	26.8	33.5	12.11
FWNCO (ours)	2.3	97.7	19.0	2.8	37.0	13.5	11.5	10.9	23.6	15.0	83.5	20.7	23.0	9.2	26.4	28.8	1.56

In our experiments, Table 2, our method achieves the best overall performance among all baselines, including both learning-based and classical solvers. The consistently low standard deviation of the average optimality gap across instance classes indicates stable behavior and good generalization across distributions. Moreover, our approach is the fastest neural method, running noticeably faster than other neural baselines. The only two methods with lower runtime, **SM** and **RRWM**, are purely traditional heuristics, which come at a substantial loss in solution quality. While Table 2 reports sizes 12–64, our method scales to larger instances (> 64) with similar average gaps (Table 8). Most prior methods do not scale to such sizes with reasonable compute resources.

Ablation on projection and T for QAP. Algorithm 1 does not theoretically require its input to be in the Birkhoff polytope. However, we observed a weaker OOD generalization results for QAP due to the strong heterogeneity and distribution shift present in **QAPLIB**. Unconstrained outputs, i.e., "far from the polytope", cause the decomposition algorithm to return low quality permutations under drastic distributional shift. We therefore include a lightweight Sinkhorn projection layer that enforces approximate doubly-stochasticity, improving generalization. This step is introduced for practical considerations, not theoretical necessity. See Appendix D.4 for an ablation on T.

4.3 TSP via Spanning Trees and Learned Matchings

We consider spanning tree polytopes, where $\mathcal{C}$ is the set of spanning trees of a graph. For each $T \in \mathcal{C}$, the objective $f_{\text{TSP}}(T)$ is the cost of the TSP tour obtained deterministically via Christofides's algorithm Christofides (1976). We optimize a spanning-tree distribution for this objective.

Algorithm-guided neural optimization for TSP. One of the appeals of our approach is that it allows us to infuse learnable NN modules into an algorithmic framework. We adopt an algorithmically aligned, data-driven approach that integrates learning into the classical Christofides–Serdyukov algorithmic framework. Christofides's algorithm builds a TSP solution from a minimum spanning tree, whereas recent advances (e.g., Asadpour et al. (2017); Karlin et al. (2021; 2023)) sample from spanning tree distributions constructed from a subtour LP solution before applying variants of the Christofides pipeline. Our approach can be viewed as a learning-based analogue of this paradigm: instead of constructing such distributions analytically, we learn a distribution over spanning trees in an end-to-end manner, optimized directly for the final TSP objective. Moreover, we extend this idea further by incorporating a learning module for the matching step, effectively learning over matching polytopes as well. Given the neural network output, FW Decomposition returns a sparse distribution over spanning trees using Kruskal's algorithm as LMO. We sample trees from this distribution, and recover tours by applying minimum-weight perfect matching on odd degree nodes followed by shortcutting. This pipeline defines a differentiable surrogate objective that mirrors the Christofides algorithm end to end. This also allows the perfect-matching step itself to be learned jointly, improving solution quality.

We consider Euclidean TSP instances $(G = (V, E), D)$, where V is a set of points drawn uniformly at random from the unit square $[0, 1]^2$, E is the complete edge set on V, and D is the corresponding symmetric distance matrix induced by Euclidean distances, which satisfies the triangle inequality. A tour is a Hamiltonian cycle that visits each node exactly once, and its cost is given by the sum of the lengths of its constituent edges under D. The objective is to find a tour of minimum total cost.

Neural TSP methods commonly fall into two paradigms Ma et al. (2025): Learning-to-Construct (LC), which builds TSP tours sequentially with hard feasibility checks but suffers from high in-

Table 3: Performance comparison on TSP benchmarks across different scales. Results include optimality gap (%) and inference time. Methods are categorized as Unsupervised Learning (UL), Exact, Heuristics, Reinforcement Learning (RL), and Supervised Learning (SL). Among neural methods, indicates ranking the 1st, the 2nd in each column.

		TSP-50		TSP-100		TSP-500		TSP-1000	
Method	**Type**	Gap	Time	Gap	Time	Gap	Time	Gap	Time
Concorde (Optimal)	Exact	*opt*	0.04s	*opt*	0.21s	*opt*	17.91s	*opt*	435.61s
Gurobi (Optimal)	Exact	*opt*	0.27s	*opt*	1.14s	N/A	N/A	N/A	N/A
LKH3 (500)	Heuristics	0.01%	0.03s	0.01%	0.05s	0.93%	0.32s	1.30%	1.06s
DIFUSCO	SL	0.53%	0.43s	1.24%	0.66s	9.79%	2.19s	10.90%	7.68s
COExpander ($S = 1, D_s = 3, I_s = 5$)	SL	0.02%	0.11s	0.07%	0.20s	3.80%	0.69s	6.53%	2.50s
DIMES	RL	13.71%	0.05s	11.60%	0.08s	15.34%	0.39s	13.97%	0.79s
RL4CO (Sym-NCO)	RL	1.23%	0.36s	13.78%	0.66s	N/A	N/A	N/A	N/A
FWNCO (ours)	UL	1.81%	0.23s	3.48%	0.26s	7.30%	1.27s	8.27%	4.26s
FWNCO+learned matching (Ours)	UL	1.40%	0.52s	2.84%	0.74s	6.90%	2.77s	7.43%	9.16s

ference latency on large instances due to its inherently sequential decoding, and Global Prediction (GP), which predicts a full structure (e.g., a heatmap) and relies on post-processing, yielding fast inference but strong sensitivity to recovery heuristics. COExpander Ma et al. (2025) combines LC and GP via Adaptive Expansion (AE), but still depends on heatmap recovery, which still is a combinatorial optimization subroutine, inheriting GP's core limitation. Motivated by critiques of heatmap-guided post-hoc search Xia et al. (2024), our method directly optimizes feasible tours end-to-end by differentiating through the selected FW branch, reducing reliance on hand-crafted post-processing.

We compare against four baseline categories: (i) exact solvers (**Concorde**, **Gurobi**); (ii) classical heuristics (**LKH3** Helsgaun (2017), 500 trials); (iii) neural LC methods (**Sym-NCO** Kim et al. (2022)), implemented in the RL4CO library Berto et al. (2025); and (iv) neural GP/AE and heatmap-based methods across RL and supervised settings, including **DIMES** Qiu et al. (2022), **DIFUSCO** Sun & Yang (2023), and **COExpander**, a SOTA AE method. In Section F, we provide discussion regarding approaches that use inference optimization techniques such as Gradient-based Active Search, and Search-based Refinement such as MCTS and 2-opt.

Table 3 shows that our method consistently outperforms unsupervised neural baselines including RL. On larger instances, it also surpasses **DIFUSCO**, a strong diffusion-based approach trained with supervision. While we are still behind the strongest supervised baselines, e.g. **COExpander**, our approach is more resource-efficient. In particular, **DIFUSCO** and **COExpander** rely on up to $\sim$1.5M labeled instances (depending on the instance size) and require extensive training time on good hardware, whereas our model can be trained in under one hour on a single outdated GPU.

Ablation for TSP. We examine the effect of projecting the neural network output onto the spanning tree polytope and find that removing projection slightly improves performance on both TSP-500 and TSP-1000 (Table 13). We also study generalization across instance sizes and observe strong cross-size generalization (Table 12). Our model uses subtour LP solutions Dantzig et al. (1954) as features, obtained via the QSopt LP solver, also used by Concorde. Despite this shared solver, our learning pipeline is significantly faster on large instances. Our ablation study, Section E.7, confirm that these gains arise from learned structures rather than arbitrary perturbations of the LP relaxation.

5 Conclusion

We propose a general framework for self-supervised CO under constraints. Our approach is flexible in that it does not require the neural network output to lie in a feasible polytope. It uses a generic procedure, to decompose the output and obtain a sparse distribution over feasible solutions, effectively enabling SSL. Our method shows strong empirical performance compared to neural baselines on several CO problems. We present an example of infusing learned modules within an algorithm in an end-to-end pipeline, showcasing this approach on TSP using the Christofides algorithm. We believe this paves the way toward new NN-infused algorithms that can effectively tackle complex CO problems. Model architecture can affect performance; we leave its study to future work.

AI USE STATEMENT

We used generative AI tools to polish the writing and assist with implementing our method and experiments. We did **not** use AI to develop research ideas, formulate mathematical claims, write proofs, design experiments, or interpret results. The authors reviewed all AI-edited text, and one author reviewed and tested all AI-generated code for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The appendix provides experimental details, including training data, hyperparameters, and baseline configurations. Code and data are available in link.

REFERENCES

Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Learning to solve circuit-sat: An unsupervised differentiable approach. In *International Conference on Learning Representations (ICLR)*, 2018.

David L Applegate, Robert E Bixby, Vašek Chvátal, and William J Cook. The traveling salesman problem: a computational study. In *The Traveling Salesman Problem*. Princeton university press, 2011.

Arash Asadpour, Michel X Goemans, Aleksander Madry, Shayan Oveis Gharan, and Amin Saberi. An o (log n/log log n)-approximation algorithm for the asymmetric traveling salesman problem. *Operations Research*, 65(4):1043–1061, 2017.

Siddharth Barman. Approximating nash equilibria and dense bipartite subgraphs via an approximate version of caratheodory's theorem. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pp. 361–369, 2015.

Federico Berto, Chuanbo Hua, Junyoung Park, Laurin Luttmann, Yining Ma, Fanchen Bu, Jiarui Wang, Haoran Ye, Minsu Kim, Sanghyeok Choi, et al. Rl4co: an extensive reinforcement learning for combinatorial optimization benchmark. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 5278–5289, 2025.

Maximilian Böther, Otto Kißig, Martin Taraz, Sarel Cohen, Karen Seidel, and Tobias Friedrich. What's wrong with deep learning in tree search for combinatorial optimization. In *International Conference on Learning Representations (ICLR)*, 2022.

Fanchen Bu, Hyeonsoo Jo, Soo Yong Lee, Sungsoo Ahn, and Kijung Shin. Tackling prevalent conditions in unsupervised combinatorial optimization: Cardinality, minimum, covering, and more. In *41st International Conference on Machine Learning (ICML)*, pp. 4696–4729, 2024.

Rainer E Burkard, Stefan E Karisch, and Franz Rendl. Qaplib–a quadratic assignment problem library. *Journal of Global optimization*, 10(4):391–403, 1997.

M. D. Canon and C. D. Cullum. A tight upper bound on the rate of convergence of Frank–Wolfe algorithm. *SIAM Journal on Control*, 6(4):509–516, 1968.

Constantin Carathéodory. Über den variabilitätsbereich der koeffizienten von potenzreihen, die gegebene werte nicht annehmen. *Mathematische Annalen*, 64(1):95–115, 1907.

Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, 1976.

Cyrille W. Combettes and Sebastian Pokutta. Revisiting the approximate Carathéodory problem via the Frank-Wolfe algorithm. *Mathematical Programming*, 197(1):191–214, January 2023. ISSN 1436-4646.

Yves Crama. Concave extensions for nonlinear 0–1 maximization problems. *Mathematical Programming*, 61(1):53–60, 1993.

George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

Priya L. Donti, David Rolnick, and J Zico Kolter. DC3: A learning method for optimization with hard constraints. In *International Conference on Learning Representations (ICLR)*, 2021.

James E Falk and Karla R Hoffman. A successive underestimation method for concave minimization problems. *Mathematics of operations research*, 1(3):251–259, 1976.

Marguerite Frank, Philip Wolfe, et al. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956.

Dan Garber and Ofer Meshi. Linear-memory and decomposition-invariant linearly convergent conditional gradient algorithm for structured polytopes. *Advances in neural information processing systems*, 29, 2016.

Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6): 1115–1145, 1995.

Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12:966–980, 2017.

Martin Jaggi. Revisiting Frank–Wolfe: Projection-free sparse convex optimization. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pp. 427–435. PMLR, 2013.

Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. *Advances in Neural Information Processing Systems*, 33: 6659–6672, 2020.

Nikolaos Karalias, Joshua Robinson, Andreas Loukas, and Stefanie Jegelka. Neural set function extensions: Learning with discrete functions in high dimensions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Nikolaos Karalias, Akbar Rafiey, Yifei Xu, Zhishang Luo, Behrooz Tahmasebi, Connie Jiang, and Stefanie Jegelka. Geometric algorithms for neural combinatorial optimization with constraints. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Anna R Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric tsp. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pp. 32–45, 2021.

Anna R Karlin, Nathan Klein, and Shayan Oveis Gharan. A deterministic better-than-3/2 approximation algorithm for metric tsp. In *International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pp. 261–274. Springer, 2023.

Nicolas Keriven and Samuel Vaiter. What functions can graph neural networks compute on random graphs? the role of positional encoding. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:11823–11849, 2023.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 6348–6358, 2017.

Minsu Kim, Junyoung Park, and Jinkyoo Park. Sym-nco: Leveraging symmetricity for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 35:1936–1949, 2022.

Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 12. MIT Press, 1999.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations (ICLR)*, 2019.

Rahul G Krishnan, Simon Lacoste-Julien, and David Sontag. Barrier frank-wolfe for marginal inference. *Advances in Neural Information Processing Systems*, 28, 2015.

Yam Kushinsky, Haggai Maron, Nadav Dym, and Yaron Lipman. Sinkhorn algorithm for lifted assignment problems. *SIAM Journal on Imaging Sciences*, 12(2):716–735, 2019.

Simon Lacoste-Julien and Martin Jaggi. On the global linear convergence of frank-wolfe optimization variants. *Advances in Neural Information Processing Systems (NeurIPS)*, 28, 2015.

Simon Lacoste-Julien, Martin Jaggi, Mark Schmidt, and Patrick Pletscher. Block-coordinate Frank-Wolfe optimization for structural SVMs. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, Proceedings of Machine Learning Research. PMLR, 2013.

Xinpeng Li, Enming Liang, and Minghua Chen. Gauge flow matching for efficient constrained generative modeling over general convex set. In *ICLR 2025 Workshop on Deep Generative Model in Machine Learning: Theory, Principle and Efficacy*, 2025.

Enming Liang and Minghua Chen. Efficient bisection projection to ensure nn solution feasibility for optimization over general set.

Enming Liang, Minghua Chen, and Steven H Low. Low complexity homeomorphic projection to ensure neural-network solution feasibility for optimization over (non-) convex set. In *International Conference on Machine Learning (ICML)*, pp. 20623–20649, 2023.

Enming Liang, Minghua Chen, and Steven H Low. Homeomorphic projection to ensure neural-network solution feasibility for constrained optimization. *Journal of Machine Learning Research*, 25(329):1–55, 2024.

Derek Lim, Joshua David Robinson, Lingxiao Zhao, Tess Smidt, Suvrit Sra, Haggai Maron, and Stefanie Jegelka. Sign and basis invariant networks for spectral graph representation learning. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023.

Chang Liu, Zetian Jiang, Runzhong Wang, Lingxiao Huang, Pinyan Lu, and Junchi Yan. Revocable deep reinforcement learning with affinity regularization for outlier-robust graph matching. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023.

Francesco Locatello, Rajiv Khanna, Michael Tschannen, and Martin Jaggi. A unified optimization view on generalized matching pursuit and frank-wolfe. In *Artificial intelligence and statistics (AISTAT)*, pp. 860–868. PMLR, 2017.

László Lovász. Submodular functions and convexity. In *Mathematical Programming The State of the Art: Bonn 1982*, pp. 235–257. Springer, 1983.

Jiale Ma, Wenzheng Pan, Yang Li, and Junchi Yan. Coexpander: Adaptive solution expansion for combinatorial optimization. In *International Conference on Machine Learning*, 2025.

Yimeng Min, Frederik Wenkel, Michael Perlmutter, and Guy Wolf. Can hybrid geometric scattering networks help solve the maximum clique problem? *Advances in Neural Information Processing Systems (NeurIPS)*, 35:22713–22724, 2022.

Vahab Mirrokni, Renato Paes Leme, Adrian Vladu, and Sam Chiu wai Wong. Tight bounds for approximate Carathéodory and beyond. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70, pp. 2440–2448. PMLR, 2017.

Konrad Mundinger, Max Zimmer, Aldo Kiem, Christoph Spiegel, and Sebastian Pokutta. Neural discovery in mathematics: Do machines dream of colored planes? In *Forty-second International Conference on Machine Learning (ICML)*, 2025.

Kazuo Murota. Discrete convex analysis. *Mathematical Programming*, 83(1):313–371, 1998.

George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Math. Program.*, 14(1):265–294, 1978.

Robert R Nerem, Zhishang Luo, Akbar Rafiey, and Yusu Wang. Differentiable extensions with rounding guarantees for combinatorial optimization over permutations. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Gilles Pisier. Remarques sur un résultat non publié de b. maurey. *Séminaire d'Analyse fonctionnelle (dit" Maurey-Schwartz")*, pp. 1–12, 1981.

Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. Dimes: A differentiable meta solver for combinatorial optimization problems. *Advances in Neural Information Processing Systems (NeurIPS)*, 35: 25531–25546, 2022.

Gerhard Reinelt. Tsplib—a traveling salesman problem library. *ORSA journal on computing*, 3(4): 376–384, 1991.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Approximation ratios of graph neural networks for combinatorial problems. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.

Martin JA Schuetz, J Kyle Brubaker, and Helmut G Katzgraber. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4(4):367–377, 2022.

Richard Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The annals of mathematical statistics*, 35(2):876–879, 1964.

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

Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:3706–3731, 2023.

Zhentao Tan and Yadong Mu. Learning solution-aware transformers for efficiently solving quadratic assignment problem. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pp. 47627–47648, 2024.

Mohit Tawarmalani and Nikolaos V Sahinidis. Convex extensions and envelopes of lower semi-continuous functions. *Mathematical programming*, 93(2):247–263, 2002.

Jan Toenshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Graph neural networks for maximum constraint satisfaction. *Frontiers in artificial intelligence*, 3:580607, 2021.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2692–2700, 2015.

Haoyu Peter Wang, Nan Wu, Hang Yang, Cong Hao, and Pan Li. Unsupervised learning for combinatorial optimization with principled objective relaxation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022a.

Runzhong Wang, Junchi Yan, and Xiaokang Yang. Neural graph matching network: Learning lawler's quadratic assignment problem with extension to hypergraph and multiple-graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):5261–5279, 2021.

Runzhong Wang, Li Shen, Yiting Chen, Xiaokang Yang, Dacheng Tao, and Junchi Yan. Towards one-shot neural combinatorial solvers: Theoretical and empirical notes on the cardinality-constrained case. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2022b.

Runzhong Wang, Yunhao Zhang, Ziao Guo, Tianyi Chen, Xiaokang Yang, and Junchi Yan. Linsatnet: the positive linear satisfiability neural networks. In *International Conference on Machine Learning (ICML)*, pp. 36605–36625. PMLR, 2023.

David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

Yifan Xia, Xianliang Yang, Zichuan Liu, Zhihao Liu, Lei Song, and Jiang Bian. Position: Rethinking post-hoc search-based neural approaches for solving large-scale traveling salesman problems. In *International Conference on Machine Learning (ICML)*, pp. 54178–54190. PMLR, 2024.

Hongtai Zeng, Chao Yang, Yanzhen Zhou, Cheng Yang, and Qinglai Guo. Glinsat: The general linear satisfiability neural network layer by accelerated gradient descent. *Advances in Neural Information Processing Systems (NeurIPS)*, 37:122584–122615, 2024.

Baojian Zhou and Yifan Sun. Approximate Frank–Wolfe algorithms over graph-structured support sets. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pp. 27303–27337. PMLR, 2022.

A Related Work

Extensions and optimization. A long line of work in CO views discrete problems through the lens of continuous optimization by embedding feasible solutions into a convex polytope and optimizing over this relaxation. This perspective is deeply rooted in algorithm design and polyhedral geometry, where constructing convex or concave extensions with favorable optimization properties has been a central theme Lovász (1983); Crama (1993); Murota (1998); Tawarmalani & Sahinidis (2002). In particular, the convex closure provides the tightest convex extension of a discrete set function and underlies many relaxation-based methods Falk & Hoffman (1976); Tawarmalani & Sahinidis (2002). Classical solvers and approximation algorithms routinely exploit this framework: prominent examples include LP and SDP relaxations for problems such as Max Cut and TSP, with solvers like Concorde and Gurobi relying on large-scale LP and cutting-plane techniques Goemans & Williamson (1995); Williamson & Shmoys (2011); Dantzig et al. (1954); Applegate et al. (2011). While these approaches yield strong theoretical guarantees, they typically rely on problem-specific, rounding procedures that are non-differentiable and difficult to integrate into modern, data-driven learning pipelines.

Neural CO. Recent work has explored unsupervised and self-supervised learning approaches for CO that optimize continuous surrogates of discrete objectives, typically induced through probabilistic relaxations and expected-value formulations Amizadeh et al. (2018); Karalias & Loukas (2020); Toenshoff et al. (2021); Wang et al. (2022a). Compared to reinforcement learning or supervised methods Khalil et al. (2017); Kool et al. (2019); Vinyals et al. (2015), these approaches avoid labeled data and often exhibit more stable training. Unsupervised CO pipelines involve several crucial components, including neural network architecture (Min et al., 2022; Vinyals et al., 2015; Sato et al., 2019) and the role of input features for well-known classes of models (Lim et al., 2023; Keriven & Vaiter, 2023). In this work, our focus is on loss function design and rounding. A common strategy in self-supervised CO is to parameterize a distribution over discrete solutions and use the expected value of the discrete objective as a differentiable training loss Karalias & Loukas (2020); Wang et al. (2022a); Bu et al. (2024), enabling smoother optimization but raising challenges related to constraint enforcement. A major consideration in these approaches is enforcing constraints on the outputs of neural networks. Techniques in the literature typically involve projecting the network output onto the feasible set. This has been explored for cardinality constraints using ideas from optimal transport (Wang et al., 2023), and for general linear constraints using gradient-based methods (Donti et al., 2021; Zeng et al., 2024). This also includes Sinkhorn-style projection methods and their extensions (Sinkhorn, 1964; Wang et al., 2022b). In the continuous (and potentially non-convex) optimization setting, several projection techniques have also been developed (Liang et al., 2023; 2024; Liang & Chen; Li et al., 2025). Other approaches add penalty terms or improve training dynamics through annealing or physics-inspired formulations Sun et al. (2022); Schuetz et al. (2022); Mundinger et al. (2025). Our method differs from these approaches by incorporating feasibility directly through geometric decomposition. We define a distribution supported entirely on feasible solutions, allowing constraints to be built into the distribution itself. Thus, the training loss focuses solely on the discrete objective while remaining a.e. differentiable. A closely related line of work develops neural CO methods based on geometric relaxations of feasibility polytopes. Karalias et al. (2022; 2025) propose differentiable extensions for set-function optimization under various constraints, while Nerem et al. (2025) study CO problems over permutations. These approaches typically require the neural network output to lie within the feasibility polytope. In contrast, our approach allows arbitrary

continuous network outputs and applies a general-purpose geometric decomposition across a broad class of combinatorial constraints.

Approximate Carathéodory and Frank-Wolfe. Our approach builds on results from convex geometry and projection-free optimization. A central component is the Frank–Wolfe algorithm Frank et al. (1956); Canon & Cullum (1968), a first-order method for constrained optimization. Several variants with improved convergence have been studied Jaggi (2013); Lacoste-Julien & Jaggi (2015), and Frank–Wolfe has been widely applied to structured and combinatorial polytopes Garber & Meshi (2016); Lacoste-Julien et al. (2013); Krishnan et al. (2015). Approximate variants of Frank–Wolfe with provable guarantees have also been proposed Locatello et al. (2017); Zhou & Sun (2022). From a geometric perspective, the Carathéodory theorem Carathéodory (1907) states that any point in a polytope $P \subseteq \mathbb{R}^d$ can be expressed as a convex combination of at most $d+1$ corners of the polytope. Approximate versions provide dimension-independent bounds on sparse approximations Barman (2015); Mirrokni et al. (2017). Obtaining such sparse convex combinations of polytope vertices is essential for our framework, and Frank–Wolfe-type algorithms provide a natural algorithmic mechanism to achieve this Mirrokni et al. (2017); Combettes & Pokutta (2023).

B Proofs for Theorem 3.3 and Corollary 3.4

Proof of Theorem 3.3. The convex-combination structure follows directly from the update $\mathbf{u}_t = (1-\gamma_t)\mathbf{u}_{t-1} + \gamma_t \mathbf{v}_t$ with $\gamma_t \in [0,1]$ and the initialization $\mathbf{u}_0 = \mathbf{v}_0 \in \mathrm{ext}(\mathcal{P})$. Inductively,

$$\mathbf{u}_{T-1} = \sum_{t=0}^{T-1} \alpha_t \mathbf{v}_t, \qquad \mathbf{v}_t \in \mathrm{ext}(\mathcal{P}), \qquad \alpha_t \geq 0, \qquad \sum_{t=0}^{T-1} \alpha_t = 1.$$

For the convergence rate, let $D = \mathrm{diam}(\mathcal{P})$ and $J(\mathbf{u}) = \frac{1}{2}\|\mathbf{x} - \mathbf{u}\|_2^2$. The gradient of J is 1-Lipschitz, so the Frank–Wolfe curvature of J over $\mathcal{P}$ is bounded by D^2. The standard Frank–Wolfe rate with exact linear minimization/maximization oracles and line search gives, after k updates,

$$J(\mathbf{u}_k) - J(\mathbf{u}^\star) \leq \frac{2D^2}{k+2}, \qquad \mathbf{u}^\star \in \arg\min_{\mathbf{u}\in\mathcal{P}} J(\mathbf{u}).$$

Setting $k = T-1$ yields

$$J(\mathbf{u}_{T-1}) - J(\mathbf{u}^\star) \leq \frac{2D^2}{T+1}.$$

It remains to justify the almost-everywhere differentiability claim under the stable oracle decision assumption. For almost every $\mathbf{x}$, the assumption gives a neighborhood on which all selected vertices and line-search clipping regimes are fixed. Fix such an input and neighborhood. The initialization $\mathbf{u}_0 = \mathbf{v}_0$ is constant on this neighborhood. Recursively, if $\mathbf{u}_{t-1}(\mathbf{x})$ is smooth there, then

$$\mathbf{r}_t(\mathbf{x}) = \mathbf{x} - \mathbf{u}_{t-1}(\mathbf{x}), \qquad \mathbf{d}_t(\mathbf{x}) = \mathbf{v}_t - \mathbf{u}_{t-1}(\mathbf{x})$$

are smooth because $\mathbf{v}_t$ is fixed. When $\mathbf{d}_t(\mathbf{x}) \neq 0$, the unclipped line-search value

$$\tilde{\gamma}_t(\mathbf{x}) = \frac{\langle \mathbf{r}_t(\mathbf{x}), \mathbf{d}_t(\mathbf{x})\rangle}{\|\mathbf{d}_t(\mathbf{x})\|_2^2}$$

is a smooth algebraic function on a possibly smaller neighborhood with nonzero denominator. When $\mathbf{d}_t = 0$, we use the algorithmic convention $\gamma_t = 0$, and the update is locally constant in that direction. Since the clipping regime is fixed on the neighborhood, γ_t is either 0, 1, or $\tilde{\gamma}_t$, and is therefore differentiable there. Hence

$$\mathbf{u}_t = (1-\gamma_t)\mathbf{u}_{t-1} + \gamma_t \mathbf{v}_t$$

involves only differentiable algebraic operations. The same recursive update shows that each coefficient $\alpha_t(\mathbf{x})$ is differentiable on the neighborhood. The only excluded inputs lie in the zero-volume switching set from the stable oracle decision assumption, so the coefficient map $\mathbf{x} \mapsto \boldsymbol{\alpha}(\mathbf{x})$ is differentiable almost everywhere. □

Proof of Corollary 3.4. The convex-combination conclusion is unchanged because the update still uses $\gamma_t \in [0, 1]$ and vertices of $\mathcal{P}$. The relative FW-gap condition

$$\langle \mathbf{r}_t, \mathbf{v}_t - \mathbf{u}_{t-1} \rangle \geq \delta \max_{\mathbf{v} \in \mathcal{P}} \langle \mathbf{r}_t, \mathbf{v} - \mathbf{u}_{t-1} \rangle$$

is the maximization form of the standard approximate Frank–Wolfe oracle condition. Applying the approximate Frank–Wolfe theorem of Locatello et al. (2017) to $J(\mathbf{u}) = \frac{1}{2}\|\mathbf{x} - \mathbf{u}\|_2^2$, with Lipschitz constant 1 and diameter D, gives after $T - 1$ updates

$$J(\mathbf{u}_{T-1}) - J(\mathbf{u}^\star) \leq \frac{2\left(D^2/\delta + h_0\right)}{\delta(T-1) + 2}, \qquad h_0 = J(\mathbf{u}_0) - J(\mathbf{u}^\star).$$

The almost-everywhere differentiability statement follows from the same stable oracle decision argument used above. □

C Details and Further Experiments for Maximum Coverage

Hardware. All experiments, including training and inference on Maximum Coverage, TSP and QAP, are done on 16 cores (32 threads) of Intel(R) Xeon(R) Platinum 8268 CPU (24 cores, 48 threads in total), 32 GB DDR4 ram, with a single Nvidia RTX8000 48GB GPU.

C.1 Experiment Setup

Datasets. Following prior work (Bu et al., 2024; Wang et al., 2022b; Karalias et al., 2025), we evaluate on synthetic and real-world bipartite graphs (U, V, E), where V is the ground set and the task is to select k nodes from V to maximize the total weight of covered nodes in U. For synthetic data, we use the Random Uniform dataset under two scales, `Random500` and `Random1000`. Each setting contains a training dataset of 100 independently generated instances with $|V| = 500, |U| = 1000$ (`Random500`) or $|V| = 1000, |U| = 2000$ (`Random1000`). Each $u \in U$ is assigned an integer weight sampled uniformly from $\{1, \ldots, 100\}$, and each $v \in V$ covers a random subset of U whose cardinality is sampled uniformly from $\{10, \ldots, 30\}$. Testing are conducted on datasets sampled from the same distribution. For real-world testing, we additionally evaluate on three *Railway* instances derived from Italian railway crew assignment graphs: `rail507` ($|V| = 507, |U| = 63009$), `rail516` ($|V| = 516, |U| = 47311$), and `rail582` ($|V| = 582, |U| = 55515$).

Training. In general, we use the same training setup as Karalias et al. (2025), using a 3-layer GNN with residual connection. To enhance optimization stability and solution quality, we incorporate specific regularization strategies to promote exploration within the hypersimplex. An entropy regularization term is applied with a cosine-decayed weight λ_t, which is initialized at 0.05 and annealed to zero by the 30th epoch. A dropout rate of 0.1 is also applied exclusively during the training phase.

The model is trained for a total of 80 epochs using the AdamW optimizer with a weight decay of 1×10^{-4}. We employ a batch size of 4 and a fixed random seed of 42 to ensure reproducibility. The learning rate follows a `warmup_cosine` scheduler, which includes 50 warmup epochs and decays from a peak of 5×10^{-3} to a minimum of 5×10^{-5}. Data loading and preprocessing are handled efficiently by 16 CPU worker threads.

Inference. In inference, we run our FW Decomposition for 50 iterations for each instance, and select the best result out of the decomposed sets. We do not use any randomized or local improvment method.

C.2 Baselines

We compare against three classes of baselines that span exact solvers, classical heuristics, and neural approaches.

(i) Exact solvers (non-learning). We employ **Gurobi** to obtain exact MIP-based solutions, imposing a time limit of 120 seconds for each graph.

(ii) Classical heuristics (non-learning). We include a **Random** baseline, which samples subsets of size k uniformly at random over multiple trials and selects the best solution found within 240 seconds. We also report the **Greedy algorithm** (Nemhauser et al., 1978), which iteratively adds the element with the largest marginal gain to achieve the well-known $(1 - \frac{1}{e})$-approximation guarantee. Additionally, we classify **UCOM2** (Bu et al., 2024) as a non-learning baseline due to its use of incremental greedy de-randomization, and we report results for its short variant.

(iii) Neural baselines. We evaluate a diverse set of neural methods. We include **CardNN** (Wang et al., 2022b) variants (without Test-Time Optimization, following Bu et al. (2024)) and a naive version of **EGN** (Karalias & Loukas, 2020) with naive probabilistic objective construction and iterative rounding. We also compare against **GeoNCO** (Karalias et al., 2025), which employs the same Carathéodory decomposition framework as our method but lacks our proposed enhancements. Finally, we include a Reinforcement Learning (**RL**) baseline following (Karalias et al., 2025) trained with the Actor-Critic (Konda & Tsitsiklis, 1999) algorithm on the same problem instances.

C.3 Ablation study on projection

We ablate the effect of explicitly projecting the network output back into the feasible solution polytope before running FW DECOMPOSITION.We use the same projection method as Karalias et al. (2025), while they rely on such a projection to enforce feasibility, our framework does not require it since feasibility is already guaranteed by the decomposition.

Table 4 shows that removing projection preserves solution quality and in fact yields slightly better coverage across both `rand500` and `rand1000` settings, with slightly lower wall-clock inference time. We attribute this to the fact that the projection is a hard, piecewise operator that can compress the effective search region and reduce gradient signal diversity; in our pipeline, it becomes largely redundant because feasibility is enforced by the decomposition itself. By contrast, allowing unconstrained continuous scores retains a richer exploration space for the model, while FW DECOMPOSITION still produces feasible candidates, leading to steadier optimization and marginally improved performance in practice.

Table 4: Ablation study results. Grouped by dataset size.

Method	rand500		rand1000	
	$k = 10$	$k = 50$	$k = 20$	$k = 100$
With Projection	15190.10 0.4635s	42138.45 0.5513s	29906.24 0.6423s	83474.98 0.9940s
No Projection	15192.64 0.4261s	42172.44 0.5378s	29926.33 0.6472s	83559.22 0.9135s

C.4 Ablation on Hard Instances for Greedy

Hard example for greedy. The greedy algorithm performs well, especially on random graphs, for the MC problem because the coverage function is monotone and submodular. To show our method works even when greedy fails, we construct the GREEDYTRAP500 and GREEDYTRAP1000 datasets for MC. In this construction, the greedy algorithm is pushed near its theoretical $1 - 1/e = 0.63212$ lower bound and attracted by sets with large initial marginal coverage, falling significantly short of the optimal solution.

The construction is as follows. We first choose a coverable subset of elements from $\mathcal{U}$ and partition it into k disjoint parts. For each part, we add one set that covers the entire part; selecting these k sets covers all coverable elements. We then add k greedy-trap sets. Each trap set covers slightly more than a $1/k$ fraction of every part, so these sets have large marginal gain at the beginning and are selected by greedy. The remaining sets are low-degree filler sets. For GREEDYTRAP500, we use 500 sets and 1000 elements; for GREEDYTRAP1000, we use 1000 sets and 2000 elements. In testing, we report the coverage ratio over all elements.

As shown in Table 5, greedy falls significantly short of OPT on these hard instances, whereas our method consistently outperforms greedy and achieves coverage much closer to OPT. This shows

Table 5: Performance comparison on the GREEDYTRAP datasets. Values represent coverage ratios in percentage averaged across 64 test instances.

	GreedyTrap500		**GreedyTrap1000**	
Method	$k = 10$	$k = 50$	$k = 20$	$k = 100$
Greedy	46.30%	66.50%	45.80%	70.60%
FWNCO(ours)	59.90%	78.40%	55.60%	82.30%
OPT	60.10%	81.90%	58.60%	86.70%

that our method can still learn useful problem structure and maintain strong performance on data distributions where the standard greedy heuristic fails.

C.5 EFFECT OF THE FRANK–WOLFE ITERATION BUDGET

We investigate the effect of the number of Frank–Wolfe decomposition iterations T on solution quality and runtime for Maximum Coverage. Each iteration of Algorithm 1 performs one LMO call and a closed-form line-search step. Note that the runtime is linear in T.

Table 6 reports results for four configurations, with solution quality normalized by the weighted coverage obtained at $T = 200$. Increasing T consistently improves quality, with diminishing gains at larger budgets. At $T = 50$, the method already achieves 99.391–99.652% of the reference quality across all four configurations. Increasing the budget to $T = 200$ improves relative quality by only 0.348–0.609 percentage points, while increasing the reported runtime by approximately 3.7–3.8×. These results support our choice of $T = 50$ as a practical balance between solution quality and computational cost.

Table 6: Maximum coverage quality and inference time under different FW iteration budgets. Quality is normalized by the weighted coverage obtained at $T = 200$ and reported as a percentage. Time recorded on an NVIDIA GeForce RTX 4090 GPU and not directly comparable with other results.

	n=500, k=10		n=500, k=50		n=1000, k=20		n=1000, k=100	
T	Quality (%)	Time (s)	Quality (%)	Time (s)	Quality (%)	Time (s)	Quality (%)	Time (s)
10	98.197	0.0802	99.230	0.1153	98.712	0.1418	99.464	0.1965
30	98.942	0.2046	99.376	0.2853	99.296	0.3556	99.548	0.4920
50	**99.391**	**0.3295**	**99.494**	**0.4554**	**99.501**	**0.5669**	**99.652**	**0.7852**
70	99.470	0.4516	99.630	0.6246	99.601	0.7784	99.755	1.0770
100	99.659	0.6333	99.771	0.8775	99.695	1.0889	99.861	1.5150
150	99.859	0.9345	99.909	1.2994	99.848	1.5979	99.930	2.2447
200	**100.000**	**1.2365**	**100.000**	**1.7197**	**100.000**	**2.1068**	**100.000**	**2.9653**

D DETAILS AND FURTHER EXPERIMENTS ON QAP

In this section, we provide a detailed discussion of the numerical results for QAP. The full experimental results are reported in Table 7.

D.1 EXPERIMENT SETUP

Instance distribution and data splits. For synthetic training, we follow the data generation method of Tan & Mu (2024) and generate a synthetic training pool with 5120 instances, $n = 32$ and $p = 0.3$. During training, each gradient step samples a mini-batch uniformly from the training pool.

For test, we use the QAPLIB dataset to demostrate the generalization ability of our method. QAPLIB (Burkard et al., 1997) is a widely used benchmark for QAP, collecting real-world and synthetic instances from multiple sources. Each instance is specified by a flow matrix and a distance

Table 7: Complete experiment results on QAPLIB. The mean/max/min gaps are reported for each category. The mean performance over all categories and the inference time (s) per instance are reported.

	bur (26)			chr(12-25)			esc (16-64)			had (12 - 20)		
	Mean↓	Min↓	Max↓	Mean↓	Min↓	Max↓	Mean↓	Min↓	Max↓	Mean↓	Min↓	Max↓
SM	22.3	20.3	24.9	460.1	144.6	869.1	301.6	**0.0**	3300.0	17.4	14.7	21.5
RRWM	23.1	19.3	27.3	616.0	120.5	1346.3	63.9	**0.0**	200.0	25.1	22.1	28.3
SK-JA	4.7	2.8	6.2	**38.5**	**0.0**	**186.1**	364.8	**0.0**	2200.0	25.8	6.9	100.0
NGM	3.4	2.8	4.4	121.3	45.4	251.9	126.7	**0.0**	200.0	8.2	6.0	11.6
RGM	7.1	4.5	9.0	112.4	23.4	361.4	32.8	**0.0**	141.5	6.2	1.9	9.0
SAWT	2.8	2.2	3.4	110.7	8.6	201.2	**13.5**	**0.0**	**63.7**	3.8	1.6	6.5
FWNCO (ours)	**2.3**	**1.9**	**2.6**	97.7	28.8	209.9	19.1	**0.0**	119.1	**2.8**	**1.2**	**3.7**

	kra (30-32)			lipa (20-60)			nug (12-30)			rou (12-30)		
	Mean↓	Min↓	Max↓	Mean↓	Min↓	Max↓	Mean↓	Min↓	Max↓	Mean↓	Min↓	Max↓
SM	65.3	63.8	67.3	19.0	3.8	34.8	45.5	34.2	64.0	35.8	30.9	38.2
RRWM	58.8	53.9	67.7	20.9	3.6	41.2	67.8	52.6	79.6	51.2	39.3	60.1
SK-JA	41.4	38.9	44.4	**0.0**	**0.0**	**0.0**	25.3	10.9	100.0	13.7	10.3	17.4
NGM	31.6	28.7	36.8	16.2	3.6	29.4	21.0	14.0	28.5	30.9	23.7	36.3
RGM	**15.0**	**10.4**	**20.6**	13.3	3.0	23.8	9.7	6.1	**12.9**	13.4	**7.1**	16.7
SAWT	30.1	28.1	34.2	0.4	**0.0**	2.6	**9.2**	**4.1**	13.1	**10.8**	8.2	13.1
FWNCO (ours)	37.0	34.8	39.2	13.5	2.2	26.3	11.5	7.3	14.7	10.9	9.7	**12.1**

	scr (12-20)			sko(42-64)			ste(36)			tai(12-64)		
	Mean↓	Min↓	Max↓	Mean↓	Min↓	Max↓	Mean↓	Min↓	Max↓	Mean↓	Min↓	Max↓
SM	123.4	104.0	139.1	29.0	26.6	31.4	475.5	197.7	1013.6	180.5	21.6	1257.9
RRWM	173.5	98.9	218.6	48.5	47.7	49.3	539.4	249.5	1117.8	197.2	26.8	1256.7
SK-JA	48.6	44.3	55.7	18.3	16.1	20.5	120.4	72.5	200.4	25.2	1.6	107.1
NGM	55.5	41.4	66.2	25.2	22.8	27.7	101.7	57.6	172.8	61.4	18.7	352.1
RGM	45.5	30.2	56.1	**10.6**	**9.9**	**11.2**	134.1	69.9	237.0	17.3	11.4	**28.6**
SAWT	28.5	**10.2**	48.0	17.7	16.7	18.7	93.5	**46.7**	170.2	**16.5**	**0.5**	48.7
FWNCO (ours)	**23.6**	13.6	**39.8**	15.0	13.8	15.6	**83.5**	58.5	**130.2**	20.7	1.7	43.0

	tho (30-40)			wil(50)			**Average(12-64)**			Time per instance (in seconds)
	Mean↓	Min↓	Max↓	Mean↓	Min↓	Max↓	Mean↓	Min↓	Max↓	
SM	55.0	54.0	56.0	13.8	11.7	15.9	181.2	46.9	949.9	**0.01**
RRWM	80.6	78.2	83.0	18.2	12.5	23.8	169.5	49.5	432.9	0.15
SK-JA	32.9	30.6	35.3	8.8	**6.7**	10.7	93.2	**9.0**	497.9	563.4
NGM	27.5	24.8	30.2	10.8	8.2	11.1	62.4	17.8	129.7	15.72
RGM	**20.7**	**12.7**	28.6	8.1	7.9	**8.4**	35.8	10.7	101.1	75.53
SAWT	24.8	23.2	26.4	**8.1**	7.6	8.6	26.8	11.2	**47.0**	12.11
FWNCO (ours)	23.0	20.7	**25.3**	9.2	9.2	9.2	**26.4**	14.5	49.3	1.56

matrix, and the naming convention uses a class prefix (e.g., `bur`, `chr`, `tai`) together with the problem size. The distributions and structural properties of the flow/distance matrices vary substantially across classes, making QAPLIB a challenging testbed for cross-class generalization.

Input representation. Given A and B, we build a node embedding for each matrix separately using per-node statistics (diagonal entry and row/column aggregations) together with random-projection features computed with a fixed Gaussian projection matrix R. Features are computed after per-instance magnitude normalization (i.e., dividing by $\max_{i,j} |M_{ij}|$) to stabilize training across varying scales.

Model, optimizer, and training schedule. We use a two-layer GraphSAGE aggregator with a residual skip connection and hidden size `model_dim`=128, applied independently to A and B. The neural network outputs a matrix followed by a Sinkhorn projection layer. We train with AdamW (learning rate $3 \cdot 10^{-4}$, weight decay 10^{-4}) and gradient clipping at norm 1.0. Training runs for 100 epochs; each epoch consists of 100 instances randomly sampled from the training pool. A single global RNG seed (0) is used for both training and inference.

Sinkhorn as a practical projection layer. Although our FW-based pipeline does *not* theoretically require the network output to lie inside the Birkhoff polytope (the FW DECOMPOSITION can be applied to any real matrix), in QAP this relaxation is empirically fragile because QAP instances are substantially more heterogeneous than the other tasks: QAPLIB in particular exhibits highly dispersed scales, sparsity patterns, and instance-generation mechanisms across data classes, leading to large distribution gaps relative to synthetic pretraining. In this regime, leaving the model output unconstrained often yields score matrices whose magnitude and row/column mass distribution are

poorly calibrated, which in turn destabilizes the downstream greedy steps and makes the distribution returned by FW DECOMPOSITION less informative.

To mitigate this, we insert a Sinkhorn Sinkhorn (1964) projection layer with 20 iterations as a soft projection that maps raw assignment logits to an approximately doubly-stochastic matrix. This operation enforces nonnegativity and approximately unit row/column sums, placing $\mathbf{X}$, the output of neural network, closer to the Birkhoff polytope. It serves as a lightweight output-range calibration mechanism that significantly improves numerical stability under the large inter-class distribution shift in QAPLIB. We emphasize that this Sinkhorn step is used for practical robustness, not as a theoretical requirement of the FW DECOMPOSITION itself.

Decomposition settings. We run our FW DECOMPOSITION algorithm over permutation matrices with a iteration budget $T = 2000$ in training and $T = 3000$ in inference.

At each iteration, the LMO is instantiated by a *greedy* maximum-weight assignment routine (no Hungarian), applied to the current residual matrix. The resulting decomposition yields a sequence of permutations and their weights; in inference, we select the minimum-cost permutation under the QAP objective for reporting.

Table 8: QAPLIB results for instances with $n > 64$.

Instance	pred_best	best_known	gap
lipa70a	172973	169755	1.90%
lipa70b	5872840	4603200	27.58%
sko72	75952	66256	14.63%
lipa80a	257769	253195	1.81%
lipa80b	9983972	7763962	28.59%
tai80b	1057836224	818415043	29.25%
tai80a	15311664	13499184	13.43%
sko81	104214	90998	14.52%
lipa90a	366329	360630	1.58%
lipa90b	16017197	12490441	28.24%
sko90	132078	115534	14.32%
sko100a	174044	152002	14.50%
sko100b	174378	153890	13.31%
sko100c	170276	147862	15.16%
sko100d	170574	149576	14.04%
sko100e	170746	149150	14.48%
sko100f	169558	149036	13.77%
tai100a	23638054	21052466	12.28%
tai100b	1498196608	1185996137	26.32%
wil100	296136	273038	8.46%
esc128	244	64	281.25%
tai150b	619556288	498896643	24.19%
tho150	9579958	8133398	17.79%
tai256c	50753096	44759294	13.39%
		Average gap	26.87%

QAPLIB evaluation. Following previous approaches Tan & Mu (2024); Liu et al. (2023), We report per-class gap statistics and the aggregate summary as in Table 7 for instances of size 12-64 since many baseline methods will run out of time on very large instances. However, our method is also capable of solving instances with larger sizes, as listed below in Table 8.

D.2 RELATED WORK (QAP / MATCHING).

QAP and matching are among the most challenging CO problems for learning-based methods, largely due to severe distribution shifts and outliers such as those in QAPLIB. **NGM** Wang et al.

(2021) was one of the earliest attempts to tackle QAP/graph matching with neural models. Subsequent work proposed **RGM** Liu et al. (2023), which introduced a *learn-to-construct (L2C)* pipeline for QAP, building a permutation by making a sequence of constructive decisions, following that, **SAWT** Tan & Mu (2024) introduced a *learn-to-improve (L2I)* pipeline, which starts from an initial permutation and repeatedly applies local modifications to improve it.

Both L2C and L2I are inherently iterative: they start from an empty/partial solution or an initial solution and improve it step by step. This makes inference naturally less efficient and less friendly to large instances, and robustness to outliers is achieved through relatively complex neural architectures and nontrivial inference procedures.

Our general framework applies to QAP in a single-pass direct-output manner: we produce assignment logits once and deterministically output a permutation without repeated optimization loops, yielding an immediate speed advantage. Moreover, with only a lightweight Sinkhorn normalization layer added to the base framework, we obtain outlier-friendly behavior at very low additional cost.

D.3 Baselines

We compare against two classes of baselines that cover both classical matching solvers and representative neural methods on QAPLIB.

(i) Classical solvers (non-learning). We include **SM**, **RRWM**, and **SK-JA** Kushinsky et al. (2019) as standard non-learning baselines for graph matching / QAP.

(ii) Neural methods. We evaluate **NGM**, **RGM**, and **SAWT** as representative learning-based baselines on QAPLIB.

We follow the same per-class reporting protocol as prior QAPLIB evaluations, and report the mean/min/max optimality gaps for each QAPLIB class together with the average per-instance inference time. As the public releases of both **SAWT** and **RGM** do not include a complete evaluation setup and code for QAPLIB, we are unable to reproduce their experiments locally. Instead, we directly cite their reported timing results. Consequently, due to hardware disparities, these runtime comparisons should be interpreted as indicative references rather than strictly controlled benchmarks. We also note an additional caveat in their QAPLIB results: for the WIL class they report multiple results, whereas within range ($n \in [12, 64]$) where they reported, WIL contains only a single instance (WIL50). Since we cannot verify the exact evaluation protocol (e.g., whether additional variants or sizes were included), we therefore quote their reported numbers in full without further reinterpretation.

D.4 Effect of the Frank–Wolfe Iteration Budget

We evaluate the sensitivity of QAP solution quality and runtime to the Frank–Wolfe decomposition budget (T) on QAPLIB. Table 9 reports relative quality normalized by the weighted solution obtained at $T = 10000$, together with runtime for each budget.

Solution quality improves substantially as the budget increases from $T = 50$ to $T = 1000$, rising from 94.247% to 99.237% of the reference quality. Further iterations yield progressively smaller improvements. At our chosen budget of $T = 3000$, relative quality reaches 99.837%, with a runtime of 4.410 seconds. Increasing the budget to $T = 10000$ adds only 0.163 percentage points of relative quality while increasing runtime to 15.798 seconds, approximately $3.6\times$ higher. We therefore use $T = 3000$ to balance solution quality and runtime.

Compared with Maximum Coverage, QAP empirically benefits from a substantially larger decomposition budget. The approximation guarantee in Theorem 3.3 provides qualitative context: the squared approximation error is bounded by $4D^2/(T + 1)$, where D is the diameter of the feasible polytope. Our choice of T is guided by this and the observed quality–runtime trade-off.

Table 9: QAP solution quality under different FW iteration budgets on QAPLIB. Results are normalized by the weighted solution obtained at T = 10000. Time recorded on an NVIDIA GeForce RTX 4090 GPU and not directly comparable with other results.

FW budget T	Relative quality (%)	Time (s)
50	94.247	0.054
100	95.632	0.105
200	96.825	0.216
500	98.595	0.594
1000	99.237	1.304
2000	99.672	2.828
3000	**99.837**	**4.410**
4000	99.893	6.017
5000	99.898	7.639
7500	99.921	11.707
10000	100.000	15.798

E Details and Further Experiments on TSP

E.1 Experiment Setup

Instance distribution and data splits. We train and evaluate on the 2D Euclidean TSP, where each instance consists of n points in $[0, 1]^2$ and tour length is computed under standard Euclidean distances. We use the training and evaluation data from **COExpander** Ma et al. (2025). During training, for a given n, at each epoch, 64 instances are sampled from the training pool, and different data are used across epochs. The test split has 1280 instances for TSP-50 and TSP-100, 128 instances for TSP-500 and 32 instances for TSP-1000, so that most methods could complete the evaluation in reasonable time frame.

Graph construction and input features. For each instance we construct the complete graph on n nodes (undirected edges enumerated with $u < v$), and represent it in PyG using the standard bidirected encoding (each undirected edge appears in both directions in `edge_index`). Node features are the raw coordinates augmented with a Fourier positional encoding (with F=4 frequencies) passed through a small MLP before message passing. Edge features consist of a single scalar channel: the LP relaxation value $x_{\text{LP}}(e)$ for the undirected edge, duplicated across both directed copies.

LP relaxation feature computation. We compute x_{LP} by solving the subtour-elimination LP using the QSopt LP solver, which is also the LP solver used by Concorde. Edge costs are obtained from Euclidean distances and scaled to integer weights using a fixed coordinate scale of 10^6 for the QSopt interface. LP solves are performed on CPU for every instance in both train and test splits.

Model, optimizer, and training schedule. The encoder is a GATv2 model with residual blocks: hidden size 128, 3 message-passing layers, and 4 attention heads per layer. We train with AdamW with a learning rate of 10^{-3} and a weight decay of 10^{-4} for 50 epochs, using a single global RNG seed (0) for reproducibility. The best model is saved and used for inference.

Decomposition hyperparameters used in training and inference. We use a fixed decomposition budget of $T = 64$ iterations in both training and testing, and a max-entropy regularization with temperature `fw_entropy_tau`$= 2.0$ is added to the decomposition. All reported TSP test results use the same hyperparameters as training and the reported result is the best permutation from the decomposition.

E.2 Related Work: Neural Methods for TSP

Learning paradigms: LC vs. GP. Neural approaches for TSP are often categorized by the granularity at which they make decisions. According to Ma et al. (2025), the *Learning-to-Construct (LC)* paradigm builds a tour sequentially, repeatedly selecting the next decision conditioned on the current

partial solution. A key strength of LC is that feasibility can be *hard-enforced* during decoding (e.g., by imposed masks that rule out invalid moves), which provides a transparent and reliable path to valid tours. However, this same sequential structure typically incurs high inference latency on large instances, since the model must execute many dependent steps.

In contrast, the *Global Prediction (GP)* paradigm predicts a globally complete structure in one (or a few) forward passes, commonly as an edge-probability heatmap, and then applies a separate recovery procedure to obtain an integer-feasible tour. GP can be substantially faster at inference and has become a dominant template for large-scale settings, including diffusion- and consistency-based solvers that generate global solutions through iterative denoising. At the same time, GP shifts difficulty into the heatmap-to-solution stage: the final performance can depend heavily on the specific decoding/recovery algorithm (e.g., Parallel Sampling, Greedy, Monte Carlo Tree Search (MCTS)) and additional improvement heuristics (e.g., 2-opt), making the pipeline sensitive to post-processing choices and sometimes blurring what is being evaluated (the neural predictor vs. the downstream solver), since heatmap recovery is itself a combinatorial optimization subroutine.

Adaptive Expansion (AE) and the remaining heatmap bottleneck. **COExpander** introduces *Adaptive Expansion (AE)* as an intermediate paradigm intended to combine the best of both worlds: it uses a global predictor to produce a heatmap under partial-solution prompting, and then expands the determined set via a determination/decoding operator, effectively adapting the decision granularity across iterations. While AE improves the efficiency–feasibility trade-off relative to pure LC or pure GP, it still inherits a core limitation of GP-style pipelines: it relies on predicting a heatmap and then recovering an integer tour via post-processing. So overall success remains coupled to the robustness and tuning of heatmap-based recovery.

Toward end-to-end pipelines with reduced problem-specific setup. Xia et al. (2024) critically examine the prevalent *heatmap-guided post-hoc search* paradigm for large-scale TSP (notably heatmap-guided MCTS), questioning the practical value of learning heatmaps that ultimately require substantial downstream search and advocating a shift toward more *autonomous and generalizable* ML pipelines with less hand-crafted machinery and stronger guarantees. Motivated by this perspective, our method aims to retain LC's feasibility discipline and GP's efficiency while avoiding the heatmap bottleneck: we adopt an end-to-end pipeline that differentiates through the selected FW branch, with the *optimization objective aligned with the prediction target*—we do not learn a surrogate heatmap and then rely on a separate recovery/search step, but instead directly optimize for improved feasible tours through a structured, deterministic procedure. This design reduces reliance on hand-crafted post-processing, improves robustness across instance distributions, and enables stronger theoretical characterization of the resulting algorithmic guarantees.

E.3 BASELINES

We compare against four classes of baselines that span both classical optimization and representative neural paradigms under different supervision regimes.

(i) Exact solvers (non-learning). We report **Concorde** as an exact TSP solver, and the **Gurobi** optimizer as an exact baseline on small instances. For larger instances, exact results are omitted when the solver fails to finish within the runtime cutoff.

(ii) Classical heuristics (non-learning). We include **LKH3** Helsgaun (2017) as a strong hand-engineered heuristic, run for 500 trials.

(iii) Neural LC baselines. We evaluate **RL4CO (SymNCO)** Berto et al. (2025) as a representative *Learning-to-Construct (LC)* method trained with reinforcement learning. LC-style solvers typically decode solutions sequentially and rely on imposed masks (or equivalent constraints) to hard-guarantee feasibility during construction.

(iv) Neural GP/AE baselines. We test heatmap-driven pipelines in both RL-style and supervised regimes. **DIMES** Qiu et al. (2022) is included as a representative GP-style method trained via meta-RL. On the supervised side, we include **DIFUSCO** Sun & Yang (2023), a diffusion-based GP approach, and **COExpander**, a heatmap-based *Adaptive Expansion (AE)* method that achieves state-of-the-art performance.

Table 10: Raw performance comparison on TSP instances. We report the objective value (Obj.) and inference time (s).

Method	Type	TSP-50		TSP-100		TSP-500		TSP-1000	
		Obj.	Time	Obj.	Time	Obj.	Time	Obj.	Time
Concorde (Optimal)	Exact	5.69	0.04	7.76	0.21	16.55	17.91	23.12	435.61
Gurobi (Optimal)*	Exact	5.69	0.27	7.76	1.14	-	-	-	-
LKH3 (500)	Heuristics	5.69	0.03	7.76	0.05	16.70	0.32	23.41	1.06
DIMES (RL+S)	RL	6.47	0.05	8.66	0.08	19.09	0.39	26.35	0.79
RL4CO (Sym-NCO)†	RL	5.76	0.36	8.83	0.66	-	-	-	-
DIFUSCO (S=1, I=50)	SL	5.72	0.43	7.86	0.66	18.17	2.19	25.64	7.68
COExpander (S=1,Ds=3,Is=5)	SL	5.69	0.11	7.76	0.20	17.17	0.69	24.63	2.50
Ours-lp	UL	5.79	0.23	8.03	0.26	17.76	1.27	25.03	4.26
Ours-lp+learned matching	UL	5.77	0.52	7.98	0.74	17.70	2.77	24.84	9.16

* Results on larger instances are omitted since it was unable to finish within the cutoff time.
† Results are omitted due to prohibitive computational costs during training on large datasets.

Overall, under a fixed compute budget, we aim to test the landscape as comprehensively as possible across (a) paradigms (LC vs. GP vs. AE) and (b) supervision regimes (RL / meta-RL vs. supervised learning), while transparently reporting any omissions caused by solver timeouts or training-cost constraints.

E.4 Detailed Results

Runtime breakdown. In Table 11, we measure the average per-instance wall-clock cost of each stage in the TSP pipeline . The Subtour-Elimination LP and FW DECOMPOSITION dominates the runtime; in the learned matching version, the matching takes up a large portion of the overall runtime.

Stage	**Time (s/inst)**
LP (Subtour-Elimination)	3.09
Neural Network	0.06
FW over trees	0.93
Greedy matching	0.18
Learned Matching	4.23

Table 11: Average per-instance time on TSP1000.

Generalization Experiments (Across Instance Sizes). We further evaluate how well a model trained on one problem size transfers to other sizes. Specifically, we train three separate models on TSP-100, TSP-500, and TSP-1000, respectively, and test each model on the same evaluation splits at sizes $\{100, 500, 1000\}$. Table 12 reports the resulting tour lengths.

Overall, we observe strong cross-size generalization: performance is largely stable when transferring between sizes, with only modest degradation when training on smaller instances and testing on larger ones. For example, a model trained on TSP-100 remains competitive on TSP-500/1000, albeit with a small gap relative to models trained on the target size. Conversely, models trained on larger instances transfer well to smaller instances, with only minor differences on TSP-100. These results suggest that the learned scoring function captures size-agnostic geometric regularities of Euclidean TSP, enabling effective transfer across instance scales without retraining on each target size.

Table 12: Generalization performance across different instance sizes.

Training Set	Testing Set		
	TSP-100	TSP-500	TSP-1000
TSP-100	8.03	17.85	25.28
TSP-500	8.09	17.76	25.06
TSP-1000	8.16	17.81	25.03

E.5 Discussion on Learned Matching

Motivation and positioning. In our TSP pipeline, each candidate spanning tree produced by the FW decomposition is rounded into a tour via the Christofides procedure. A key step in Christofides is computing a perfect matching on the odd-degree vertex set induced by the tree. Our default implementation uses a greedy matching routine for efficiency, while the learned-matching variant replaces this step with a trainable module that is *optimized end-to-end for the TSP objective*, rather than supervised to recover an exact minimum-weight perfect matching.

Matching as another structured polytope component. For a given tree, let O be its odd-degree vertex set (with $|O|$ even), and let E_O denote all unordered pairs in O, which forms a structured polytope. The matching network outputs a vector $S \in (0,1)^{|E_O|}$ over E_O. We then run our FW DECOMPOSITION procedure on the matching polytope over O to obtain a short convex decomposition $A_{\text{match}} \approx \sum_k \beta_k M_k$, where each M_k is an *integral* perfect matching (an extreme point of the polytope), $\beta_k \geq 0$, and $\sum_k \beta_k = 1$.

Joint objective (learning matchings for tours). Training couples the spanning-tree and matching components through the downstream tour length. Concretely, the TSP model produces X which is then decomposed through FW DECOMPOSITION over the spanning-tree polytope, and that yields a short list of trees with weights $\{\alpha_t\}$. For each tree t, we form its odd set O_t, which then goes through the neural network which outputs S_t on E_{O_t}, and we follow that by applying FW to obtain matchings $\{M_{t,k}\}$ with weights $\{\beta_{t,k}\}$. We evaluate the Christofides tour length for each (t,k) pair and minimize the mixture objective

$$\mathcal{L}_{\text{tour}} = \sum_t \alpha_t \sum_k \beta_{t,k} \left(\textsc{Christofides}(T_t, M_{t,k})\right),$$

where $\textsc{Christofides}(T_t, M_{t,k})$ represents the length of the TSP tour generated by the given parameters. So the learned matchings are directly trained to be *useful for producing short tours* under our pipeline. We also include reconstruction loss in the same way as with other problems, in order to keep X close or inside the polytope. This training also does not require ground-truth matchings; gradients are driven by the downstream TSP objective through the Christofides rounding.

Matching network architecture. The learned matching utilizes a lightweight 2 layer MLP structure to perform fast inferences, it encodes nodes using Fourier positional encoding of the 2D coordinates as input features. The matching FW EXTENSION uses a small iteration budget $T_{\text{match}} = 8$. The TSP neural network working on the spanning tree polytope follows the same setup as Section E.1.

Empirically, the learned matching variant improves objective values over the greedy-matching version (Table 10) at the cost of higher inference time, and the runtime breakdown (Table 11) shows that matching can dominate the additional overhead on large instances.

E.6 Ablation Study and Discussion on Projection

Following the experiments for Maximum Coverage, we ablate the effect of projecting the output of the neural network back into the feasible polytope. Table 13 shows that, for TSP, removing projection yields slightly better performance on both TSP-500 and TSP-1000.

We attribute this to a structural property of (2D) Euclidean TSP. Unlike problems with highly heterogeneous instance (e.g., QAP), Euclidean TSP is constrained by a low-dimensional geometric

embedding and metric distances, which makes extremely "hard" out-of-distribution instances comparatively rare—both for synthetic generators and for real-world benchmarks that remain within the same problem family. To support this claim empirically, we further test the model trained on TSP-1000 on 2D Euclidean instances from **TSPLIB** Reinelt (1991) with 51 to 1002 nodes. As shown in Table 14, our method remains robust even on this substantially broader collection.

These results suggest that, for problems like Euclidean TSP, explicit projection is not only unnecessary but can be counterproductive, while the no-projection design is fully viable and can even yield better performance in practice.

Table 13: Ablation results with and without projection

Dataset	With Projection	No Projection
TSP-500	18.06	17.76
TSP-1000	25.47	25.03

Table 14: Performance on TSPLIB of the model trained on TSP-1000.

Train Set	Test Set	Avg. Pred	Avg. Opt	Gap	Avg. Time (s)
TSP-1000	TSPLIB	8.848170	8.061844	9.75%	1.703

E.7 Ablation Study: LP + Noise

To test whether the gains come from *learned* structure rather than arbitrary perturbations of the LP relaxation, we construct a randomized baseline by adding i.i.d. Gaussian noise to the subtour-LP solution: $\tilde{x} = x_{\mathrm{LP}} + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma^2)$ and $\sigma = 0.2$. We then run the *same* FW decomposition and downstream rounding pipeline using $\tilde{x}$ as the input weights. For each instance we repeat this procedure for 10 independent noise draws and report the *best* (shortest) tour length, which is a favorable "best-of-10" setting for the noise baseline. As shown in Table 15, this randomized perturbation is unable to match our method, with the gap becoming especially pronounced on larger instances (TSP500–TSP1000). This indicates that our network learns instance-dependent signals that improve the pipeline beyond what can be obtained by injecting random noise into x_{LP}.

	LP + Gaussian noise+FW (σ=0.2, best of 10)	**Ours**
TSP50	5.93	5.79
TSP100	8.86	8.03
TSP500	32.83	17.76
TSP1000	65.51	25.03

Table 15: Ablation results on replacing the learned edge weights with random perturbations of the LP solution. Lower is better.

F Further Discussion on Inference Settings

In the field of Neural Combinatorial Optimization (NCO), strategies to refine solution quality *during inference* are widely adopted. While terminology varies across domains—often referred to as Test-Time Optimization (TTO) or Local Search/Sampling—these procedures share a common goal: to improve the solution beyond a single neural forward pass. Broadly, they fall into three categories:

1. **Gradient-based Active Search:** Fine-tuning model parameters on individual test instances (e.g., active search).
2. **Search-based Refinement:** Applying discrete search algorithms, such as beam search, MCTS, or heuristic local search (e.g., 2-opt), to the model's output.
3. **Parallel Sampling:** Running multiple inference trajectories, such as multiple diffusion noise seeds, batched augmentations, or independent neural forward passes, to enlarge the

candidate pool and select the best solution. We distinguish this from native candidate-set decoding, where a single neural forward pass deterministically or procedurally produces several candidates as part of the method's prescribed decoder.

Decoupling Learned Priors from Search Strategies. While these inference-time techniques significantly enhance performance, they introduce a critical challenge in evaluation: **they often obscure the intrinsic quality of the learned neural model**.

Recent studies have critically examined this phenomenon. Böther et al. (2022) demonstrated that in tree-search-based neural methods, the search algorithm frequently performs the majority of the "heavy lifting," rendering the contribution of the learned neural guidance marginal or, in some cases, redundant. Xia et al. (2024) finds that this is also true for tree-search based heatmap recovery. They further point out that this inconsistency is particularly concerning because the search algorithm, which is critical for determining the final solution, is not integrated during neural network training. This confounding effect where the solver relies more on inference time compute than on the learned structural prior has also been noted in recent works such as **UCOM2** Bu et al. (2024) and **GeoNCO** Karalias et al. (2025), which sets a clear difference between TTO and non-TTO methods and make no comparison between them.

Evaluation Protocol: The "One-Shot" Setting. In this work, our primary goal is to evaluate the intrinsic constructive capability of the proposed framework. We aim to assess how well the model aligns with the problem structure in a minimal-latency setting, without relying on extensive post-hoc search. Consequently, we strictly standardize the evaluation to a **non-TTO** context:

- For Maximum Coverage baselines (e.g., **CardNN**, **UCOM2**, **GeoNCO**), we utilize their "short" inference modes which comes without gradient-based finetune on inference or iterative refinement steps.
- For TSP baselines (e.g., **DIMES**, **DIFUSCO**, **COExpander**), which often rely on active search, parallelized diffusion steps or local search to refine results, we restrict them to their greedy or native candidate-set decoding equivalents. We exclude parallel sampling strategies that enlarge the candidate pool through repeated inference, such as running diffusion-based solvers with multiple noise seeds. However, when a method's native decoder generates multiple feasible candidates from one neural network pass and selects the best one, we still regard it as one-shot. For example, **DIMES (RL+S)** uses a number of samples generated from a single neural network pass, so this is treated as native candidate-set decoding rather than parallel sampling. Our FW decomposition follows the same principle: one neural output is decomposed into a sparse set of feasible vertices, and inference reports the best vertex among them.

By isolating the raw performance of the underlying neural architectures, we provide a clearer benchmark of their structural learning capabilities, ensuring that improvements are attributed to better representation learning rather than increased inference computation. This protocol also keeps the comparison consistent with the decoding budgets used by strong baselines, where candidate-set selection is often part of the native decoder, while excluding additional inference-time computation whose main effect is to enlarge the search pool.

Performance with Test-Time Optimization. As discussed above, our concern with TTO is that additional inference-time search can make it difficult to assess the quality of the learned model itself: the final solution reflects both the model and the search procedure, and may depend strongly on the search budget. We therefore choose to use the no-TTO setting for our primary evaluation. This choice does not limit the applicability of TTO to our framework. For completeness, we also apply the same TTO procedures used by GeoNCO Karalias et al. (2025) to FW-NCO on Maximum Coverage. Table 16 reports results at three inference budgets, with Short corresponding to no TTO. FW-NCO benefits substantially from TTO: coverage increases from 42172.44 to 44924.09 on Random500 with $k = 50$, and from 83559.22 to 89873.32 on Random1000 with $k = 100$. It also outperforms GeoNCO in all four settings at every TTO level. Thus, the no-TTO protocol enables a clearer assessment of native model performance, while the matched TTO comparison shows that our method's advantage persists when additional test-time optimization is applied.

Table 16: Maximum Coverage results under different levels of test-time optimization (TTO). Short corresponds to no TTO.

		Random500		Random1000	
Method	TTO	$k = 10$	$k = 50$	$k = 20$	$k = 100$
Greedy	None	15640.99	44597.56	31105.89	88685.40
GeoNCO	Short	15177.77	41247.80	29810.12	81357.29
	Medium	15410.23	43152.87	30257.05	84186.79
	Long	15714.63	44866.84	31230.26	88796.03
FW-NCO	Short	15192.64	42172.44	29926.63	83559.22
	Medium	15638.14	44285.09	31162.54	86740.46
	Long	15720.45	44924.09	31338.52	89873.32

G GPU Memory Usage

We report the GPU memory footprint of FWNCO across all training configurations. Table 17 reports peak CUDA allocated memory during training. These measurements describe allocated GPU memory, rather than total device-memory consumption, which can additionally include reserved but unused memory and CUDA runtime overhead.

Table 17: Peak CUDA allocated memory during FWNCO training.

Problem	Dataset / k	Peak memory (GiB)
MC	Random500, $k = 10$	0.135
MC	Random500, $k = 50$	0.135
MC	Random1000, $k = 20$	0.194
MC	Random1000, $k = 100$	0.194
TSP	TSP50	0.043
TSP	TSP100	0.106
TSP	TSP500	2.012
TSP	TSP1000	7.708
QAP	QAP32	5.168

Overall, the framework has a relatively modest memory footprint of below 8 GiB for every configuration. These results indicate that FWNCO does not require the latest high-end GPUs for the evaluated configurations and suggest potential for scaling to substantially larger instances.

H Polytopes with Efficient (Approximate) LMO

Matroid polytope Matroid polytopes provide a unified geometric framework for a broad class of combinatorial constraints. Given a matroid $\mathcal{M} = (V, \mathcal{I})$ with rank function $r(\cdot)$, Edmonds' classical formulation characterizes the associated matroid base polytope as

$$\mathcal{P}_{\text{base}} = \left\{ \mathbf{x} \in \mathbb{R}^n_{\geq 0} \;\middle|\; \sum_{i \in A} x_i \leq r(A)\ \forall A \subseteq V,\ \sum_{i \in V} x_i = r(V) \right\}.$$

A key property of this polytope is that it admits an efficient LMO: optimizing a linear function over $\mathcal{P}_{\text{base}}$ reduces to finding a maximum-weight base of the matroid, which can be solved in polynomial time via the greedy algorithm. This class of polytopes subsumes several fundamental constraints as special cases:

- **Cardinality constraints (uniform matroid).** The ground set is $V = \{1, \dots, n\}$ with rank function $r(A) = \min\{|A|, k\}$. The corresponding base polytope is the hypersimplex $\Delta_{n,k} = \left\{\mathbf{x} \in \mathbb{R}^n_{\geq 0} \;\middle|\; \sum_{i=1}^n x_i = k,\ 0 \leq x_i \leq 1\right\}$.
- **Partition matroids.** Let $V = \bigcup_{j=1}^c V_j$ be a partition of the ground set with associated budgets $k_1, \dots, k_c$. The rank function is $r(A) = \sum_{j=1}^c \min\{|A \cap V_j|, k_j\}$, and the base polytope is given by $\left\{\mathbf{x} \in \mathbb{R}^n_{\geq 0} \;\middle|\; \sum_{i \in V_j} x_i = k_j\ \forall j\right\}$.
- **Spanning tree constraints (graphic matroid).** Given an undirected graph $G = (V, E)$, the ground set consists of edges and the rank function is $r(A) = |V| - \kappa(A)$, where $\kappa(A)$ denotes the number of connected components in (V, A). The corresponding base polytope is $\left\{\mathbf{x} \in \mathbb{R}^{|E|}_{\geq 0} \;\middle|\; \sum_{e \in E} x_e = |V| - 1,\ \sum_{e \in A} x_e \leq |V| - \kappa(A)\ \forall A \subseteq E\right\}$.

Our setup naturally covers all of these cases within a single algorithmic framework. In contrast, Karalias et al. (2025) study these settings in a case-specific manner, requiring the design of separate projection operators onto each polytope as well as bespoke decomposition procedures tailored to each constraint. By relying solely on the availability of an efficient LMO, our results apply uniformly across all these matroidal constraints without modifying the algorithm.

Birkhoff polytope An $n \times n$ matrix is called *doubly stochastic* if all of its entries are nonnegative and the sum of the entries in each row and each column equals one. *Permutation matrices* form a special subclass of doubly stochastic matrices, characterized by having exactly one entry equal to one in each row and column and zeros elsewhere. The set of all $n \times n$ doubly stochastic matrices constitutes a convex polytope known as the *Birkhoff polytope* $\mathcal{B}_n$. The Birkhoff polytope lies in an $(n-1)^2$-dimensional affine subspace of n^2-dimensional Euclidean space defined by $2n - 1$ independent linear constraints specifying that the row and column sums all equal 1. The extreme points of Birkhoff polytope $\mathcal{B}_n$ are exactly the permutation matrices. Given a residual matrix $R \in \mathbb{R}^{n \times n}$, the LMO over $\mathcal{B}_n$ solves

$$P^\star \in \operatorname*{arg\,max}_{P \in \mathrm{ext}(\mathcal{B}_n)} \langle R, P \rangle = \sum_i \sum_j R(i,j) P(i,j)$$

This problem is equivalent to finding a maximum-weight perfect matching in a complete bipartite graph $G = (U, V, E)$, where U and V index the rows and columns of R, respectively, and each edge $(i, j) \in E$ has weight $R(i, j)$. Hence the linear maximization problem over Birkhoff polytope can be solved using the Hungarian algorithm in time $O(n^3)$. Hence, our approach subsumes the setting considered in Nerem et al. (2025) without requiring any projection onto the Birkhoff polytope. In contrast, the authors of Nerem et al. (2025) develop a specialized decomposition algorithm and introduce an additional loss term to penalize neural network outputs that lie outside the polytope.

DMO: greedy bipartite matching. Note that a simple greedy algorithm produces a 0.5-approximation to maximum weight matching for any (positively) weighted graph. This still outputs a rather good matching while being significantly faster than running an exact Hungarian algorithm oracle at every iteration. Empirically, we observe that the greedy DMO does not sacrifice quality.

Matching polytope Building on the discussion of the Birkhoff polytope, our approach also applies to matching and perfect matching polytopes.