
Geometric Algorithms for Neural Combinatorial Optimization with Constraints

Anonymous Author(s)

Affiliation

Address

email

Abstract

1 Self-Supervised Learning (SSL) for Combinatorial Optimization (CO) is an emerg-
2 ing paradigm for solving combinatorial problems using neural networks. In this
3 paper, we address a central challenge of SSL for CO: solving problems with
4 discrete constraints. We design an end-to-end differentiable framework that en-
5 ables us to solve discrete constrained optimization problems with neural networks.
6 Concretely, we leverage algorithmic techniques from the literature on convex ge-
7 ometry and Carathéodory’s theorem to decompose neural network outputs into
8 convex combinations of polytope corners that correspond to feasible sets. This
9 decomposition-based approach enables self-supervised training but also ensures
10 efficient quality-preserving rounding of the neural net output into feasible solutions.
11 Extensive experiments in cardinality-constrained optimization show that our ap-
12 proach can consistently outperform neural baselines. We further demonstrate that
13 our method generalizes beyond cardinality-constrained problems to a diverse set of
14 combinatorial optimization tasks, including finding independent sets in graphs, and
15 solving matroid-constrained problems.

16 1 Introduction

17 Combinatorial Optimization (CO) encompasses a broad category of optimization problems where the
18 objective is to find a configuration of discrete objects that satisfies specific constraints and is optimal
19 given a prescribed criterion. It has a wide range of real-world applications [64, 58, 30, 6, 93, 74]
20 while also being of central importance in complexity theory and algorithm design. These problems
21 are often non-convex, and solving them involves exploring large-scale discrete combinatorial spaces
22 of configurations. To that end, neural networks have emerged as a powerful tool for designing CO
23 algorithms as they can learn to exploit patterns in real-world data and discover high-quality solutions
24 efficiently. A compelling proposal in that direction is self-supervised CO because it eschews the need
25 to acquire labeled data, which can be computationally expensive for those problems. This usually
26 involves training a neural network to produce solutions to input instances by minimizing a continuous
27 proxy for the discrete objective of the problem [34, 83, 1, 35, 89]. While the efficiency and scalability
28 of this approach are appealing, it also harbors significant challenges.

29 Enforcing constraints on the output of the neural network is one of the primary obstacles. When
30 training, this is often done by including a weighted term in the loss function that penalizes solutions
31 that do not comply with the constraints. There are two key considerations when doing this. Having
32 access to a differentiable function that encodes constraint violation for the discrete problem, and
33 carefully tuning its contribution to the overall loss. Furthermore, at inference, it is important to
34 guarantee that the output of the neural network is discrete and it complies with the constraints.
35 In practice, this is usually achieved with an algorithm that rounds the continuous output of the
36 neural network to a feasible discrete solution. Such algorithms are often heuristics and treat inference

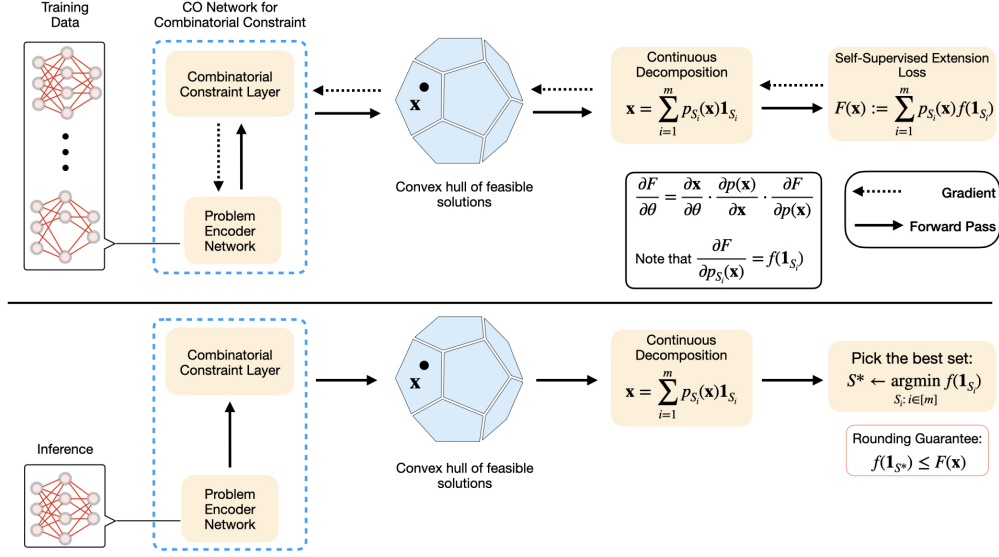


Figure 1: Overview of our framework. During training (top), the model learns to output a point in the convex hull of feasible solutions. A self-supervised extension loss is computed by decomposing this point and evaluating the objective, which is then used to backpropagate. During inference (bottom), the model outputs a relaxed solution, which is decomposed and the best feasible set is selected with a rounding guarantee.

37 differently from training which may hurt performance. Despite recent developments that have enabled
38 tackling new combinatorial problems in the self-supervised setting, combining loss function design
39 and rounding in a coherent fashion can still be highly nontrivial.

40 In this work, building on ideas from the literature on self-supervised CO and techniques from
41 geometric algorithms and combinatorial optimization, we propose an approach to loss function design
42 that seamlessly integrates training and inference while also providing a rounding guarantee. The
43 main idea is to *learn to map input instances to distributions of feasible solutions* that maximize
44 (or minimize resp.) the objective. Given an input instance, we use the neural network to predict a
45 continuous vector in the convex hull of feasible solutions. Using an iterative algorithm, this vector is
46 decomposed into a distribution of discrete feasible sets. This algorithm draws from the literature on
47 Carathéodory’s theorem [12] to express points in the interior of a convex polytope as a sparse convex
48 combination of the corners of the polytope. This decomposition yields a distribution that allows us to
49 tractably calculate the expected value of the discrete objective. We use this expectation as our loss
50 and minimize it with a neural net and standard automatic differentiation in a self-supervised fashion.
51 By backpropagating derivatives from this expectation, the neural network learns to generate outputs
52 in the feasible set that optimize the expected objective. At inference time, the iterative algorithm
53 generates candidate feasible solutions to the problem. The pipeline is summarized in Figure 1.

54 **Our contribution.** Our method generalizes previous work, and offers strong empirical benefits in
55 efficiency and performance on large-scale CO instances. Our contributions can be summarized as
56 follows:

- 57 • We use geometric algorithms to design an end-to-end pipeline for constrained neural CO.
- 58 • Our geometric algorithms effectively tackle both the challenge of loss function design for
- 59 training with constraints, and the challenge of rounding neural network outputs at inference.
- 60 • We show strong results in cardinality constrained CO on large-scale instances with hundreds
- 61 of thousands of nodes, often surpassing previous neural baselines and heuristics. We also
- 62 show how the method can be extended to other matroid constraints and independent sets.
- 63 • We conduct ablation studies to test the effectiveness of optimizing the extension with gradient
- 64 descent and the contribution of the model in the performance.

65 2 Related work

66 **Self-Supervised CO and continuous extensions.** Our work follows a long line of work on unsu-
 67 pervised CO [1, 2, 34, 83, 35, 81, 86, 29, 73, 23, 63, 77, 42, 60, 89]. Crucial components of those
 68 pipelines include the choice of model, the use of specific input features (positional encodings), the loss
 69 function, and the rounding algorithm. There has been extensive research on neural net architectures
 70 for combinatorial problems [57, 84, 75, 92] and the role of input features for well known classes of
 71 models [90, 53, 37]. Our primary focus will be on loss function design and rounding. We build on
 72 previous work that proposed using continuous extensions of discrete functions as losses for neural
 73 CO [35]. Following this blueprint, bespoke extensions have been used for learning on permutations
 74 [62], hierarchical clustering [42] and interpretable graph learning [15]. However, there is no general
 75 extension design paradigm that can accommodate different constraints. We address this issue by
 76 proposing a template for extension design and show how it enables building the constraints into the
 77 distribution. This streamlines the process of building extensions and dispenses with the need to tune
 78 constraint terms in the loss function that is present in other approaches to self-supervised CO.

79 **Loss function design and enforcing constraints.** The design of specific loss functions in self-
 80 supervised CO to enable smoother optimization, integration with different powerful architectures,
 81 and/or better rounding has been studied over the past few years. A common approach is to adopt
 82 a probabilistic perspective and parametrize a distribution of outputs with a neural network that
 83 learns to optimize the expected cost and the probability of constraint violation [60, 86, 35, 73, 62].
 84 Deriving and/or computing those expectations and probability terms for more complex problems can
 85 be difficult, which motivated the work by [9] that provides derivations and fast rounding techniques
 86 for cases such as spanning trees and cardinality constraints. In our work, the constraints are built
 87 into the distribution and the loss only has to handle the objective function. It is challenging to craft
 88 such losses which has led to the development of techniques that improve training dynamics via
 89 annealed training [81, 32]. Other approaches to loss function design leverage ideas from physics
 90 [77, 78], and semidefinite programming [92, 31]. Those cases also involve tunable constraint terms.
 91 Another major consideration is that of enforcing constraints on the outputs of the neural networks.
 92 Techniques in the literature typically involve calculating the projection of the neural net output
 93 onto the feasible set. This has been done for cardinality constraints using ideas from the theory of
 94 optimal transport [88], and general linear constraints with gradient-based techniques [17, 94]. In the
 95 continuous (potentially non-convex) optimization setting, several projection techniques have also
 96 been developed [51, 52, 50, 48].

97 **Convex geometry and learning over polytopes.** Our approach relies on fundamental results from
 98 convex geometry and geometric algorithms. Specifically, the Carathéodory Theorem states that any
 99 point in a polytope $P \subseteq \mathbb{R}^d$ can be expressed as a convex combination of at most $d + 1$ points. The
 100 constructive versions of this result [24] have been adapted to yield efficient algorithms in polytopes
 101 for ranking [41] and scheduling problems [28, 20, 7]. Approximate versions of the decomposition
 102 have been developed that do not explicitly depend on the ambient dimension d [3] and admit efficient
 103 algorithms with fast convergence rates [59, 16]. In the context of machine learning, polyhedral
 104 methods have been used extensively to impose constraints on the outputs of models for generative
 105 modelling [21], for learning with end-to-end differentiable sampling from exponential families using
 106 perturb-and-map techniques [67], for applications to differentiable sorting [5, 72] and learning to
 107 optimize over permutations [25, 56]. For additional discussion, please refer to the appendix.

108 3 Problem formulation and learning setup

109 Let $V = \{v_1, \dots, v_n\}$ be a set of variables, each assigned a value from a discrete domain $D =$
 110 $\{1, \dots, d\}$. A combinatorial optimization problem consists of a universe of instances \mathcal{T} , where each
 111 instance $T \in \mathcal{T}$ can be represented by a tuple $T = (V, D, \mathcal{C}, f)$. The set $\mathcal{C} \subseteq D^n$ defines the feasible
 112 solutions to the instance. The real-valued function $f : D^n \rightarrow \mathbb{R}$ is called the objective function. The
 113 problem is then to find the optimal feasible vector \mathbf{x}^* , i.e., to find the feasible vector $\mathbf{x} \in D^n$ that
 114 maximizes the objective function f :

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

115 In this paper, we focus on CO problems with Boolean domain, i.e., $D = \{0, 1\}$, and $\mathcal{C} \subseteq \{0, 1\}^n$.
 116 This domain choice corresponds to combinatorial optimization problems that involve set functions,

117 i.e., \mathbf{x} indicates a subset $S \subseteq V$ of a ground set $V = \{v_1, \dots, v_n\}$ of elements and $x_i = 1$ if element
 118 i is in the subset, and f maps each subset to a real number. The subsets can be, for instance, nodes or
 119 edges in a graph, and are central to many problems in applied mathematics and operations research [8].
 120 The set $\mathcal{C} \subseteq \{0, 1\}^n$ includes all subsets of V that obey the constraints of the problem. Each subset
 121 S is identified with its indicator vector $\mathbf{1}_S \in \{0, 1\}^n$ and we use $f(S)$ and $f(\mathbf{1}_S)$ interchangeably
 122 throughout this paper. The objective function is the set function $f : \{0, 1\}^n \rightarrow \mathbb{R}$. The goal is to
 123 select the optimal feasible subset so Equation (1) takes the form $\max_{S \in \mathcal{C}} f(S)$.

124 **Self-Supervised CO.** Given a problem instance T and input features $\mathbf{Z}_T \in \mathbb{R}^{n \times d}$, we use a neural
 125 network NN_θ , which we will often refer to as the encoder network, to map the instance data to an
 126 output prediction $\mathbf{x} \in \mathbb{R}^n$ by computing $\mathbf{x} = \text{NN}_\theta(\mathbf{Z}_T; T)$. For example, the input instance could
 127 be a graph, the input features could be positional encodings for the graph, and the neural net a
 128 Multi-Layer Perceptron (MLP). The output \mathbf{x} may not correspond to a discrete feasible point. In
 129 those cases, a rounding algorithm will be used to map \mathbf{x} to a feasible indicator vector of a set. The
 130 goal is for the neural net to learn to predict the optimal solution \mathbf{x}^* . In self-supervised CO, this is
 131 done by training NN_θ on a collection of instances T_1, T_2, \dots, T_m . The neural net minimizes the
 132 problem-specific loss function \mathcal{L}_T computed for each instance $\mathcal{L}_T(\text{NN}_\theta(\mathbf{Z}_{T_i}; T_i))$ and averaged over
 133 a batch (or the entire training set). Since there are no labels, how the loss function is designed to fit
 134 the constraints of the problem is of critical importance. At inference time, a test instance is processed
 135 through the neural network to obtain a prediction. At this step, rounding heuristics are typically used
 136 to ensure that the solution is feasible and discrete. It should be noted that, instead of predicting, it is
 137 possible to directly optimize the model with gradient descent on the test instances since no labels
 138 are leveraged during training. In this case, the algorithm incurs the additional computational cost of
 139 backpropagation which may not be practical in time-critical settings.

140 4 Proposed approach

141 First, we present an overview of the key components in the pipeline and then we discuss each
 142 component in more detail. Our proposed method focuses on the design of $\mathcal{L}_T(\mathbf{x})$ that is used to train
 143 the model. The objective f is a set function with discrete domain and cannot be directly optimized.
 144 Following the literature, we construct a continuous extension of f , denoted by $F(\mathbf{x}) : [0, 1]^n \rightarrow \mathbb{R}$,
 145 and use it as our loss function, i.e. $\mathcal{L}_T(\mathbf{x}) = F(\mathbf{x})$. The extension is defined as

$$F(\mathbf{x}) := \mathbb{E}_{S \sim \mathcal{D}_C(\mathbf{x})}[f(S)], \quad (2)$$

146 where $\mathcal{D}_C(\mathbf{x})$ is a distribution parametrized by the output of the neural network and $S \in \mathcal{C}$ for all
 147 S in the support of $\mathcal{D}_C(\mathbf{x})$. The key challenge lies in how to tractably construct and parameterize
 148 such a distribution on feasible sets in a way that enables end-to-end learning. We achieve this using
 149 an iterative algorithm that ensures $\mathcal{D}_C(\mathbf{x})$ is supported on $O(n)$ feasible sets and the probability of
 150 each set is a differentiable function of \mathbf{x} . This allows us to efficiently compute the exact expectation
 151 in Equation (2) and to calculate its derivatives with standard automatic differentiation packages. At
 152 inference time, the geometric algorithm can be used as a rounding algorithm for \mathbf{x} since $\mathcal{D}_C(\mathbf{x})$ has a
 153 small number of sets in its support. Specifically, we can round \mathbf{x} to the best set in the support.

154 4.1 A general decomposition algorithm for extension design

155 For a feasible set \mathcal{C} , consider the polytope $\mathcal{P}(\mathcal{C}) = \text{Conv}(\{\mathbf{1}_S : S \in \mathcal{C}\})$, i.e., the convex hull of the
 156 indicator vectors of the sets in \mathcal{C} . Any point $\mathbf{x} \in \mathcal{P}(\mathcal{C})$ is a convex combination $\mathbf{x} = \sum_{S \in \mathcal{C}} \alpha_S \mathbf{1}_S$
 157 of feasible solutions from \mathcal{C} . In fact, from Carathéodory's theorem we know that this convex
 158 combination requires at most $n + 1$ points where n is the ambient dimension of the space. The
 159 coefficients of the convex combination can be viewed as the probabilities of a distribution $\mathcal{D}_C(\mathbf{x})$
 160 and hence $\mathbf{x} = \mathbb{E}_{S \sim \mathcal{D}_C(\mathbf{x})}[\mathbf{1}_S]$. If we can uniquely and efficiently decompose each point $\mathbf{x} \in \mathcal{P}(\mathcal{C})$
 161 into such a convex combination, we can leverage that distribution to compute the extension in
 162 Equation (2). In Algorithm 1, we propose a general template for an iterative algorithm that yields
 163 such decompositions and forms the foundation for our approach.

164 Intuitively, the algorithm decomposes \mathbf{x} iteratively by removing a corner $\mathbf{1}_S$ weighted by a suitably
 165 chosen coefficient in each iteration. This is done until the coefficients and the corresponding corners
 166 can be used to reconstruct \mathbf{x} up to some small error ϵ .

Let $\mathbf{x} \in \mathcal{P}(\mathcal{C})$ and $\mathbf{x}_0 = \mathbf{x}$. In each iteration t , the algorithm finds a feasible solution in \mathcal{C} that is both an extremal point of the convex hull and lies in the support of \mathbf{x}_t . Let this point be $\mathbf{1}_{S_t} \in \mathcal{C} \subseteq \{0, 1\}^n$. Let $a_t \in [0, 1]$ be the coefficient at iteration t . We express each iterate \mathbf{x}_t as

$$\mathbf{x}_t = a_t \mathbf{1}_{S_t} + (1 - a_t) \mathbf{x}_{t+1}, \quad (3)$$

where a_t is a function of \mathbf{x}_t and is chosen such that \mathbf{x}_{t+1} remains in $\mathcal{P}(\mathcal{C})$. We can recursively apply the same process to \mathbf{x}_{t+1} until \mathbf{x} has been decomposed into a convex combination of feasible sets. After the algorithm terminates, for $a_0 = 0$, we can obtain a distribution $\mathcal{D}_{\mathcal{C}}(\mathbf{x})$ which

is completely characterized by the sets S_t and probabilities $p_{\mathbf{x}_t}(S_t) = a_t \prod_{i=0}^{t-1} (1 - a_i)$ of the decomposition. Determining the specific form of steps 3 and 4 in a way that yields a polynomial time algorithm is a non-trivial task that depends on the polytope. The following theorem is a constructive version of the Carathéodory theorem, also known in the literature as the GLS Method (from Grötschel-Lovász-Schrijver) [28, 41]. It establishes such a decomposition for a large class of polytopes.

Theorem 4.1. (Informal) *There exists polynomial-time algorithm that for any well-described polyhedron \mathcal{P} given by a strong optimization oracle, for any rational vector \mathbf{x} , finds vertex-probability pairs $\{p_{\mathbf{x}_t}(S_t), \mathbf{1}_{S_t}\}$ for $t = 1, 2, \dots, n$ such that $\mathbf{x} = \sum_{t=1}^n p_{\mathbf{x}_t}(S_t) \mathbf{1}_{S_t}$.*

Briefly, the proof proceeds by assuming access to a certain polynomial-time optimization oracle for Step 3 in Algorithm 1. The algorithm leverages the oracle to obtain a vertex for a face of the polytope that contains the current iterate. At step 4, a_t is chosen so that \mathbf{x}_{t+1} intersects the boundary of the polytope. In each iteration, the iterate lies on a lower-dimensional face than the previous one, and hence the algorithm terminates in polynomial time, in at most n iterations [24, Theorem 6.5.11].

Necessary conditions for end-to-end learning. Our approach draws insights from this theorem to design a decomposition that can be integrated in gradient-based optimization pipelines. Here, we summarize the basic conditions for the general decomposition to yield an extension that can be optimized with standard automatic differentiation, which will enable end-to-end learning. We require:

1. A differentiable method that ensures our starting point \mathbf{x} lies in the polytope $\mathcal{P}(\mathcal{C})$.
2. A routine that efficiently finds a vertex $\mathbf{1}_{S_t} \in \mathcal{P}(\mathcal{C})$ in the support of \mathbf{x}_t .
3. An (almost everywhere) differentiable function $g(\mathbf{x}_t, S)$ that yields a valid decomposition.

Satisfying these conditions will allow us to tractably construct a distribution of feasible sets $\mathcal{D}_{\mathcal{C}}(\mathbf{x})$, which will enable efficient calculation of the expectation in Equation (2). It also means that we can update the parameters of the neural network that is used to predict \mathbf{x} by optimizing the extension via standard automatic differentiation packages. Each of these steps involves different subtleties depending on the problem. We will show these requirements can be fulfilled for several problems.

Neural predictions in the feasible set polytope. Suppose our encoder network outputs $\mathbf{x} \in \mathbb{R}^n$. Since Algorithm 1 requires a point in the polytope $\mathbf{x} \in \mathcal{P}(\mathcal{C})$, we need a way to enforce this constraint on the output of our neural network. This type of problem has received significant attention (see section 2) in both the optimization and machine learning communities. Commonly, this can involve projection-based techniques such as Sinkhorn’s algorithm [79] and its various extensions [87] or other gradient-based approaches [94]. This can be a viable strategy but, depending on the polytope, may be difficult to integrate in a differentiable pipeline. We will leverage a projection-based approach when appropriate but we also introduce a simple efficient alternative. We interpret the neural encoder output as a perturbation of an interior point in $\mathcal{P}(\mathcal{C})$. More concretely, the neural net predicts a perturbation vector $\mathbf{z} \in \mathbb{R}^n$ such that $\mathbf{x} = \mathbf{z} + \mathbf{u}$, where $\mathbf{u} \in \mathbb{R}^n$ is an interior point of $\mathcal{P}(\mathcal{C})$. While this approach circumvents the need for projection, it requires access to an interior point of the polytope. It also requires care so that the perturbed point $\mathbf{z} + \mathbf{u}$ remains in the polytope. Nonetheless, we show that for several fundamental constraint families—including cardinality constraints, partition matroids, and the spanning tree base polytope—it is possible to do this efficiently in a differentiable way (e.g., Prop. 4.3).

Algorithm 1 General decomposition algorithm

Require: $\mathbf{x} \in [0, 1]^n$ in $\mathcal{P}(\mathcal{C})$.

1: $\mathbf{x}_0 \leftarrow \mathbf{x}$, $a_0 \leftarrow 0$, $t \leftarrow 1$.

2: **repeat**

3: $\mathbf{1}_{S_t} \leftarrow$ vertex from $\mathcal{P}(\mathcal{C})$ in the support of \mathbf{x}_t .

4: $a_t \leftarrow g(\mathbf{x}_t, S)$

5: $\mathbf{x}_{t+1} \leftarrow (\mathbf{x}_t - a_t \mathbf{1}_{S_t}) / (1 - a_t)$

6: $p_{\mathbf{x}_t}(S_t) \leftarrow a_t \prod_{i=0}^{t-1} (1 - a_i)$

7: $t \leftarrow t + 1$

8: **until** $\|\mathbf{x} - \sum_t p_{\mathbf{x}_t}(S_t) \mathbf{1}_{S_t}\| \leq \epsilon$

9: **return** All $\{p_{\mathbf{x}_t}(S_t), S_t\}$ pairs.

219 **Extension function, training, and inference.** Once the decomposition algorithm provides $\mathcal{D}_C(\mathbf{x})$, we
 220 can use it to compute our loss function. For the extension to be well defined, the decomposition needs
 221 to deterministically map each point in the polytope to vertex-probability pairs. This is guaranteed
 222 in our constructions. It is also straightforward to compute derivatives of the extension with respect
 223 to the neural network parameters. For simplicity, suppose the encoder network, parameterized by
 224 weights θ , outputs $\mathbf{x} \in \mathcal{P}(\mathcal{C})$. Then, we have $\frac{\partial F}{\partial \theta} = \frac{\partial F}{\partial p(\mathbf{x})} \cdot \frac{\partial p(\mathbf{x})}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{x}}{\partial \theta}$ where $p(\mathbf{x})$ is the vector of
 225 probabilities returned by Algorithm 1.

226 The extension $F(\mathbf{x}) = \mathbb{E}_{S \sim \mathcal{D}_C(\mathbf{x})}[f(S)] = \sum_t p_{\mathbf{x}_t}(S_t) f(S_t)$ has several desirable properties that
 227 have been studied in the literature [35]. Since we are considering convex combinations of the objective
 228 over the feasible set, the extreme points are preserved i.e., $\max_{\mathbf{x} \in \mathcal{P}(\mathcal{C})} F(\mathbf{x}) = \max_{S \in \mathcal{C}} f(S)$;
 229 moreover we have $\arg\max_{\mathbf{x} \in \mathcal{P}(\mathcal{C})} F(\mathbf{x}) \in \text{Conv}(\{\mathbf{1}_S : S \in \arg\max_{S \in \mathcal{C}} f(S)\})$. The expectation
 230 formulation also provides a rounding guarantee. There exists at least one feasible solution S^* in the
 231 decomposition of \mathbf{x} such that $f(S^*) \geq F(\mathbf{x})$. This property enables fast inference on the output of
 232 the encoder network without any degradation in solution quality.

233 4.2 Case studies

234 In this section we discuss how the decomposition can be generated for various constraints. We
 235 provide a detailed treatment of the cardinality constraint to build intuition and then discuss additional
 236 examples including spanning trees and independent sets. All proofs, additional details and additional
 237 experiments can be found in the Appendix.

238 4.2.1 Cardinality constraint

239 Here, we focus on combinatorial problems that involve optimizing a set function under an exact
 240 cardinality constraint. This setting is sufficiently general to encompass many fundamental and
 241 practical combinatorial optimization problems. Formally, let $f : 2^V \rightarrow \mathbb{R}$ be a set function that
 242 assigns a real value to each subset of $V = \{v_1, \dots, v_n\}$. For a given positive integer k , the set of
 243 feasible solutions is defined as $\mathcal{C} = \{S : |S| = k, S \subseteq V\}$, which includes all subsets of V of size
 244 exactly k . The resulting constrained combinatorial optimization problem is:

$$\max_{\mathbf{x} \in \{0,1\}^n, \|\mathbf{x}\|_1 = k} f(\mathbf{x}). \quad (4)$$

245 The associated convex polytope, $\mathcal{P}(\mathcal{C}) = \text{Conv}(\{\mathbf{1}_S : |S| = k\})$, is the convex hull of Boolean
 246 vectors with exactly k ones and $n - k$ zeros and the *matroid base polytope* corresponding to a uniform
 247 matroid on n elements of rank k . It is known as the (k,n) -hypersimplex and is denoted by $\Delta_{n,k}$ [71].

248 **Designing the decomposition algorithm.** For now, we assume that our predictions lie in the polytope
 249 of the feasible set, i.e. $\mathbf{x} \in \Delta_{n,k}$. To construct a decomposition, we need to ensure that steps 3 and 4
 250 in Algorithm 1 comply with our conditions for efficient end-to-end learning. To obtain a vertex of the
 251 polytope in the support of \mathbf{x}_t for Step 3 in Algorithm 1, we set

$$\mathbf{1}_{S_t} = \arg\max_{\mathbf{y} \in \{0,1\}^n, \|\mathbf{y}\|_1 = k} \mathbf{x}_t^\top \mathbf{y}. \quad (5)$$

252 For step 4, we use Equation (3) to calculate the largest possible coefficient that keeps the iterate in
 253 the polytope. This yields

$$a_t = \min \left\{ \min_{i \in S_t} \mathbf{x}_t(i), 1 - \max_{i \notin S_t} \mathbf{x}_t(i) \right\}. \quad (6)$$

254 The following result shows that our decomposition algorithm produces a valid convex decomposition
 255 of points in the hypersimplex and is suitable for end-to-end learning.

256 **Theorem 4.2.** *For $\mathbf{x} \in \Delta_{n,k}$, Algorithm 1 with Step 3 according to Equation (5) and Step 4
 257 according to Equation (6), terminates in at most n iterations and returns probability-vertex pairs
 258 $\{(p_{\mathbf{x}_t}(S_t), \mathbf{1}_{S_t})\}$ such that every S_t is a set of size k and $\mathbf{x} = \sum_t p_{\mathbf{x}_t}(S_t) \mathbf{1}_{S_t}$. Moreover, each
 259 $p_{\mathbf{x}_t}(S_t)$ is a continuous and a.e. differentiable function of \mathbf{x} .*

260 Intuitively, the result follows a similar strategy as Theorem B.3. Our choice of coefficient ensures
 261 that the next iterate intersects the boundary of the polytope and lies in a lower-dimensional space.
 262 This in turn guarantees that the algorithm terminates in at most n steps.

263 **Generating neural predictions in the hypersimplex.** The last concern to address is that of obtaining
 264 a vector $\mathbf{x} \in \Delta_{n,k}$ in an efficient and differentiable manner. We follow the perturbation strategy that
 265 we outlined earlier and use the neural network to predict a perturbation vector $\mathbf{z} \in [0, 1]^n$ for an
 266 interior point. The following result ensures that the perturbed point \mathbf{x} remains in the polytope.

267 **Proposition 4.3.** *Let $\mathbf{z} \in [0, 1]^n$ and $\mathbf{u} = [\frac{k}{n}, \frac{k}{n}, \dots, \frac{k}{n}]$. Let μ be the mean of \mathbf{z} , and $\tilde{\mathbf{z}}$ be the*
 268 *mean-centered version of \mathbf{z} . For $s = \min(\frac{k/n}{\mu}, \frac{(n-k)/n}{1-\mu})$ and $\mathbf{x} = s\tilde{\mathbf{z}} + \mathbf{u}$, we have that $\mathbf{x} \in \Delta_{n,k}$.*

269 This simple transformation is efficient to compute and maintains differentiability of \mathbf{x} with respect to
 270 the neural network parameters.

271 4.2.2 Decomposition for Partition Matroids, Spanning Trees, and Independent Sets

272 **Partition Matroids.** We extend our result to general constraints beyond cardinality. Let V_1, \dots, V_c
 273 be disjoint subsets with $V = \cup_{i=1}^d V_i$. The CO problem is $\max_{S: |S \cap V_i| = k_i} f(S)$. When $c = 1$, this
 274 reduces to the cardinality case. In step 3 of Algorithm 1, we obtain a vertex in the support of \mathbf{x}_t
 275 by setting $\mathbf{1}_{S_t} = \arg\max_{\mathbf{y} \in \{0,1\}^n, \|\mathbf{y}(V_i)\|_1 = k_i} \mathbf{x}_t^\top \mathbf{y}$. Step 4 matches Equation (6). The encoder can
 276 similarly be constrained to the polytope, with scaling factor s being handled per partition.

277 **Spanning Trees.** Let $G = (V, E)$ be a graph with n nodes and m edges. The CO problem is
 278 $\max_{S \subseteq E: S \in \mathcal{F}} f(S)$, where \mathcal{F} is the set of full *spanning forests* of G . The convex hull of feasible sets
 279 is the *graphical matroid base polytope*. In Step 3 of Algorithm 1, we obtain a vertex in the support of
 280 \mathbf{x}_t by finding a maximum spanning forest. Step 4 matches Equation (6). The encoder learns edge
 281 weights to perturb the uniform spanning tree distribution, where we use Kirchhoff’s Matrix–Tree
 282 Theorem [40] and results from spectral graph theory.

283 **Independent Sets.** The maximum independent set problem is an NP-hard problem where the goal is
 284 to find the largest subset of nodes S in a graph $G = (V, E)$ such that no pair of nodes in S is adjacent.
 285 There is no compact description of the polytope of independent sets of a graph, so it is a particularly
 286 challenging case. We instead work on its relaxation, which admits a compact description and is
 287 known as the *fractional stable set polytope* [55]. We use a fast gradient-based approach to project
 288 to the polytope. For Step 3 of the decomposition, we follow a similar approach to Equation (5) and
 289 use a greedy independent set algorithm. For Step 4, similar to the cardinality constraint case, in each
 290 iteration we select the largest coefficient that does not violate the polytope constraints.

291 5 Experiments

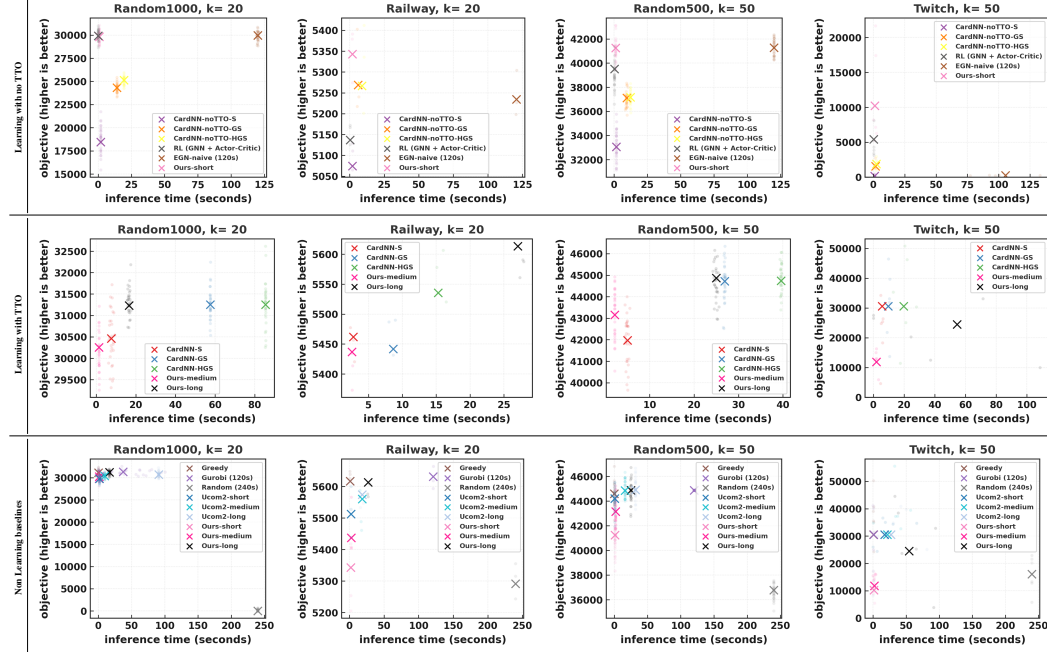
292 5.1 Experimental setup

293 We consider the Maximum Coverage problem [38], a fundamental combinatorial problem with
 294 many variants that have been extensively studied. It appears in numerous applications, including
 295 document summarization [54] and identifying influential users in social networks [36, 14]. Let
 296 $U = \{e_1, \dots, e_m\}$ be a set of n elements and $V = \{T_1, \dots, T_n\}$ be n subsets of U . Given $k \leq n$,
 297 the Max Coverage problem asks for k subsets whose union covers the largest number of elements.
 298 In the weighted version, each e_i has a weight, and the goal is to maximize the total weight of
 299 covered elements. This problem can be represented as a bipartite graph $G = (U, V, E)$, where
 300 $(e_j, T_i) \in E$ iff $e_j \in T_i$. A node $u \in U$ is covered by $S \subseteq V$ if it is adjacent to some vertex in
 301 S . Set $f(S, u) = 1$ if u is covered by S , and $f(S, u) = 0$ otherwise. The optimization problem is
 302 $\max_{S \subseteq V, |S|=k} \sum_{u \in U} f(S, u)$.

303 **Problem Encoder:** To encode the bipartite graph, we exploit three layers of GraphSage [27] followed
 304 by a fully-connected layer with our *noise perturbation layer* to map the output vector to the hypersim-
 305 plex $\Delta_{n,k}$. We also added normalization and residual connections in between the GraphSage layers
 306 to improve gradient flow and mitigate over-smoothing during training.

307 **Baselines.** We adopt the same baselines as prior work, including: random, which samples k
 308 candidates uniformly at random over multiple trials and selects the best within 240 seconds; Greedy
 309 algorithm [61] that iteratively adds the element with the largest marginal gain and achieves a
 310 well-known $(1 - \frac{1}{e})$ -approximation guarantee; Gurobi for exact MIP-based solutions, with time
 311 limited to 120 seconds for each graph; CardNN [87] and its variants (with and without test-time
 312 optimization, following [9] approach) ; a naive version of EGN with a naive probabilistic objective

Table 1: Performance comparison of our method against baseline approaches across three evaluation settings—Learning without TTO, Learning with TTO, and non-learning/traditional baselines—on multiple datasets. Metrics used are inference time (lower is better) and objective value (higher is better). In the Learning without TTO setting, our short version consistently outperforms all baselines in both inference time and objective value, demonstrating strong learning capability. When extended to medium and long versions, our method surpasses most TTO-based baselines across datasets, with the exception of the Twitch dataset. It also outperforms the greedy algorithm on multiple benchmarks.



construction and iterative rounding [34]; UCOM2 with variants [9]; and a Reinforcement Learning (RL) baseline. For the RL baseline, we follow the instructions in [45, 70] to use GraphSage layers [26] as policy network with Actor-Critic [44] algorithm to train on the same problem instances. We classify UCOM2 as a non-learning method because their incremental greedy de-randomization, when applied to a uniform noise or all-zero input vector, performs as well as—or in some cases better than—when applied to the output of their encoder network suggesting that its performance is mostly due to the greedy module.

Dataset and training. We use synthetic graphs for training and real-world graphs for testing. Two k values are chosen per dataset to assess method generalizability. (Due to space constraints, only one value of k is shown; the full set is in the appendix.) **Random graphs:** the number in front of “Random” indicates the size of U . Each group contains 100 graphs from either a uniform or Pareto distribution. **Real-world graphs:** Built from set systems like Twitch and Railway, each group includes multiple graphs from the same source for broader coverage. Due to limited large-scale data, real-world datasets are used only for testing, with training done on synthetic data. We point out that scaling the probability weights in Equation (6) generates diverse decompositions, exposing the model to more sets during training and inference. We use a list of *scaling factors* to produce these decompositions in *parallel*.

5.2 Discussion

Learnability results. Our end-to-end pipeline shows a very strong learning ability in comparison to the existing learning baselines. In the setting where no method performs optimization at inference time, the `short` version of our method already shows strong performance compared to existing baselines. In this variant, we apply our decomposition procedure to the output of the neural encoder and select the best set from the decomposition. Based on the rounding guarantees of our extension, the resulting discrete solution is provably at least as good as the encoder’s output. Compared to other

baselines and CardNN-noTTO (across all three variants), our inference method achieves superior efficiency in both runtime and output quality. Note that we do not include the UCOM2 method in this comparison, as it falls outside the scope of this setting; more details are provided later in the paper. Regarding the Twitch dataset, although our short method performs significantly better than other approaches in this setting, we observe that all learning-based methods perform poorly overall on this dataset. As mentioned in the introduction, this underperformance is largely attributed to the architecture of the encoder network and the specific characteristics of the data distribution. This highlights a need for further investigation and suggests that adapting the model architecture to better suit such data is an important direction for future work.

Test-Time Optimization (TTO). In this setting, additional computational time is allowed during inference to improve solution quality. Specifically, we perform a simple optimization procedure: after selecting the best set from the decomposition of the encoder output, we apply local improvement by swapping a few elements with those from other sets in the decomposition’s support. This lightweight optimization step demonstrates that the encoder has successfully learned meaningful structure from the data, as reflected in the quality of the decomposition’s support. This is the medium variant of our method. In the long version we moreover perform data augmentation similar to [33]. These procedures yield noticeable improvements on the Random500 and Random1000 datasets, and produce reasonably good results on the Railway dataset. On the twitch dataset, the CardNN methods directly optimize the neural network output on the test sample with Adam, therefore overcoming the model architecture problem. For our methods, performance remains poor on the Twitch dataset, consistent with observations discussed earlier.

Comparison with Non-Learning methods. While greedy is an efficient baseline with a strong approximation guarantee, our method performs competitively and is capable of outperforming it on datasets like Random500 for larger values of k . We are also able to outperform UCOM in several cases (e.g., Random1000 and Railway), though there are instances where greedy and/or UCOM perform the best, such as the twitch dataset.

Ablation. We conduct an ablation study on the NP-hard problem of finding a cardinality-constrained maximum cut. Given a graph $G = (V, E)$, the goal is to find a set S of k nodes such that the number of edges with exactly one endpoint in S is maximized. We compare the following baselines on two datasets and two settings for the value k . Our decomposition-based method consists of two main parts: a NN to predict a point in the polytope, and a loss function to optimize it and aid in learning. To probe the effect of the loss function, we compare the quality of solutions obtained by the decomposition when the input is i) a random point projected onto the polytope ii) a random point optimized with Adam on the loss. To probe the effect of the neural network, we test the solutions obtained by the decomposition of the predictions of a neural network that has optimized the loss on the same data. Finally, to assess generalization, we compare the decomposed sets obtained from the predictions of a neural net that has been trained on a separate training set. As a sanity check, we also include a greedy algorithm. Due to space constraints the table and more details of the experiment are provided in the appendix. We consistently observe that directly optimizing the extension on the test set with Adam significantly improves the objective compared to randomly decomposing a point and reporting the result of the best performing set. Furthermore, the neural net trained directly on the test set improves consistently over direct optimization with Adam and is competitive with greedy pointing to the benefits of parametrization. Finally, the performance of the SSL approach is close to the competing neural baseline, suggesting that the model generalizes well.

6 Conclusion

We proposed a novel geometric approach to neural Combinatorial Optimization with constraints that effectively tackles the challenges of loss function design and rounding. Our method achieved strong empirical results against both neural baselines and classical methods, and opens up promising avenues for further research on incorporating classical geometric algorithms into neural net pipelines. It is important to note that the model architecture can play an important role in results; while we did not address this aspect here, it merits further exploration in future work. While our framework directly applies to a range of problems, some more complex constraints will require new decomposition algorithms. This can be difficult in cases where a compact description of the convex polytope is not available. Nevertheless, we believe that the blueprint described in this work paves the way towards machine learning algorithms that will effectively tackle these challenges.

References

- [1] Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Learning to solve circuit-sat: An unsupervised differentiable approach. In *International conference on learning representations*, 2018. 1, 3
- [2] Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Pdp: A general neural framework for learning constraint satisfaction solvers, 2019. 3
- [3] 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*, pages 361–369, 2015. 3
- [4] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning, 2017. URL <https://openreview.net/forum?id=rJY3vK9eg>. 16
- [5] Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis R Bach. Learning with differentiable perturbed optimizers. In *NeurIPS*, 2020. 3
- [6] Lucio Bianco, Giovanni Rinaldi, and Antonio Sassano. *A combinatorial optimization approach to aircraft sequencing problem*. Springer, 1987. 1
- [7] Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, and Pramod Viswanath. Costly circuits, submodular schedules and approximate carathéodory theorems. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*, pages 75–88, 2016. 3
- [8] Endre Boros and Peter L Hammer. Pseudo-boolean optimization. *Discrete applied mathematics*, 123(1-3):155–225, 2002. 4
- [9] Fanchen Bu, Hyeonsoo Jo, Soo Yong Lee, Sungsoo Ahn, and Kijung Shin. Tackling prevalent conditions in unsupervised combinatorial optimization: Cardinality, minimum, covering, and more. *arXiv preprint arXiv:2405.08424*, 2024. 3, 7, 8, 28, 29
- [10] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák. Maximizing a submodular set function subject to a matroid constraint. *SIAM J. Computing*, 40(6), 2011. 20
- [11] Quentin Cappart, Didier Chételat, Elias B Khalil, Andrea Lodi, Christopher Morris, and Petar Velickovic. Combinatorial optimization and reasoning with graph neural networks. *J. Mach. Learn. Res.*, 24:130–1, 2023. 16
- [12] Constantin Carathéodory. Über den variabilitätsbereich der koeffizienten von potenzreihen, die gegebene werte nicht annehmen. *Mathematische Annalen*, 64(1):95–115, 1907. 2
- [13] Chandra Chekuri and Amit Kumar. Maximum coverage problem with group budget constraints and applications. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 72–83. Springer, 2004. 21
- [14] Wei Chen, Xiaoming Sun, Jialin Zhang, and Zhijie Zhang. Network inference and influence maximization from samples. In *Proceedings of the 38th International Conference on Machine Learning*, pages 1707–1716, 2021. 7
- [15] Yongqiang Chen, Yatao Bian, Bo Han, and James Cheng. Interpretable and generalizable graph learning via subgraph multilinear extension. In *ICLR 2024 Workshop on Machine Learning for Genomics Explorations*, 2024. 3
- [16] Cyrille W Combettes and Sebastian Pokutta. Revisiting the approximate carathéodory problem via the frank-wolfe algorithm. *Mathematical Programming*, 197(1):191–214, 2023. 3
- [17] Priya L Donti, David Rolnick, and J Zico Kolter. Dc3: A learning method for optimization with hard constraints. *arXiv preprint arXiv:2104.12225*, 2021. 3

- [18] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. *arXiv preprint arXiv:2110.07875*, 2021. 39
- [19] Adam N Elmachtoub and Paul Grigas. Smart “predict, then optimize”. *Management Science*, 68(1):9–26, 2022. 16
- [20] Yahya H Ezzeldin, Martina Cardone, Christina Fragouli, and Giuseppe Caire. Polynomial-time capacity calculation and scheduling for half-duplex 1-2-1 networks. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 460–464. IEEE, 2019. 3
- [21] Thomas Frerix, Matthias Nießner, and Daniel Cremers. Homogeneous linear inequality constraints for neural network activations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 748–749, 2020. 3
- [22] Tobias Friedrich, Andreas Göbel, Frank Neumann, Francesco Quinzan, and Ralf Rothenberger. Greedy maximization of functions with bounded curvature under partition matroid constraints. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2272–2279, 2019. 21
- [23] Elīza Gaile, Andis Draguns, Emīls Ozoliņš, and Kārlis Freivalds. Unsupervised training for neural tsp solver. In *International Conference on Learning and Intelligent Optimization*, pages 334–346. Springer, 2022. 3
- [24] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2. Springer Science & Business Media, 2012. 3, 5, 16, 17, 19
- [25] Aditya Grover, Eric Wang, Aaron Zweig, and Stefano Ermon. Stochastic optimization of sorting networks via continuous relaxations. In *International Conference on Learning Representations*, 2018. 3
- [26] Will Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. Embedding logical queries on knowledge graphs. *Advances in neural information processing systems*, 31, 2018. 8, 29
- [27] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018. URL <https://arxiv.org/abs/1706.02216>. 7
- [28] Ruben Hoeksma, Bodo Manthey, and Marc Uetz. Efficient implementation of carathéodory’s theorem for the single machine scheduling polytope. *Discrete applied mathematics*, 215: 136–145, 2016. 3, 5
- [29] Seong-Hyun Hong, Hyun-Sung Kim, Zian Jang, Deunsol Yoon, Hyungseok Song, and Byung-Jun Lee. Unsupervised training of diffusion models for feasible solution generation in neural combinatorial optimization. *arXiv preprint arXiv:2411.00003*, 2024. 3
- [30] Xiuzhen Huang, Jing Lai, and Steven F Jennings. Maximum common subgraph: some upper bound and lower bound results. *BMC bioinformatics*, 7:1–9, 2006. 1
- [31] Jan Hůla, David Mojžíšek, and Mikoláš Janota. Understanding gnns for boolean satisfiability through approximation algorithms. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 953–961, 2024. 3
- [32] Yuma Ichikawa. Controlling continuous relaxation for combinatorial optimization. *Advances in Neural Information Processing Systems*, 37:47189–47216, 2024. 3
- [33] Wei Jin, Tong Zhao, Jiayuan Ding, Yozen Liu, Jiliang Tang, and Neil Shah. Empowering graph representation learning with test-time graph transformation. In *The Eleventh International Conference on Learning Representations*, 2023. 9
- [34] Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. *arXiv preprint arXiv:2006.10643*, 2020. 1, 3, 8, 29

- [35] Nikolaos Karalias, Joshua Robinson, Andreas Loukas, and Stefanie Jegelka. Neural set function extensions: Learning with discrete functions in high dimensions. *Advances in Neural Information Processing Systems*, 35:15338–15352, 2022. 1, 3, 6, 16
- [36] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, page 137–146. Association for Computing Machinery, 2003. 7
- [37] 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*, 36:11823–11849, 2023. 3
- [38] Samir Khuller, Anna Moss, and Joseph Seffi Naor. The budgeted maximum coverage problem. *Information processing letters*, 70(1):39–45, 1999. 7, 21
- [39] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 39
- [40] Gustav Kirchhoff. Ueber die auflösung der gleichungen, auf welche man bei der untersuchung der linearen vertheilung galvanischer ströme geführt wird. *Annalen der Physik*, 148(12): 497–508, 1847. 7, 26
- [41] Till Kletti, Jean-Michel Renders, and Patrick Loiseau. Introducing the expohedron for efficient pareto-optimal fairness-utility amortizations in repeated rankings. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pages 498–507, 2022. 3, 5
- [42] Marcel Kollovich, Bertrand Charpentier, Daniel Zügner, and Stephan Günnemann. Expected probabilistic hierarchies. *Advances in Neural Information Processing Systems*, 37:13818–13850, 2024. 3
- [43] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf. 16
- [44] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf. 8, 29
- [45] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems!, 2019. URL <https://arxiv.org/abs/1803.08475>. 8, 16, 29
- [46] Andreas Krause, Carlos Guestrin, Anupam Gupta, and Jon Kleinberg. Near-optimal sensor placements: Maximizing information while minimizing communication cost. In *Proceedings of the 5th international conference on Information processing in sensor networks*, pages 2–10, 2006. 20
- [47] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21188–21198. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/f231f2107df69eab0a3862d50018a9b2-Paper.pdf. 16
- [48] 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. 3
- [49] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015. 39

- [50] Enming Liang and Minghua Chen. Efficient bisection projection to ensure nn solution feasibility for optimization over general set. **3**
- [51] Enming Liang, Minghua Chen, and Steven Low. Low complexity homeomorphic projection to ensure neural-network solution feasibility for optimization over (non-) convex set. 2023. **3**
- [52] 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. **3**
- [53] Derek Lim, Joshua Robinson, Lingxiao Zhao, Tess Smidt, Suvrit Sra, Haggai Maron, and Stefanie Jegelka. Sign and basis invariant networks for spectral graph representation learning. *arXiv preprint arXiv:2202.13013*, 2022. **3**
- [54] Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 510–520, 2011. **7**
- [55] László Lovász. Semidefinite programs and combinatorial optimization. In *Recent advances in algorithms and combinatorics*, pages 137–194. Springer, 2003. **7, 27, 28**
- [56] Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Snoek. Learning latent permutations with gumbel-sinkhorn networks. In *International Conference on Learning Representations*, 2018. **3**
- [57] 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*, 35:22713–22724, 2022. **3**
- [58] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, et al. Chip placement with deep reinforcement learning. *arXiv preprint arXiv:2004.10746*, 2020. **1**
- [59] Vahab Mirrokni, Renato Paes Leme, Adrian Vladu, and Sam Chiu-wai Wong. Tight bounds for approximate carathéodory and beyond. In *International Conference on Machine Learning*, pages 2440–2448. PMLR, 2017. **3**
- [60] Konrad Munding, Max Zimmer, Aldo Kiem, Christoph Spiegel, and Sebastian Pokutta. Neural discovery in mathematics: Do machines dream of colored planes? *arXiv preprint arXiv:2501.18527*, 2025. **3**
- [61] 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. **7, 29**
- [62] Robert R Nerem, Zhishang Luo, Akbar Rafiey, and Yusu Wang. Differentiable extensions with rounding guarantees for combinatorial optimization over permutations. *arXiv preprint arXiv:2411.10707*, 2024. **3**
- [63] Emils Ozolins, Karlis Freivalds, Andis Draguns, Eliza Gaile, Ronalds Zakovskis, and Sergejs Kozlovics. Goal-aware neural sat solver. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2022. **3**
- [64] Panos M Pardalos and Mauricio GC Resende. Handbook of applied optimization. (*No Title*), 2002. **1**
- [65] Anselm Paulus, Michal Rolínek, Vít Musil, Brandon Amos, and Georg Martius. Comboptnet: Fit the right np-hard problem by learning integer programming constraints. *arXiv preprint arXiv:2105.02343*, 2021. **16**
- [66] Anselm Paulus, Georg Martius, and Vít Musil. Lpgd: A general framework for backpropagation through embedded optimization layers. *arXiv preprint arXiv:2407.05920*, 2024. **16**
- [67] Max Benedikt Paulus, Dami Choi, Daniel Tarlow, Andreas Krause, and Chris J Maddison. Gradient estimation with stochastic softmax tricks. In *NeurIPS 2020*, 2020. **3**

- [68] Marin Vlastelica Pogančič, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolínek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2019. 16
- [69] Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. Dimes: A differentiable meta solver for combinatorial optimization problems. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 25531–25546. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/a3a7387e49f4de290c23beea2dfcdc75-Paper-Conference.pdf. 16
- [70] Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. DIMES: A differentiable meta solver for combinatorial optimization problems. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL <https://openreview.net/forum?id=9u05zrOnhx>. 8, 29
- [71] Fred J Rispoli. The graph of the hypersimplex. *arXiv preprint arXiv:0811.2981*, 2008. 6
- [72] Michael Eli Sander, Joan Puigcerver, Josip Djolonga, Gabriel Peyré, and Mathieu Blondel. Fast, differentiable and sparse top-k: a convex analysis perspective. In *International Conference on Machine Learning*, pages 29919–29936. PMLR, 2023. 3
- [73] Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. A diffusion model framework for unsupervised neural combinatorial optimization. In *Proceedings of the 41st International Conference on Machine Learning*, pages 43346–43367, 2024. 3
- [74] Aditya P Saraf and Gary L Slater. An efficient combinatorial optimization algorithm for optimal scheduling of aircraft arrivals at congested airports. In *2006 IEEE aerospace conference*, pages 11–pp. IEEE, 2006. 1
- [75] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Approximation ratios of graph neural networks for combinatorial problems, 2019. 3
- [76] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998. 17
- [77] 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. 3
- [78] Martin JA Schuetz, J Kyle Brubaker, Zhihuai Zhu, and Helmut G Katzgraber. Graph coloring with physics-inspired graph neural networks. *Physical Review Research*, 4(4):043131, 2022. 3
- [79] Richard Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The annals of mathematical statistics*, 35(2):876–879, 1964. 5
- [80] Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 563–568, 2008. 26
- [81] Haoran Sun, Etash K Guha, and Hanjun Dai. Annealed training for combinatorial optimization on graphs. *arXiv preprint arXiv:2207.11542*, 2022. 3
- [82] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000. 16
- [83] Jan Toenschhoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Graph neural networks for maximum constraint satisfaction. *Frontiers in artificial intelligence*, 3:98, 2021. 1, 3
- [84] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/29921001f2f04bd3baee84a12e98098f-Paper.pdf. 3, 16

- 628 [85] J. Vondrák. Optimal approximation for the submodular welfare problem in the value oracle
629 model. In *Symposium on Theory of Computing (STOC)*, 2008. 20
- 630 [86] Haoyu Peter Wang, Nan Wu, Hang Yang, Cong Hao, and Pan Li. Unsupervised learning
631 for combinatorial optimization with principled objective relaxation. In *Advances in Neural
632 Information Processing Systems*, 2022. 3
- 633 [87] Runzhong Wang, Li Shen, Yiting Chen, Xiaokang Yang, Dacheng Tao, and Junchi Yan. Towards
634 one-shot neural combinatorial solvers: Theoretical and empirical notes on the cardinality-
635 constrained case. In *The Eleventh International Conference on Learning Representations*, 2022.
636 5, 7, 28, 29
- 637 [88] Runzhong Wang, Yunhao Zhang, Ziao Guo, Tianyi Chen, Xiaokang Yang, and Junchi Yan.
638 Linsatnet: the positive linear satisfiability neural networks. In *International Conference on
639 Machine Learning*, pages 36605–36625. PMLR, 2023. 3
- 640 [89] Hao Xu, Ka-Hei Hui, Chi-Wing Fu, and Hao Zhang. Tilingnn: learning to tile with self-
641 supervised graph neural network. *ACM Transactions on Graphics (TOG)*, 39(4):129–1, 2020.
642 1, 3
- 643 [90] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural
644 networks? In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=ryGs6iA5Km>. 3
- 645
- 646 [91] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM
647 SIGKDD international conference on knowledge discovery and data mining*, pages 1365–1374,
648 2015. 39
- 649 [92] Morris Yau, Nikolaos Karalias, Eric Lu, Jessica Xu, and Stefanie Jegelka. Are graph neural
650 networks optimal approximation algorithms? *Advances in Neural Information Processing
651 Systems*, 37:73124–73181, 2024. 3
- 652 [93] Bülent Yener, Yoram Ofek, and Moti Yung. Combinatorial design of congestion-free networks.
653 *IEEE/ACM transactions on networking*, 5(6):989–1000, 1997. 1
- 654 [94] Hongtai Zeng, Chao Yang, Yanzhen Zhou, Cheng Yang, and Qinglai Guo. Glinsat: The
655 general linear satisfiability neural network layer by accelerated gradient descent. *arXiv preprint
656 arXiv:2409.17500*, 2024. 3, 5

657 A Additional discussion of related work

658 Reinforcement learning (RL)-based construction heuristics have been one of the leading paradigms
 659 in neural CO. The first application to use RL with neural networks are [4], where they combined
 660 a Pointer Network [84] with actor-critic reinforcement algorithm [43] to generate solutions for the
 661 Traveling Salesman Problem. Later works including [45], [47], and [69] have gained great success in
 662 solving larger-scale CO problems with RL algorithms, and also showed the ability to generalize to
 663 large scale problem instances. There are similarities between our approach and the general approach
 664 followed in RL algorithms like policy gradient[82] that are worth discussing. In RL, one learns
 665 a distribution that maximizes the expected reward. This is done by sampling solutions from the
 666 outputs of a neural network which are treated as the parameters of a probability distribution, then
 667 calculating their reward and weighing it by their log probability, and finally backpropagating to the
 668 parameters of the neural network. Stochasticity is often at the core of RL and the ways that one
 669 may sample solutions for a given problem differ. Our goal is to propose a structured approach that
 670 relies on methods from polyhedral combinatorics and convex geometry. Concretely, the output of
 671 the neural network is transformed so that it yields a point in the convex hull of feasible solutions
 672 of the problem. Then, instead of sampling, we leverage the decomposition algorithm to explicitly
 673 construct the support of the distribution that is parametrized by the neural net output. The extension
 674 we maximize can be viewed as the expected 'reward' (objective), which is similar to how RL operates.
 675 In terms of differentiability, our approach is slightly different since our extension is a.e. differentiable
 676 and we can directly use automatic differentiation while RL deals with this using techniques like the
 677 log-derivative trick. Importantly, extensions can be used to optimize any black-box objective which
 678 is also the case for RL.

679 Other related lines of work include the Predict then Optimize paradigm [19] which combines learned
 680 predictions with a deterministic optimization problem that is parametrized by them. The mathematical
 681 relationship to extensions originates from the fact that extensions can be viewed as feasible solutions
 682 to a linear program for the convex envelope that is parametrized by some neural net predictions
 683 [35]. Another line of work that combines predictive and algorithmic components is the work on
 684 backpropagating through solvers [68, 65, 66]. For a more complete reference on combinatorial
 685 optimization with (graph) neural networks we refer the reader to [11].

686 B Decomposition Theorem: extended discussion

687 We will provide an extended discussion of the GLS result and its implications for our decomposition
 688 approach. For that we will need some definitions as presented in [24] in order to provide a self-
 689 contained discussion of the theorem, its proof, and its implications for our approach.

690 **Definition B.1** (Defining properties of polyhedra). Let $\mathcal{P} \subseteq \mathbb{R}^n$ be a polyhedron and let φ and ν be
 691 positive integers.

- 692 1. We say that \mathcal{P} has **facet-complexity** at most φ if there exists a system of inequalities
 693 with rational coefficients that has solution set \mathcal{P} and such that the encoding length of each
 694 inequality of the system is at most φ . In case $\mathcal{P} = \mathbb{R}^n$ we require $\varphi \geq n + 1$.
- 695 2. We say that \mathcal{P} has **vertex-complexity** at most ν if there exist finite sets V, E of rational
 696 vectors such that

$$\mathcal{P} = \text{conv}(V) + \text{cone}(E)$$
 697 and such that each of the vectors in V and E has encoding length at most ν . In case $\mathcal{P} = \emptyset$
 698 we require $\nu \geq n$.
- 699 3. A **well-described polyhedron** is a triple $(\mathcal{P}; n, \varphi)$ where $\mathcal{P} \subseteq \mathbb{R}^n$ is a polyhedron with
 700 facet-complexity at most φ . The encoding length $\langle \mathcal{P} \rangle$ of a well-described polyhedron
 701 $(\mathcal{P}; n, \varphi)$ is

$$\langle \mathcal{P} \rangle = \varphi + n.$$

702 □

703 **Definition B.2** (Strong optimization problem). The strong optimization problem for a given rational
 704 vector \mathbf{x} in n dimensions and any well described $(\mathcal{P}; n, \varphi)$ is given by

$$\max \mathbf{c}^\top \mathbf{x}, \quad \mathbf{c} \in \mathcal{P}. \quad (7)$$

705 We may now restate Theorem B.3.

706 **Theorem B.3.** (GLS) *There exists an oracle polynomial-time algorithm that for any well-described*
 707 *polyhedron $(P; n, \varphi)$ given by a strong optimization oracle, for any rational vector \mathbf{x} , finds affinely*
 708 *independent vertices $\mathbf{1}_{S_1}, \mathbf{1}_{S_2}, \dots, \mathbf{1}_{S_k}$ and positive rational numbers $\lambda_1, \lambda_2, \dots, \lambda_k$ such that $\mathbf{x} =$*
 709 *$\sum_{t=1}^k \lambda_t \mathbf{1}_{S_t}$.*

710 *Proof.* Set $\mathbf{x}_0 = \mathbf{x}$. The algorithm begins by finding the smallest face that contains \mathbf{x} and then obtain
 711 a vertex $\mathbf{1}_{S_0}$ of that face. If that vertex is \mathbf{x}_0 then the algorithm terminates. Otherwise, a half-line is
 712 drawn from $\mathbf{1}_{S_1}$ through \mathbf{x}_0 until the boundary of the polytope is intersected. At the intersection point,
 713 we obtain a new iterate \mathbf{x}_1 . Notice that the previous iterate \mathbf{x}_0 can be written as a convex combination
 714 of \mathbf{x}_1 and $\mathbf{1}_{S_1}$ since it is part of a line with endpoints \mathbf{x}_1 and $\mathbf{1}_{S_1}$. Again, we find the smallest face
 715 containing \mathbf{x}_1 and obtain a vertex $\mathbf{1}_{S_2}$ from it. If it matches the iterate we terminate, otherwise we
 716 repeat the process. Since each iterate belongs to the faces of all the previous iterates but not the
 717 subsequent ones, the iterates are affinely independent and hence the process has to terminate in $k \leq n$
 718 steps. Given the starting vector and the vertices it is straightforward to compute the coefficients with
 719 linear algebra. Furthermore the encoding length of each iterate is polynomial since it is the unique
 720 point of intersection of affine subspaces spanned by \mathbf{x} and the vertices of the polytope. \square

721 B.1 Necessary conditions for a valid decomposition

722 As we explained, this proof provides the blueprint for the methodology in this paper. For the proof to
 723 yield a useful decomposition for our setting we have the following prerequisites:

- 724 1. For each iterate, finding a minimal face that contains it and retrieving a vertex from it.
- 725 2. The ability to find intersection points of half-lines with the boundary of the polytope.

726 The first point corresponds to the step 3 of our algorithm. We will see that performing this step
 727 efficiently can be done if we have access to a fast strong optimization oracle.

728 **Obtaining a polytope vertex for Step 3 of Algorithm 1.** The proof requires a vertex of a face
 729 containing the iterate \mathbf{x}_t . The face F containing \mathbf{x}_t is the set of vectors attaining $\max_{\mathbf{c} \in \mathcal{P}} \mathbf{x}^\top \mathbf{c}$ [76].
 730 Given access to the vectors of the face F , we need to be able to retrieve a vertex of that face. This
 731 can be done by a simple technique, where we make a call to the oracle for the perturbed program

$$\max \bar{\mathbf{c}}^\top \mathbf{x}, \quad \mathbf{c} \in \mathcal{P}, \quad (8)$$

732 with $\bar{\mathbf{c}} = \mathbf{c} + [\epsilon, \epsilon^2, \dots, \epsilon^n]^\top$ for some small epsilon [24][Remark 6.5.2].

733 **Obtaining the coefficient for Step 4 of Algorithm 1.** The second prerequisite is the ability to
 734 compute intersection points of the boundary of the polytope with a half-line that passes through the
 735 current iterate. This is related to step 4 of Algorithm 1. Namely, step 4 will determine the coefficients
 736 of the convex combination that express the current iterate as a function of the current vertex and the
 737 next iterate. To obtain some general conditions when this is possible, recall that for a general polytope

$$\mathcal{P} = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{z}_i^\top \mathbf{x} \leq b_i, i = 1, \dots, m\},$$

738 In the GLS proof, the coefficient computation is resolved after all the relevant vertices of the
 739 combination have been obtained. In our approach, we pick the convex combination coefficient as

$$a_t^{\max} = \max\{a_t \in [0, 1) : \mathbf{x}_{t+1}(a_t) \in \mathcal{P}\},$$

740 This choice of coefficient leads to the next iterate being as far as possible from the current one. This
 741 means the coefficient will be pushed until some constraints of the polytope are tight for the new
 742 iterate and hence the boundary will be intersected. At iteration t , with current iterate \mathbf{x}_t and chosen
 743 corner $\mathbf{1}_{S_t}$, we may determine the coefficient by enforcing each face inequality. In particular:

$$\begin{aligned} \mathbf{z}_i^\top \mathbf{x}_{t+1} \leq b_i &\iff \mathbf{z}_i^\top \frac{\mathbf{x}_t - a_t \mathbf{1}_{S_t}}{1 - a_t} \leq b_i \\ &\iff a_t \leq \frac{b_i - \mathbf{z}_i^\top \mathbf{x}_t}{\mathbf{z}_i^\top (\mathbf{1}_{S_t} - \mathbf{x}_t)}, \quad \text{whenever } \mathbf{z}_i^\top (\mathbf{1}_{S_t} - \mathbf{x}_t) > 0. \end{aligned}$$

744 Hence

$$a_t^{\max}(\mathbf{x}_t) = \min_{i: \mathbf{z}_i^\top (\mathbf{1}_{S_t} - \mathbf{x}_t) > 0} \frac{b_i - \mathbf{z}_i^\top \mathbf{x}_t}{\mathbf{z}_i^\top (\mathbf{1}_{S_t} - \mathbf{x}_t)}, \quad (9)$$

745 which shows that the maximum feasible weight a_t is a well-defined differentiable function of the
 746 current iterate \mathbf{x}_t . This is crucial because it allows us to backpropagate through the coefficient and
 747 leverage the decomposition in gradient-based optimization. Note that we can make such an argument
 748 assumming we have a description of the polytope in terms of inequalities.

749 Equation (9) and Equation (8) lead us to sufficient conditions for our decomposition. Specifically,
 750 given access to a fast algorithm for Equation (9) and a fast algorithm for Equation (8) we can obtain
 751 a tractable decomposition that allows us to build $\mathcal{D}_C(\mathbf{x})$ in a differentiable manner. An important
 752 observation is that both of those conditions will depend on the polytope. Indeed, for matroid
 753 polytopes strong optimization oracles are available. A fast algorithm for Equation (8) also requires a
 754 compact description of \mathcal{P} in terms of inequalities. If the polytope is defined by exponentially many
 755 inequalities, computing the minimum becomes intractable. These can be challenging issues when
 756 using our framework for constraints associated with NP-Hard problems like independent sets but we
 757 will discuss how they can be mitigated.

758 C Decomposition with controlled approximation

759 One of the limitations of the decomposition as presented in the approach is that the support of the
 760 distribution we obtain depends exactly on the ambient dimension of the space. There are a few
 761 reasons why this might not be desirable. First, an exact decomposition like this fixes at most n sets in
 762 the support. That may be too restrictive in problems with 'sparse rewards', i.e., when good objective
 763 values are achieved on only a few points. It would be preferable if we had more control over the
 764 number of sets in the support of the distribution, which could in turn lead to better outcomes for
 765 optimization. We show how we can achieve this with a simple tweak that allows us to control the
 766 reconstruction quality, which we also utilize in our max coverage experiments.

767 C.1 Decomposition via rescaling

768 Start with $\mathbf{x}_0 = \mathbf{x}$. For each iteration $t = 0, 1, 2, \dots, k$, we first pick the corner $\mathbf{1}_{S_t}$ and compute the
 769 iteration coefficient

$$\tilde{a}_t = \min \left\{ \min_{i \in S_t} x_t(i), \min_{j \notin S_t} (1 - x_t(j)) \right\}.$$

770 The difference now is that instead of picking the maximum coefficient we will rescale the \tilde{a}_t in each
 771 step by some fixed constant $b \in (0, 1]$. This means that the next iterate will not be intersecting the
 772 boundary and hence the algorithm will require more steps to terminate, hence requiring a tolerance
 773 parameter ϵ as we have described in Algorithm 1. Additionally, we fix some lower bound ℓ on the
 774 rescaling so that

$$a_t = \begin{cases} b \tilde{a}_t, & b \tilde{a}_t \geq \ell, \\ \tilde{a}_t, & b \tilde{a}_t < \ell. \end{cases} \quad (10)$$

775 Using this coefficient we compute the new iterate using Equation (3).

776 **Proposition C.1.** *Suppose we pick step 3 in Algorithm 1 according to Equation (10). Then the*
 777 *reconstruction error of the algorithm (step 8 in Algorithm 1) decays exponentially in k .*

778 *Proof.* After k iterations we may express \mathbf{x} as

$$\mathbf{x} = \sum_{t=1}^k \underbrace{\left(\prod_{i=1}^{t-1} (1 - a_i) \right)}_{p_{\mathbf{x}_t}(S_t)} a_t \mathbf{1}_{S_t} + \left(\prod_{i=1}^k (1 - a_i) \right) \mathbf{x}_{k+1}.$$

779 Recall from step 8 of Algorithm 1 we have that the stopping criterion (given by the reconstruction
780 error) is

$$\|\mathbf{x} - \sum_{t=1}^k p_{\mathbf{x}_t(S_t)} \mathbf{1}_{S_t}\| \leq \epsilon.$$

781 Clearly, the term inside the norm on the left hand side of the inequality is the residual

$$\mathbf{r}_{k+1} = \left(\prod_{i=1}^k (1 - a_i) \right) \mathbf{x}_{k+1}.$$

782 For the reconstruction error we just need to compute the norm of the residual and show that it decays
783 exponentially in k . Hence, we calculate

$$\begin{aligned} \|\mathbf{r}_{k+1}\|_2 &= \left(\prod_{i=1}^k (1 - a_i) \right) \|\mathbf{x}_{k+1}\|_2 \\ &\leq (1 - \ell)^k \|\mathbf{x}\|_2 \\ &\leq (1 - \ell)^k n, \end{aligned}$$

784 which completes the proof. □

785 Given some tolerance parameter ϵ , this allows us to control the number of sets considered in the
786 iteration. We observed that using this approach helped with the training dynamics of the model as
787 well as with performance.

788 D Deferred Proofs from Section 4.2

789 Proof of Theorem 4.2

790 *Proof.* The proof follows the same argument as the proof in [24][Theorem 6.5.11]. To make the
791 result more intuitive we may assume without loss of generality that the entries of \mathbf{x}_t are sorted, i.e.
792 $\mathbf{x}_t(1) \geq \mathbf{x}_t(2) \geq \dots \geq \mathbf{x}_t(n)$. Under this ordering,

$$\mathbf{1}_{S_t} = \arg \max_{\substack{\mathbf{y} \in \{0,1\}^n \\ \|\mathbf{y}\|_1 = k}} \mathbf{x}_t^\top \mathbf{y},$$

793 selects exactly the top k coordinates. Since $\|\mathbf{x}_t\|_1 = k$ and $\|\mathbf{1}_{S_t}\|_1 = k$, taking the ℓ_1 -norm of both
794 sides of Equation (3) gives

$$k = a_t k + (1 - a_t) \|\mathbf{x}_{t+1}\|_1,$$

795 so $\|\mathbf{x}_{t+1}\|_1 = k$ and the sum-to- k constraint is preserved exactly. Apart from the norm constraint,
796 we also have to ensure that each coordinate remains in the hypercube. To keep each coordinate in
797 the hypercube we solve for the coefficient a_t that satisfies $0 \leq \mathbf{x}_{t+1} \leq 1$ in our recurrence equation.
798 First, we rearrange Equation (3)

$$\mathbf{x}_{t+1} = \frac{\mathbf{x}_t - a_t \mathbf{1}_{S_t}}{1 - a_t}$$

799 and note that subtraction at the numerator affects only the top k coordinates. It is clear that for $i \in S_t$
800 the coordinates cannot possibly exceed one because the denominator is always larger, but they may
801 drop below 0. So we need to ensure

$$\frac{\mathbf{x}_t(i) - a_t}{1 - a_t} \geq 0, \tag{11}$$

802 which implies $a_t \leq \min_{i \in S_t} \mathbf{x}_t(i)$ and corresponds to a constraint on the k -th coordinate in this
803 ordering. On the other hand, for $j \notin S_t$ they can only increase in magnitude, so they may exceed 1,
804 which leads us to

$$\frac{\mathbf{x}_t(j)}{1 - a_t} \leq 1, \tag{12}$$

and hence $a_t \leq 1 - \max_{j \notin S_t} \mathbf{x}_t(j)$ which constrains the $(k+1)$ -th coordinate. Ensuring that both Equation (11) and Equation (12) are satisfied, and following the GLS proof, we pick the largest possible coefficient which leads us to Equation (6):

$$a_t = \min\left\{\min_{i \in S_t} \mathbf{x}_t(i), 1 - \max_{j \notin S_t} \mathbf{x}_t(j)\right\}.$$

If $a_t = \min_{i \in S_t} \mathbf{x}_t(i)$, the k -th entry is set to zero; otherwise the $(k+1)$ -th coordinate is set to 1. Note that from Equation (3), once a coordinate has been fixed to either 0 or 1, it cannot change in subsequent iterations. Therefore, the process will terminate when the next iterate becomes a corner of $\Delta_{n,k}$. Since there are k zeros and $n-k$ ones at the final iterate and each iteration fixes exactly one coordinate, the process terminates after at most $k + (n-k) = n$ iterations. In line with the GLS proof, the next iterate intersects the boundary of the polytope at a face of lower dimension because the minimum is guaranteed to fix a coordinate either to 0 or to 1.

Finally, because each a_t is defined via pointwise minima and maxima of the entries of \mathbf{x}_t , it is almost everywhere differentiable in \mathbf{x}_t . By the chain rule, the resulting probability weights

$$p_{\mathbf{x}_t}(S_t) = a_t \prod_{i < t} (1 - a_i)$$

are almost-everywhere differentiable in the original input \mathbf{x}_0 . This completes the proof. \square

Proof of Proposition 4.3

Proof. Let $\mathbf{z} \in [0, 1]^n$ and $\mathbf{x} = s\tilde{\mathbf{z}} + \mathbf{u}$, and define

$$\mu = \frac{1}{n} \sum_{i=1}^n z_i, \quad \tilde{\mathbf{z}} = \mathbf{z} - \mu \mathbf{1}, \quad \mathbf{u} = \frac{k}{n} \mathbf{1}.$$

Since $\sum_i \tilde{z}_i = \sum_i (z_i - \mu) = 0$, we have

$$\sum_{i=1}^n x_i = s \sum_{i=1}^n \tilde{z}_i + \sum_{i=1}^n u_i = 0 + k,$$

so $\|\mathbf{x}\|_1 = k$. Next, for each coordinate

$$x_i = s(z_i - \mu) + \frac{k}{n}.$$

Because $0 \leq z_i \leq 1$, we have $-\mu \leq z_i - \mu \leq 1 - \mu$. Therefore

$$x_i \geq -s\mu + \frac{k}{n} = \frac{k}{n} \left(1 - s \frac{n\mu}{k}\right) \geq 0$$

by $s \leq \frac{k}{n\mu}$, and

$$x_i \leq s(1 - \mu) + \frac{k}{n} = 1 - \frac{n-k}{n} \left(1 - s \frac{n(1-\mu)}{n-k}\right) \leq 1$$

by $s \leq \frac{n-k}{n(1-\mu)}$. Hence $0 \leq x_i \leq 1$ for all i , and we conclude $\mathbf{x} \in \Delta_{n,k}$. \square

E Case study: Partition Matroids and Spanning Trees

E.1 Partition Matroid

Partition matroids provide a powerful and flexible framework for modeling constraints in set function optimization problems. They capture scenarios where elements are divided into categories, and we are allowed to select only a limited number from each category — a structure that arises naturally in many real-world applications. This structure naturally arises in applications such as sensor placement, where sensors are partitioned by geographic regions and each region has an independent budget [46]; job allocation and welfare maximization, where tasks are categorized by required skills and only a bounded number of tasks from each skill category can be assigned [10, 85]; and many other

834 theoretical and practical settings [22, 13, 38]. Partition matroids thus provide a flexible abstraction
 835 for modeling structured selection problems with heterogeneous constraints.

836 A partition matroid or partitional matroid is a matroid that is a direct sum of uniform matroids. It
 837 is defined over a ground set in which the elements are partitioned into different categories. For
 838 each category, there is a cardinality constraint—a maximum number of allowed elements from this
 839 category. Formally, let V_1, \dots, V_c be disjoint subsets such that $V = \cup_{i=1}^d V_i$. Let k_i be integers with
 840 $0 \leq k_i \leq |V_i|$. Consider the following combinatorial optimization problem:

$$\max_{S: |S \cap V_i| = k_i} f(S). \quad (13)$$

841 Note that when $c = 1$, this reduces to the cardinality case. The convex polytope of feasible solutions
 842 is known as the partition matroid base polytope and is described as follows:

$$\mathcal{B} = \left\{ \mathbf{x} : \forall S \subseteq E \sum_{e \in S} \mathbf{x}(e) \leq r(S), \text{ and } \|\mathbf{x}\|_1 = \sum_{i=1}^c k_i \right\} \quad (14)$$

$$r(S) = \sum_{i=1}^c \min(|S \cap V_i|, k_i) \quad \forall S \subseteq E. \quad (15)$$

843 The vertices of the matroid base polytope \mathcal{B} are the indicator vectors of sets whose intersection with
 844 every block V_i has size exactly k_i . Hence, any vector $\mathbf{x} \in \mathcal{B}$ can be written as a convex combination of
 845 the indicator vectors of feasible sets. We prove that Algorithm 2, given $\mathbf{x} \in \mathcal{B}$, returns in polynomial
 846 time a distribution over feasible sets which is continuous and differentiable with respect to \mathbf{x} . Note
 847 that Algorithm 2 is an explicit version of our generic Algorithm 1, with steps 3
 848 and 4 specified.

849 **Theorem E.1.** *Given $\mathbf{x} \in \mathcal{B}$, Algorithm 2 terminates after at most $O(n)$ iterations and returns*
 850 *$\{(p_{\mathbf{x}_t}(S_t), S_t)\}$ such that for every S_t and V_i we have $|S_t \cap V_i| = k_i$, and $\sum_t p_{\mathbf{x}_t}(S_t) = 1$ with*
 851 *$0 \leq p_{\mathbf{x}_t}(S_t) \leq 1$. Moreover, each $p_{\mathbf{x}_t}(S_t)$ is a continuous and differentiable function of \mathbf{x} .*

852 *Proof of Theorem E.1.* Suppose $\mathbf{x}_0 = \mathbf{x} \in \mathcal{B}$. We prove the correctness of our algorithm for every
 853 iteration t . To obtain a vertex of the polytope in the support of \mathbf{x}_t , we set

$$\mathbf{1}_{S_t} = \underset{\mathbf{y} \in \{0,1\}^n, \|\mathbf{y}(V_i)\|_1 = k_i}{\operatorname{argmax}} \quad \mathbf{x}_t^\top \mathbf{y}$$

854 The optimal solution for the above problem is $S_t = \cup_{i=1}^c T_i$ where for each block V_i , T_i is the indices
 855 of the top k_i coordinates within $\mathbf{x}_t(V_i)$. Define

$$a_t = \min \left\{ \min_{i \in S_t} \mathbf{x}_t(i), 1 - \max_{i \notin S_t} \mathbf{x}_t(i) \right\}, \text{ and} \quad (16)$$

$$\mathbf{x}_{t+1} = \frac{1}{1-a_t} (\mathbf{x}_t - a_t \mathbf{1}_{S_t}) \quad (17)$$

856 For the correctness of Algorithm 2 it suffices to show \mathbf{x}_{t+1} is in fact inside the matroid base polytope
 857 \mathcal{B} .

858 First, note that $\mathbf{x}_{t+1} \in [0, 1]^n$ by the definition of a_t and the fact that $\mathbf{x}_t \in [0, 1]^n$. Second, for both \mathbf{x}_t
 859 and $\mathbf{1}_{S_t}$, we have $\|\mathbf{x}\|_1 = \|\mathbf{1}_{S_t}\|_1 = \sum_{i=1}^c k_i$. Hence, by the definition $\mathbf{x}_t = a_t \mathbf{1}_{S_t} + (1 - a_t) \mathbf{x}_{t+1}$,
 860 it follows that $\|\mathbf{x}_{t+1}\|_1 = \sum_{i=1}^c k_i$. It remains to show for all $S \subseteq V$ it holds that $\sum_{e \in S} \mathbf{x}_{t+1}(e) \leq$
 861 $r(S)$. Recall that $r(S) = \sum_{i=1}^c \min(|S \cap V_i|, k_i)$ and

$$\sum_{e \in S} \mathbf{x}_{t+1}(e) = \sum_{e \in S \cap V_1} \mathbf{x}_{t+1}(e) + \dots + \sum_{e \in S \cap V_c} \mathbf{x}_{t+1}(e) \quad (18)$$

862 We show each term $\sum_{e \in S \cap V_i} \mathbf{x}_{t+1}(e) \leq \min(|S \cap V_i|, k_i)$. First suppose $\min(|S \cap V_i|, k_i) = |S \cap V_i|$.
 863 Then since $\mathbf{x}_{t+1}(e) \in [0, 1]$ we have $\sum_{e \in S \cap V_i} \mathbf{x}_{t+1}(e) \leq \sum_{e \in S \cap V_i} 1 = |S \cap V_i|$. Second, suppose
 864 $\min(|S \cap V_i|, k_i) = k_i$. Note $\sum_{e \in S \cap V_i} \mathbf{x}_{t+1}(e) \leq \sum_{e \in V_i} \mathbf{x}_{t+1}(e)$, hence it suffices to show
 865 $\sum_{e \in V_i} \mathbf{x}_{t+1}(e) \leq k_i$.

866 In this case we use the induction hypothesis that $\mathbf{x}_t \in \mathcal{B}$. \mathbf{x}_t being in \mathcal{B} implies that $\sum_{e \in E_i} \mathbf{x}(e) \leq$
867 $r(V_i) = k_i$. By definition we have

$$a_t \sum_{e \in S \cap V_i} \mathbf{1}_{S_t}(e) + (1 - a_t) \sum_{e \in S \cap V_i} \mathbf{x}_{t+1}(e) = \sum_{e \in S \cap V_i} \mathbf{x}_t(e) \leq k_i \quad (19)$$

$$\Rightarrow a_t |S_t \cap V_i| + (1 - a_t) \sum_{e \in S \cap V_i} \mathbf{x}_{t+1}(e) \leq k_i \quad (20)$$

$$\Rightarrow a_t k_i + (1 - a_t) \sum_{e \in S \cap V_i} \mathbf{x}_{t+1}(e) \leq k_i \quad (|S_t \cap V_i| = k_i \text{ by the choice of } S_t)$$

$$\Rightarrow (1 - a_t) \sum_{e \in S \cap V_i} \mathbf{x}_{t+1}(e) \leq (1 - a_t) k_i \quad (21)$$

$$\Rightarrow \sum_{e \in S \cap V_i} \mathbf{x}_{t+1}(e) \leq k_i \quad (22)$$

868 This finishes the proof that $\sum_{e \in S} \mathbf{x}_{t+1}(e) \leq r(S)$, thus $\mathbf{x}_{t+1} \in \mathcal{B}$.

869 Algorithm 2 terminates in at most $O(n)$ iterations. At each iteration t , \mathbf{x}_{t+1} differs from \mathbf{x}_t by fixing
870 one additional coordinate to either 0 or 1. Once a coordinate is fixed to 0/1, it remains unchanged in
871 all future iterates. The process terminates when \mathbf{x}_t has $n - \sum_{i=1}^c k_i$ zeros and $\sum_{i=1}^c k_i$ ones, meaning
872 all coordinates are 0/1.

873 Finally, because each a_t is defined via pointwise minima and maxima of the entries of \mathbf{x}_t , it is almost
874 everywhere differentiable in \mathbf{x}_t . By the chain rule, the resulting probability weights

$$p_{\mathbf{x}_t}(S_t) = a_t \prod_{i < t} (1 - a_i)$$

875 are almost-everywhere differentiable in the original input \mathbf{x}_0 . This completes the proof.

876 □

Algorithm 2 Decomposition for partition matroid

Require: $\mathbf{x} \in \mathcal{B}$.

1: $\mathbf{x}_0 \leftarrow \mathbf{x}$

2: **repeat**

3: $\mathbf{1}_{S_t} = \operatorname{argmax}_{\mathbf{y} \in \{0,1\}^n, \|\mathbf{y}(V_i)\|_1 = k_i} \mathbf{x}_t^\top \mathbf{y}$

4: $a_t = \min \{ \min_{i \in S_t} \mathbf{x}(i), 1 - \max_{i \notin S_t} \mathbf{x}(i) \}$

5: $\mathbf{x}_{t+1} \leftarrow (\mathbf{x}_t - a_t \mathbf{1}_{S_t}) / (1 - a_t)$

6: $p_{\mathbf{x}_t}(S_t) \leftarrow a_t \prod_{i=0}^{t-1} (1 - a_i)$

7: $t \leftarrow t + 1$

8: **until** $\mathbf{x}_t \in \{0,1\}^n$

9: **return** All $\{(p_{\mathbf{x}_t}(S_t), S_t)\}$ pairs.

877 E.1.1 Generating Neural Predictions in the Partition Matroid Base Polytope

878 Similar to the cardinality constraint case, we need to massage the output of neural network in a
879 differentiable and computationally efficient way so that it can be passed to Algorithm 2. The general
880 idea is similar to the one we proposed in Section 4.2.1, however it requires a more careful way of
881 dealing with blocks.

882 **Proposition E.2** (Block-wise scaling into a partitioned simplex). *Let the index set $[n] = \{1, \dots, n\}$*
883 *be partitioned into disjoint blocks V_1, \dots, V_c with $|V_i| = n_i$ and $\sum_{i=1}^c n_i = n$. Given any vector*
884 *$\mathbf{z} \in [0, 1]^n$, set*

$$m_i := \frac{1}{n_i} \sum_{e \in V_i} \mathbf{z}(e), \quad \mathbf{m}(e) := m_i \ (e \in V_i), \quad \mathbf{u}(e) := \frac{k_i}{n_i} \ (e \in V_i),$$

885

$$s_i := \min\left(\frac{k_i/n_i}{m_i}, \frac{(n_i-k_i)/n_i}{1-m_i}\right), \quad \mathbf{s}(e) := s_i \ (e \in V_i),$$

886 and define

$$\mathbf{x} := \mathbf{s} \odot (\mathbf{z} - \mathbf{m}) + \mathbf{u}.$$

887 Then $\mathbf{x} \in \mathcal{B}$; that is, every coordinate of \mathbf{x} lies in $[0, 1]$ and each block V_i has the prescribed sum k_i .888 *Proof.* For each block V_i and every $e \in V_i$ we have

$$\mathbf{x}(e) = s_i(\mathbf{z}(e) - m_i) + \frac{k_i}{n_i}.$$

889 Since $\sum_{e \in V_i} (\mathbf{z}(e) - m_i) = 0$, summing over V_i gives

$$\sum_{e \in V_i} \mathbf{x}(e) = k_i.$$

890 Because $-m_i \leq \mathbf{z}(e) - m_i \leq 1 - m_i$,

$$\mathbf{x}(e) \geq \frac{k_i}{n_i} - s_i m_i, \quad \mathbf{x}(e) \leq \frac{k_i}{n_i} + s_i(1 - m_i).$$

891 By the definition $s_i = \min(\frac{k_i/n_i}{m_i}, \frac{(n_i-k_i)/n_i}{1-m_i})$ both right-hand sides lie in $[0, 1]$, so $0 \leq \mathbf{x}(e) \leq 1$.892 Thus every block sums to k_i and every coordinate is in $[0, 1]$; hence $\mathbf{x} \in \mathcal{B}$. \square 893 **E.2 Graphical Matroid**

894 In this subsection we turn our attention to *graphical matroids* and show that our framework can be
 895 extended beyond cardinality constraints. Graphical matroids are core combinatorial objectives that
 896 have been studied for decades in combinatorial optimization. Some of the most basic combinatorial
 897 problems such as finding a minimum spanning tree in a connected graph to more complex problems
 898 like "does a graph G contain k disjoint spanning trees?" which appears in many applications in
 899 Network Theory.

900 Let $G = (V, E)$ be a graph with n nodes and m edges. The type of optimization problems that we
 901 consider here are expressed as follows

$$\max_{S \subseteq E: S \in \mathcal{F}} f(S) \tag{23}$$

902 where \mathcal{F} denotes the set of full *spanning forests* of G . We focus on the convex hull formed by the
 903 feasible sets edges in \mathcal{F} . That is, let $\mathbf{1}_T$ be the indicator vector of $T \in \mathcal{F}$ and define

$$\mathcal{B}(G) = \text{conv}\{\mathbf{1}_T : T \in \mathcal{F}\} \tag{24}$$

904 The combinatorial object $\mathcal{B}(G)$ is known as the *graphical matroid base polytope*. For any set of edges
 905 let $\text{rank}(S) = n - c(S)$ where $c(S)$ is the number of connected components of the subgraphs formed
 906 by the edges in S . For instance, if G is connected and S forms a spanning tree then $\text{rank}(S) = n - 1$.
 907 It is known that $\mathcal{B}(G)$ can be defined in an equivalent way as follows

$$\mathcal{B}(G) = \{\mathbf{x} \in [0, 1]^m : \forall S \subseteq E, \sum_{e \in S} \mathbf{x}(e) \leq \text{rank}(S), \text{ and } \|\mathbf{x}\|_1 = n - c(V)\} \tag{25}$$

908 Note that any vector $\mathbf{x} \in \mathcal{B}(G)$ can be written as a convex combination of the indicator vectors of
 909 feasible sets i.e, indicator vectors of spanning forests. We prove that Algorithm 3, given $\mathbf{x} \in \mathcal{B}(G)$,
 910 returns in polynomial time a distribution over feasible sets \mathcal{F} which is continuous and differentiable
 911 with respect to \mathbf{x} . The algorithm is essentially the same as what we have seen for the cardinality case,
 912 the only difference is on selecting a proper extreme vertex of the polytope in each step.

913 **Theorem E.3.** *Given $\mathbf{x} \in \mathcal{B}(G)$, Algorithm 3 terminates after at most $O(m)$ iterations and returns*
 914 *$\{(p_{\mathbf{x}_t}(S_t), S_t)\}$ such every S_t corresponds to a spanning forest in G and $\sum_t p_{\mathbf{x}_t}(S_t) = 1$ with*
 915 *$p_{\mathbf{x}_t}(S_t) \in [0, 1]$. Moreover, each $p_{\mathbf{x}_t}(S_t)$ is a continuous and differentiable function of \mathbf{x} .*

Algorithm 3 Decomposition for graphical matroid

Require: Graph $G = (V, E)$ and $\mathbf{x} \in \mathcal{B}(G)$.

```

1:  $\mathbf{x}_0 \leftarrow \mathbf{x}$ 
2: repeat
3:    $S_t \leftarrow$  A maximum spanning forest using  $\mathbf{x}_t$  as the weights for edges (treat zero entries as
      non-edges.)
4:    $a_t = \min \{ \min_{i \in S_t} \mathbf{x}(i), 1 - \max_{i \notin S_t} \mathbf{x}(i) \}$ 
5:    $\mathbf{x}_{t+1} \leftarrow (\mathbf{x}_t - a_t \mathbf{1}_{S_t}) / (1 - a_t)$ 
6:    $p_{\mathbf{x}_t}(S_t) \leftarrow a_t \prod_{i=0}^{t-1} (1 - a_i)$ 
7:    $t \leftarrow t + 1$ 
8: until  $\mathbf{x}_t \in \{0, 1\}^n$ 
9: return All  $\{(p_{\mathbf{x}_t}(S_t), S_t)\}$  pairs.

```

916 *Proof.* Without loss of generality and better exposure, we assume G is a connected graph. All of
 917 the arguments here can be extended to graphs that are not connected in a straightforward way. We
 918 hence continue by assuming G is connected thus $\|\mathbf{x}\|_1 = n - c(V) = n - 1$ for $\mathbf{x} \in \mathcal{B}(G)$. For the
 919 correctness of algorithm it suffices to prove the correctness of one iteration of Algorithm 3. We focus
 920 on one iteration.

921 Let S_t be the maximum spanning tree returned by the Kruskal's algorithm using \mathbf{x}_t as the weights for
 922 edges (treat zero entries as non-edges.)¹. Note that $\mathbf{1}_{S_t}$ is in fact an extreme vertex of the polytope
 923 $\mathcal{B}(G)$. Define

$$a_t = \min \left\{ \min_{e \in S_t} \mathbf{x}_t(e), 1 - \max_{e \notin S_t} \mathbf{x}_t(e) \right\}, \text{ and} \quad (26)$$

$$\mathbf{x}_{t+1} = \frac{1}{1-a_t} (\mathbf{x}_t - a_t \mathbf{1}_{S_t}) \quad (27)$$

924 For the correctness of Algorithm 3 it suffices to show \mathbf{y} is in fact inside the matroid base polytope
 925 $\mathcal{B}(G)$.

926 First, note that $\mathbf{x}_{t+1} \in [0, 1]^n$ by the definition of a_t and the fact that $\mathbf{x}_t \in [0, 1]^n$. Second, for both
 927 \mathbf{x}_t and $\mathbf{1}_{S_t}$, we have $\|\mathbf{x}\|_1 = \|\mathbf{1}_{S_t}\|_1 = n - 1$ since S_t is a spanning tree and must have $n - 1$ edges.
 928 Hence, by the convex combination $\mathbf{x}_t = a_t \mathbf{1}_{S_t} + (1 - a_t) \mathbf{x}_{t+1}$, it follows that $\|\mathbf{x}_{t+1}\|_1 = n - 1$. It
 929 remains to show for all $S \subseteq E$ it holds that $\sum_{e \in S} \mathbf{x}_{t+1}(e) \leq \text{rank}(S)$. Recall that $r(S) = n - c(S)$
 930 where $c(S)$ is the number of connected components in the graph induced by the edges in S . We
 931 consider the following cases and prove for each case $\sum_{e \in S} \mathbf{x}_{t+1}(e) \leq \text{rank}(S)$.

932 **Case 1: S is a spanning tree.** In this case $\text{rank}(S) = n - 1$ since G is connected and $|S| = n - 1$
 933 since S is a spanning tree. Therefore, $\sum_{e \in S} \mathbf{x}_{t+1}(e) \leq |S| = n - 1 = \text{rank}(S)$.

934 **Case 2: S induces a forest.** In this case every connected component of the graph induced by S is a
 935 tree. We count an isolated vertex as a tree. Let $c_1, \dots, c_k, k = C(S)$ be the connected components
 936 induced by S where each of these components is a tree, namely, S_1, \dots, S_k . Note that $S_i \cap S_j = \emptyset$
 937 for $i \neq j$ and the number of nodes in each connected component is $n_i = |S_i| + 1$.

$$\sum_{e \in S} \mathbf{x}_{t+1}(e) = \sum_{e \in S_1} \mathbf{x}_{t+1}(e) + \dots + \sum_{e \in S_k} \mathbf{x}_{t+1}(e) \quad (28)$$

$$\leq |S_1| + \dots + |S_k| \quad (29)$$

$$= (n_1 - 1) + \dots + (n_k - 1) = n - c(S) = \text{rank}(S). \quad (30)$$

938 **Case 3: S induces one connected component which is not a tree.** In this case $\text{rank}(S) = n - 1$,
 939 however $|S| > n - 1$ since S does not induce a tree. Recall that every $0 \leq \mathbf{x}_{t+1}(e) \leq 1$. Hence,
 940 $\sum_{e \in S} \mathbf{x}_{t+1}(e) \leq \sum_{e \in E} \mathbf{x}_{t+1}(e) \leq n - 1 = \text{rank}(S)$.

¹Kruskal's algorithm is generally used for finding minimum spanning tree, however a slight modification can give us a maximum spanning tree.

941 **Case 4: S induces several connected components some of which are tree and some are not.**
 942 This is a mix of Case 2 and Case 3. Let T_1, \dots, T_{k_1} be the connected components induced by S
 943 that are tree (isolated vertices are considered as trees); let S_1, \dots, S_{k_2} be the induced connected
 944 components that are not tree. Note that $\text{rank}(S) = n - k_1 - k_2$.

945 Let let $n(T_i)$ denotes the number of nodes in the connected subgraph T_i i.e., $n(T_i) = |T_i| + 1$. Using
 946 the same argument as Case 2 we have

$$\sum_{e \in T_1} \mathbf{x}_{t+1}(e) + \dots + \sum_{e \in T_{k_1}} \mathbf{x}_{t+1}(e) \leq |T_1| + \dots + |T_{k_1}| \quad (31)$$

$$= (n(T_1) - 1) + \dots + (n(T_{k_1}) - 1) = \sum_{i=1}^{k_1} n(T_i) - k_1 \quad (32)$$

947 Moreover, let $n(S_i)$ denotes the number of nodes in the connected subgraph S_i .

$$\sum_{e \in S_1} \mathbf{x}_t(e) + \dots + \sum_{e \in S_{k_2}} \mathbf{x}_t(e) = \sum_{e \in I \cap (\cup_{i=1}^{k_2} S_i)} \mathbf{x}_t(e) + \sum_{e \notin I \cap (\cup_{i=1}^{k_2} S_i)} \mathbf{x}_t(e) \quad (33)$$

$$\leq |S_t \cap (\cup_{i=1}^{k_2} S_i)| + (1 - a_t) \left(\sum_{i=1}^{k_2} |S_i| - |S_t \cap (\cup_{i=1}^{k_2} S_i)| \right) \quad (34)$$

$$= |S_t \cap (\cup_{i=1}^{k_2} S_i)| + (1 - a_t) \sum_{i=1}^{k_2} |S_i| - (1 - a_t) \left(|S_t \cap (\cup_{i=1}^{k_2} S_i)| \right) \quad (35)$$

$$= a_t |S_t \cap (\cup_{i=1}^{k_2} S_i)| + (1 - a_t) \sum_{i=1}^{k_2} |S_i| \quad (36)$$

948 where the last inequality is because $\mathbf{x}_t(e) \leq 1 - a_t$ for $e \notin S_t \cap (\cup_{i=1}^{k_2} S_i)$. By definition

$$\sum_{e \in S_1} \mathbf{x}_{t+1}(e) + \dots + \sum_{e \in S_{k_2}} \mathbf{x}_{t+1}(e) = \frac{1}{1 - a_t} \left(\sum_{e \in S_1} \mathbf{x}_t(e) + \dots + \sum_{e \in S_{k_2}} \mathbf{x}_t(e) - a_t |S_t \cap (\cup_{i=1}^{k_2} S_i)| \right) \quad (37)$$

$$\leq \frac{1}{1 - a_t} \left(a_t |S_t \cap (\cup_{i=1}^{k_2} S_i)| + (1 - a_t) \sum_{i=1}^{k_2} |S_i| - a_t |S_t \cap (\cup_{i=1}^{k_2} S_i)| \right) \quad (38)$$

$$= \sum_{i=1}^{k_2} |S_i| = \sum_{i=1}^{k_2} n(S_i) - k_2 \quad (39)$$

949 Putting (32) and (39) together yields

$$\sum_{e \in S} \mathbf{x}_{t+1}(e) = \sum_{e \in T_1} \mathbf{x}_{t+1}(e) + \dots + \sum_{e \in T_{k_1}} \mathbf{x}_{t+1}(e) + \sum_{e \in S_1} \mathbf{x}_{t+1}(e) + \dots + \sum_{e \in S_{k_2}} \mathbf{x}_{t+1}(e) \quad (40)$$

$$\leq \sum_{i=1}^{k_1} n(T_i) - k_1 + \sum_{i=1}^{k_2} n(S_i) - k_2 \quad (41)$$

$$= n - k_1 - k_2 = n - c(S) = \text{rank}(S). \quad (42)$$

950 This finishes the proof that $\sum_{e \in S} \mathbf{x}_{t+1}(e) \leq \text{rank}(S)$, thus $\mathbf{x}_{t+1} \in \mathcal{B}(G)$.

951 The proof of termination, continuity and a.e. differentiability is almost identical to the proof of
 952 Theorem E.1. \square

E.2.1 Generating neural predictions in the Graphical Base Matroid

We need to massage the output of the neural network in a differentiable and computationally efficient way so that it obeys a relaxed constraint and can be passed to Algorithm 3. Algorithm 3 requires \mathbf{x} to lie within the graphical matroid base polytope $\mathcal{B}(G)$, which represents a relaxed version of the original constraints. Without loss of generality we assume G is a connected graph. Every point inside the graphical matroid base polytope can be seen as a distribution over spanning trees. We first define the uniform distribution over the set of all spanning trees and forces our encoder network to generate a perturbation to this uniform distribution. Some notations are in order.

Fix an arbitrary orientation over the edges of G . For an edge $e = (u, v)$, i.e., oriented from u to v , let

$$b_e = \mathbf{1}_u - \mathbf{1}_v \text{ with } \mathbf{1}_u(v) = \begin{cases} 1 & \text{if } v = u \\ 0 & \text{otherwise} \end{cases} \quad (43)$$

The Laplacian of (positively) weighted graph G is $L_G = \sum_{e \in E} w(e) b_e b_e^\top$. Let L_G^\dagger denote the Moore–Penrose pseudo-inverse of L_G .

Generating a perturbation vector. Let \mathcal{T} denote the set of all spanning trees of G and μ be a uniform distribution of all spanning trees of G . A uniform spanning tree distribution is a uniform distribution over all spanning trees of a given graph. If the graph is weighted, then we can study the weighted uniform distribution of spanning trees where the probability of each tree is proportional to the product of the weight of its edges. For $w : E \rightarrow \mathbb{R}_+$, we say $\mu(w)$ is a w -uniform spanning tree distribution, if for any spanning tree $T \in \mathcal{T}$,

$$\Pr[T] \propto \prod_{e \in T} w(e).$$

Let $\mu_e(w) := \Pr_T \sim \mu[e \in T]$ be the marginal probability of edge e . The following result gives us a way of writing an analytical expression for μ_e .

Theorem E.4 ([80]). *For any edge e*

$$\mu_e = b_e^\top L_G^\dagger b_e.$$

Moreover, for weighted graph G with $w : E \rightarrow \mathbb{R}_+$, if $w(e)$ is the weight of e , then

$$\mu_e(w) = w(e) b_e^\top L_G^\dagger b_e.$$

The above Theorem E.4 suggests the following way of generating a point inside the graphical matroid base polytope. Let $w \in \mathbb{R}_+^E$ be the output of an encoder network that assigns a positive weight to every edge of graph G . We think of w as a perturbation to the uniform distribution over the spanning trees. Then obtain vector \mathbf{x} whose e -th entry is $\mu_e(w)$ according to the weight vector w and Theorem E.4. The vector \mathbf{x} lies inside the graphical matroid base polytope $\mathcal{B}(G)$. Recall that $\mathcal{B}(G) = \text{conv}\{\mathbf{1}_T : T \in \mathcal{T}\}$ and $\mathbf{x} = \mathbb{E}_{T \sim \mu(w)}[\mathbf{1}_T]$ is a convex combination of indicator vector of the spanning trees. It remain to show that $\mu(w)$ is continuous and differential with respect to w .

Theorem E.5. *Let $G = (V, E, w)$ be a connected, weighted graph with non-negative edge weights $w : E \rightarrow \mathbb{R}_{\geq 0}$ and let*

$$\mu_e(w) = \mathbb{P}_{T \sim \mu(w)}[e \in T] = w(e) b_e^\top L_G^\dagger(w) b_e, \quad e \in E,$$

where b_e is the signed incidence vector of e and $L_G(w) = \sum_{f \in E} w(f) b_f b_f^\top$ is the weighted Laplacian (with Moore–Penrose pseudoinverse L_G^\dagger). Then

- i. $\mu_e : \mathbb{R}_{\geq 0}^E \rightarrow \mathbb{R}$ is continuous on the closed orthant;
- ii. μ_e is C^∞ (indeed analytic) on the open orthant $\mathbb{R}_{> 0}^E$;
- iii. μ_e is differentiable everywhere on $\mathbb{R}_{\geq 0}^E$.

Proof. Let $\tilde{L}_G(w)$ denotes the reduced Laplacian with any single row/column removed. By Kirchhoff’s Matrix–Tree Theorem [40] we have

$$\tau(w) = \det(\tilde{L}_G(w)) = \sum_{\text{spanning trees } T} \prod_{f \in T} w(f),$$

990 For an edge e we have,

$$\mu_e(w) = 1 - \frac{\tau(w - e)}{\tau(w)} \quad (44)$$

991 where $w - e$ denotes the weight vector with $w(e) = 0$. Both numerator and denominator in (44) are
 992 (multivariate) polynomials; moreover $\tau(w) > 0$ whenever at least one spanning tree has positive total
 993 weight, which is guaranteed by the connectivity of G . Hence the ratio is continuous on $\mathbb{R}_{\geq 0}^E$, proving
 994 part (i).

995 On the interior $\mathbb{R}_{> 0}^E$ the reduced Laplacian $\tilde{L}_G(w)$ is positive definite (PD). The maps

$$w \mapsto \tilde{L}_G(w) \quad (\text{affine map}), \quad A \mapsto A^{-1} \quad (C^\infty \text{ on PD}),$$

996 compose smoothly, so the formula $\mu_e(w) = w(e) b_e^\top \tilde{L}_G(w)^{-1} b_e$ is infinitely differentiable i.e., is
 997 C^∞ and indeed real-analytic on $\mathbb{R}_{> 0}^E$, giving (ii).

998 It remains to show (iii) at boundary points where some coordinates of w equal 0. Let w_0 be a weight
 999 vector with at least one zero entry, and Z be the set of zero weight edges. There are two cases:

1000 Case A: $G - Z$ is connected. In this case the reduced Laplacian $\tilde{L}_G(w_0)$ positive definite since $G - Z$
 1001 has at least one spanning tree, so the same smoothness argument used for (ii) applies; thus $\mu_e(w_0)$ is
 1002 differentiable (in fact is C^∞).

1003 Case B: $G - Z$ is disconnected. Then every spanning tree of any nearby weight vector must contain
 1004 the cut-edge whose weight vanished, so the term in Equation (44) equals 1 in a whole neighbourhood;
 1005 hence μ_e is locally constant and therefore differentiable at w_0 with derivative 0.

1006 In either case one-sided (classical) derivatives exist and coincide with the interior derivative, complet-
 1007 ing (iii). \square

1008 F Case study: Maximum Independent Set

1009 For the maximum independent set problem, we want to find the largest set S of nodes in a graph
 1010 $G = (V, E)$, where no pair of nodes in the set is adjacent. Unfortunately, there is no known compact
 1011 description in terms of inequalities for the independent set polytope [55]. This automatically calls
 1012 into question whether there is a consistent decomposition that can be designed.

1013 First, let us assume that $\emptyset \in \mathcal{C}$, and that the graph G has no isolated nodes. In order to be able to
 1014 optimize over independent sets we are going to need access to a polytope description. The constraints
 1015 $x_i \geq 0$ and $x_i + x_j \leq 1$ are defining the polytope referred to in the literature as FSTAB [55]. FSTAB
 1016 is a relaxation of the independent set polytope since it includes half-integral vertices. We will explain
 1017 how we will leverage FSTAB to explore the space of independent sets.

1018 For step 3 we use Equation (8) to obtain a vertex of the polytope FSTAB. For convenience, let
 1019 us call this vertex \mathbf{v}_t . The i -th coordinate of \mathbf{v}_t is denoted by $v_{t,i}$. Note that now the vertex may
 1020 contain half-coordinates. This will affect our coefficient selection. We need to select a coefficient
 1021 that does not violate the polytope inequalities. The constraints $0 \leq \mathbf{x} \leq 1$ imply the same rule for the
 1022 coefficient a_t . However, we have an additional consideration. If we just choose the coefficients as in
 1023 the cardinality constraint case, when the coordinates $i, j \notin S$ and $(i, j) \in E$ get rescaled by $1 - a_t$,
 1024 we may break the constraint for $x_i + x_j \leq 1$. This leads us to the following rule:

$$a_t = \min \left\{ \min_{i: v_{t,i} > 0} \frac{x_i^{(t-1)}}{v_{t,i}}, \min_{j: v_{t,j} < 1} \frac{1 - x_j^{(t-1)}}{1 - v_{t,j}}, \min_{\substack{(u,v) \in E \\ v_{t,u} = v_{t,v} = 0}} (1 - x_u^{(t-1)} - x_v^{(t-1)}) \right\}. \quad (45)$$

1025 **Proposition F.1.** *The algorithm on FSTAB using coefficients from Equation (45) and Equation (8)*
 1026 *for step 3, obtains a decomposition in at most $n + |E|$ steps.*

1027 *Proof.* At each step, a coordinate is either 0, 1, or an inequality is tightened. Once any of those events
 1028 happens, they cannot be undone. If all inequalities are tightened, then we are done. This can take at
 1029 most $|E|$ steps. Similarly it requires at most n steps if all coordinates are set to 0 or 1. Therefore, the
 1030 algorithm will terminate in at most $n + |E|$ steps. \square

1031 **Obtaining a point on FSTAB.** Since we would like to build a support that avoids half-integral points
 1032 as much as possible we follow a simple gradient-based projection scheme for FSTAB. We begin by
 1033 defining the total edge-violation of a point $\mathbf{x} \in \mathbb{R}^n$ with slack parameter $s \geq 0$ as

$$V(\mathbf{x}) = \sum_{(u,v) \in E} \max\{0, x_u + x_v + s - 1\},$$

1034 which measures how much each edge (u, v) exceeds the bound $x_u + x_v + s \leq 1$. To correct all
 1035 violations at once, we compute $\nabla V(\mathbf{x}) = (d_1, \dots, d_n)$, where

$$d_i = \sum_{\{j: (i,j) \in E\}} \mathbf{1}[x_i + x_j + s - 1 > 0]$$

1036 counts how many incident edges on node i are violated. Next, for each violated edge (u, v) with
 1037 violation amount

$$b_{uv} = x_u + x_v + s - 1 > 0,$$

1038 we require a step-size η satisfying $b_{uv} - \eta(d_u + d_v) \leq 0$, which implies

$$\eta \geq \frac{b_{uv}}{d_u + d_v}.$$

1039 Taking the maximum of these ratios over all violated edges gives the minimal η that fixes every
 1040 violation:

$$\eta = \max_{(u,v): x_u + x_v + s > 1} \frac{x_u + x_v + s - 1}{d_u + d_v}.$$

1041 Finally, we compute

$$\mathbf{x}' = \sigma(\mathbf{x} - \eta \nabla V(\mathbf{x})),$$

1042 where σ is a ReLU. This guarantees $x'_u + x'_v + s \leq 1$ for every $(u, v) \in E$ and $\mathbf{x}' \geq 0$, which
 1043 ensures that we have a point in FSTAB. However, the decomposition may still yield several half-
 1044 integral corners which will significantly affect the quality of our extension, since we need the support
 1045 sets to be independent. The half-integral corners can be reduced by tightening the relaxation and
 1046 including additional inequalities in the literature including clique constraints and odd cycle constraints
 1047 [55]. Many of those come down to imposing a l1 norm constraint on some subset of a coordinates
 1048 corresponding to a subgraph. We may enforce a norm constraint followed by the FSTAB projection.
 1049 It is easy to see that, if the norm is 1 and the constraints are all satisfied, then we can obtain a
 1050 decomposition into independent sets. To encourage improved exploration we may project to a larger
 1051 norm given by a naive bound on the size of the maximum independent set such as $n/(\Delta + 1)$, where
 1052 Delta is the maximum degree. We can then keep reducing the norm/reprojecting to obtain higher
 1053 quality supports that avoid half-integral corners.

1054 G Experiments

1055 Here, we provide detailed experimental settings and some additional experimental results. All the
 1056 datasets are public and our code is available at [https://anonymous.4open.science/r/Neural_](https://anonymous.4open.science/r/Neural_Combinatorial_Optimization_with_Constraints-3485/README.md)
 1057 [Combinatorial_Optimization_with_Constraints-3485/README.md](https://anonymous.4open.science/r/Neural_Combinatorial_Optimization_with_Constraints-3485/README.md).

1058 **Hardware.** All experiments are run on 16 cores (32 threads) of Intel(R) Xeon(R) Platinum 8268
 1059 CPU (24 cores, 48 threads in total), 32 GB ram, with a single Nvidia RTX8000 48GB GPU.

1060 **Datasets.** Following previous work [9, 87], we evaluate our methods on both synthetic and real-
 1061 world bipartite graphs (U, V, E) , V is the ground set and the goal is to select k nodes from V so that
 1062 we cover maximum number of nodes from U .

- 1063 • **Random Uniform.** The *Random Uniform* datasets include both the Random500 and
 1064 Random1000 settings, and are used for both training and testing. Each dataset consists of
 1065 100 independently generated bipartite graphs, where $|V| = 500$, $|U| = 1000$ (Random500)
 1066 or $|V| = 1000$, $|U| = 2000$ (Random1000). Each $u \in U$ is assigned a weight uniformly
 1067 at random from 1 to 100. Each $v \in V$ covers a random subset of U , with the number of
 1068 covered nodes for each v chosen uniformly at random between 10 and 30.

- **Random Pareto.** The Random Pareto dataset consists of 100 independently generated bipartite graphs with $|U| = |V| = 1000$, used exclusively for training in the Twitch experiments. In each graph, the number of covered nodes per covering node $v \in V$ follows a Pareto distribution, with the α parameter randomly selected between 1 and 2, resulting in a heavy-tailed degree distribution (typically, 20% of covering nodes account for 80% of the edges). Each $u \in U$ is assigned a weight uniformly at random from 1 to 100, and every $u \in U$ is covered by at least one $v \in V$.
- **Twitch.** The Twitch datasets model social networks of streamers grouped by language, with $|U| = |V|$ set to the number of streamers. The dataset includes: DE ($|U| = |V| = 9498$), ENGB (7126), ES (4648), FR (6549), PTBR (1912), and RU (4385). The objective is to maximize the sum of logarithmic viewer counts over U .
- **Railway.** The Railway datasets are derived from real-world Italian railway crew assignments. We evaluate on three graphs: rail507 ($|V| = 507$, $|U| = 63009$), rail516 ($|V| = 516$, $|U| = 47311$), and rail582 ($|V| = 582$, $|U| = 55515$).

Baselines. We adopt the same baselines as prior work, including:

- **Random.** Samples k candidates uniformly at random over multiple trials, selects the best within 240 seconds.
- **Greedy algorithm [61].** Iteratively adds the element with the largest marginal gain, achieving a $(1 - 1/e)$ approximation.
- **Gurobi.** Exact MIP solver, time-limited to 120 seconds per instance. The version used is Gurobi 12.01.
- **EGN [34].** Optimizes a probabilistic objective with naive rounding; does not guarantee feasibility during optimization.
- **CardNN [87].** One-shot neural solvers for cardinality-constrained problems. Main variants: CardNN-S (Sinkhorn), CardNN-GS (Gumbel-Sinkhorn), CardNN-HGS (Homotopy Gumbel-Sinkhorn). For each, a CardNN-noTTO variant omits test-time optimization.
- **UCOM2 [9].** Combinatorial optimization using greedy incremental derandomization. Three variants differ only by test-time augmentation and running time: UCOM2-short (no augmentation, fastest), UCOM2-middle (moderate augmentation, medium time), UCOM2-long (maximal augmentation, longest time).
- **RL.** For the RL baseline, we follow the instructions in [45, 70] to use GraphSage layers [26] as policy network with Actor-Critic [44] algorithm to train on the same problem instances.

G.1 Training

To enhance optimization stability and solution quality, we incorporate several training techniques.

A rescaling coefficient in Appendix C.1 is referred to as a *scaling factor*. A set of *scaling factors* $:[1.0, 0.8, 0.6, 0.4, 0.2, 0.1, 0.05, 0.02, 0.01]$ are used in the training and inference parallelly, helping to get stable decompositions across different data distributions.

An *entropy regularization* term is applied with a cosine-decayed weight λ_t , initialized at 0.05 and annealed to zero by the 30th epoch, promoting exploration within the hypersimplex.

Convergence speed is controlled via a *sharpness factor* applied to the noise perturbation layer when the sigmoid activation function is used in that layer (when using min-max scaling, it is ignored). The sharpness factor is linearly increased from 0.3 at the onset of training to 1.0 by epoch 80.

Gaussian noise is injected into the model logits throughout training to foster exploration and enhance robustness. The standard deviation of this noise is initially set to 0.05 and is reduced to zero according to a cosine decay schedule as training progresses.

The other hyperparameters are listed below:

- Dropout: 0.1 (applied only during training).
- Optimizer: AdamW.

- Learning Rate: 5×10^{-3} .
- Learning Rate Scheduler: `warmup_cosine`, with 50 warmup epochs, decaying to a minimum of 5×10^{-5} .
- Weight Decay: 1×10^{-4} .
- Epochs: 80.
- Batch Size: 4.
- Seed: 42.
- Workers: 16 CPU worker threads for data loading, preprocessing, and decomposition with multiple scaling factors.

G.2 Inference strategies and local improvement

We evaluate our model using three distinct inference strategies—*Short*, *Medium*, and *Long*—each balancing computational efficiency and local improvement:

Local improvement. To further refine the candidate solution, we employ a local search algorithm based on iterative element replacement. At each step, the algorithm considers replacing a single element in the current solution set with one from outside the set (within a defined candidate pool), accepting only replacements that yield a strictly improved objective value. The procedure terminates when no improving replacements are found or when the maximum iteration count is reached.

- **Short:** For each instance, 50 decomposed sets are generated from each scaling factor and the best among them is chosen as the result, with no further local improvement applied.
- **Medium:** For each instance, 100 decomposed sets are generated from each scaling factor and the best among them is chosen as the base set. The base set then undergoes the local improvement procedure, performed for up to 10 iterations. The candidate pool consists of the first k nodes produced by the scaling factor 0.1 that is not present in the base set.
- **Long:** For each instance, 5 additional augmented graphs are generated (with 0–30% feature noise and 0–30% random edge dropout). For each graph, decomposed sets are generated from each scaling factor and the best among them is chosen as the base set. Local improvement is performed for up to k iterations on the original graph, with the candidate pool includes all nodes that appears in decomposed sets across from the scaling factor where the base set is selected. The best solution found among all augmented graphs is selected as the final result for the instance.

G.3 Ablation study: local improvement candidate pool selection

To demonstrate that the strategic selection of nodes in our candidate pool provides meaningful benefits, we conducted an ablation study comparing our decomposition-based candidate pool against a random alternative of equal size. In this ablation test, we maintain the same pre-local improvement solution (the base set) as the starting point for both approaches. The original candidate pool is constructed using our medium and long methods. The alternative candidate pool retains the same base set but replaces the remaining candidates with randomly selected nodes, ensuring both pools have identical size for fair comparison.

Table 2: Ablation study comparing decomposition-based candidate pool versus random alternative. Improvement percentages over the base set.

Method	rand500 k=10		rand1000 k=20		rand500 k=50		rand1000 k=100	
	Medium	Long	Medium	Long	Medium	Long	Medium	Long
Decomposition-based (Ours)	2.41%	3.28%	5.15%	8.09%	2.11%	3.89%	3.83%	8.07%
Random Alternative	0.23%	1.02%	3.03%	5.74%	0.43%	2.39%	2.84%	5.79%

We evaluate both local improvement strategies across medium and long inference modes. The results, Table 2, demonstrate that our strategic candidate selection provides consistent improvements over

random selection, validating that our local improvement procedure benefits specifically from the candidate pool construction from our decomposition rather than simply having access to additional nodes for local improvement.

G.4 Ablation tests for UCOM2

As stated in Section 5.2, we categorize UCOM2 as a non-learning method and conduct several ablation studies to substantiate this classification. Specifically, we evaluate two variants: setting the neural network output to zero (UCOM2-zero-start-short), and to random uniform values between 0 and 1 (UCOM2-randomstart). Both ablations disregard the output of the neural network. We find that UCOM2-zero-start-short achieves results nearly identical to the standard short variant, while UCOM2-randomstart attains similar performance to all three UCOM2 variants, depending on the number of random samples generated. These results demonstrate that UCOM2’s effectiveness is in large part due to their greedy module and the fact that the maximum coverage objective is a monotone submodular function.

Table 3: Ablation results for UCOM2 variants on Random500 and Random1000 datasets. Running time (time): smaller the better. Objective (obj): the larger the better. The standard deviation are captured to show the variability of the dataset and the method.

Method	Random500, $k = 10$		Random500, $k = 50$		Random1000, $k = 20$		Random1000, $k = 100$	
	Time↓	Obj↑	Time↓	Obj↑	Time↓	Obj↑	Time↓	Obj↑
UCOM2-zero-start-short	0.1089 ± 0.0309s	15551.01 ± 367.12	0.0885 ± 0.1032s	44311.87 ± 819.96	0.2275 ± 0.0164s	30991.19 ± 528.97	0.1682 ± 0.0360s	88693.99 ± 1248.72
UCOM2-randomstart-short	0.8862 ± 0.1238s	15294.04 ± 408.52	0.6957 ± 0.1369s	44237.92 ± 824.05	1.6408 ± 0.0329s	29878.53 ± 601.10	1.5426 ± 0.0449s	88549.61 ± 1342.89
UCOM2-randomstart-medium	3.8521 ± 0.2770s	15586.95 ± 360.89	15.4367 ± 0.0493s	44867.19 ± 741.85	9.4358 ± 0.0415s	30437.43 ± 532.00	8.8802 ± 0.0522s	89077.84 ± 1244.72
UCOM2-randomstart-long	35.9862 ± 2.5628s	15672.89 ± 351.74	30.5812 ± 0.0850s	44923.74 ± 754.78	89.8093 ± 0.3444s	30752.49 ± 529.91	84.3725 ± 0.4054s	89386.56 ± 1245.85
UCOM2-short	1.0411 ± 0.0827s	15253.35 ± 370.00	0.7392 ± 0.0150s	44208.95 ± 768.68	1.7342 ± 0.0677s	29791.66 ± 678.02	1.6216 ± 0.0644s	88472.61 ± 1261.36
UCOM2-medium	4.1891 ± 0.1150s	15589.25 ± 363.43	16.0005 ± 0.0408s	44852.82 ± 765.11	9.5523 ± 0.0620s	30420.69 ± 552.66	8.9956 ± 0.0890s	89072.92 ± 1248.58
UCOM2-long	39.2018 ± 0.9474s	15779.45 ± 358.41	31.7587 ± 0.0811s	44906.50 ± 761.75	90.7475 ± 0.3342s	30744.61 ± 512.61	85.2663 ± 0.6164s	89408.28 ± 1232.74

On the twitch dataset, UCOM2 uses zero initialization (UCOM2-zero-start) in its original implementation, fully ignoring the trained model. To assess the impact of the neural network, we also evaluate UCOM2-withmodel on twitch, which yields markedly inferior results. This further supports our observation that all learning-based methods perform poorly on Twitch due to the architecture of the encoder network.

Table 4: Ablation results for UCOM2 variants on the Twitch dataset. Running time (time): smaller the better. Objective (obj): the larger the better. The standard deviations are captured to show the variability of the dataset and the method.

Method	Twitch, $k = 20$		Twitch, $k = 50$	
	Time↓	Obj↑	Time↓	Obj↑
Ucom2-withmodel-short	73.2402 ± 80.5553s	18.67 ± 45.72	115.8045 ± 126.7061s	573.67 ± 376.52
Ucom2-withmodel-medium	333.3645 ± 363.4380s	102.00 ± 41.71	532.9243 ± 585.9112s	3018.17 ± 5208.31
Ucom2-withmodel-long	659.8775 ± 719.8658s	150.00 ± 48.88	1054.5567 ± 1160.6106s	3029.67 ± 5202.21
Ucom2-zero-start-short	17.8031 ± 18.4783s	25853.00 ± 1112.59	17.3739 ± 18.1940s	30526.00 ± 13043.86
Ucom2-zero-start-medium	19.6806 ± 19.8771s	25858.17 ± 11206.89	21.6114 ± 21.3478s	30544.17 ± 13052.91
Ucom2-zero-start-long	22.0589 ± 21.6869s	25858.17 ± 11206.89	27.2170 ± 25.6604s	30544.17 ± 13052.91

G.5 Detailed results

Below are the plots and tables showing the full detailed test results. The raw numerical results are given in Tables 9 and 10, and Table 11. The standard deviation are captured to show the variability of the dataset and the method.

Table 5: Performance comparison of our method against baseline approaches across Learning without TTO, on multiple datasets. Metrics used are inference time (lower is better) and objective value (higher is better). In the Learning without TTO setting, our short version consistently outperforms all baselines in both inference time and objective value, demonstrating strong learning capability.

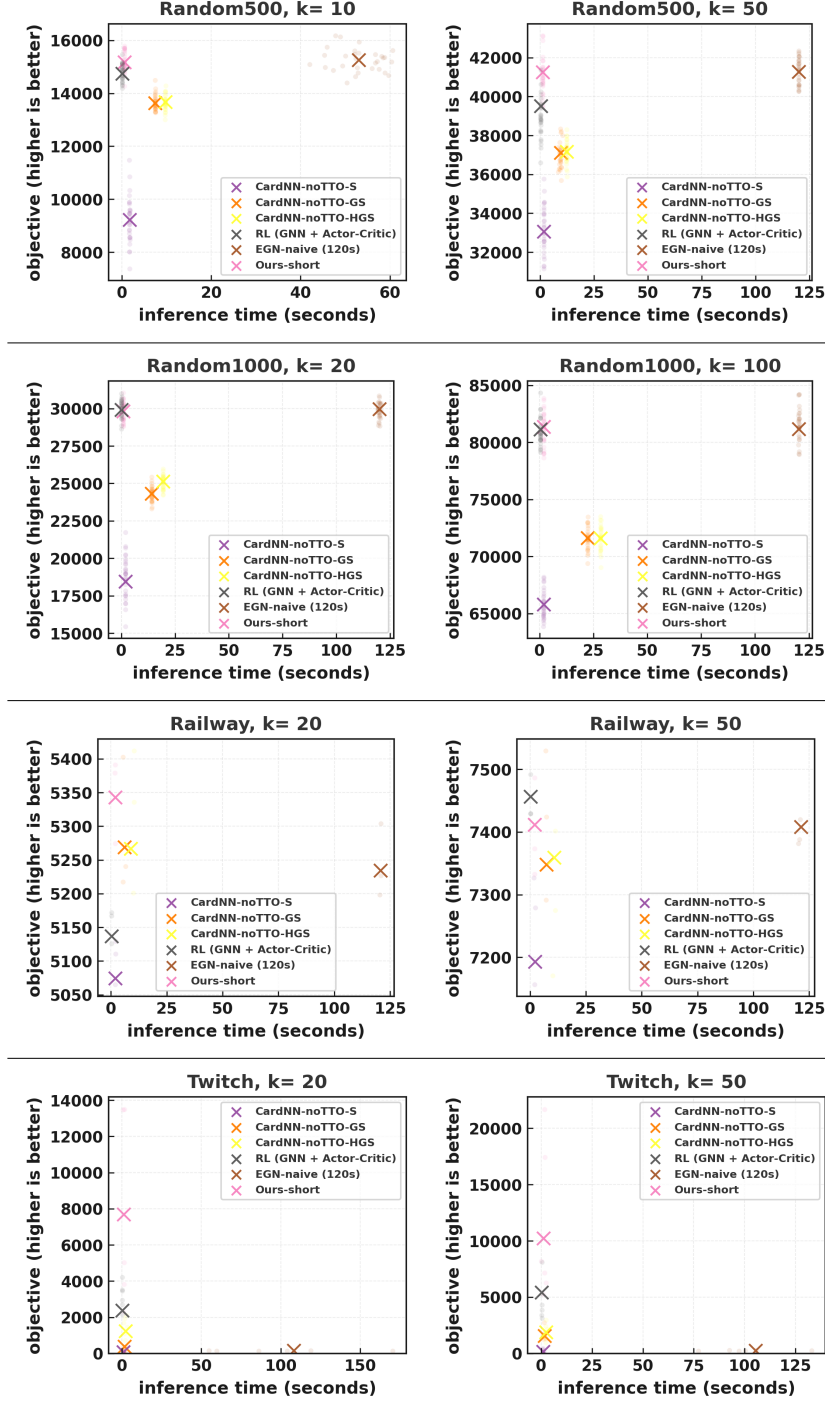


Table 6: Performance comparison of our method against baseline approaches across Learning with TTO, on multiple datasets. Metrics used are inference time (lower is better) and objective value (higher is better). When extended to medium and long versions, our method surpasses most TTO-based baselines across datasets, with the exception of the Twitch dataset.

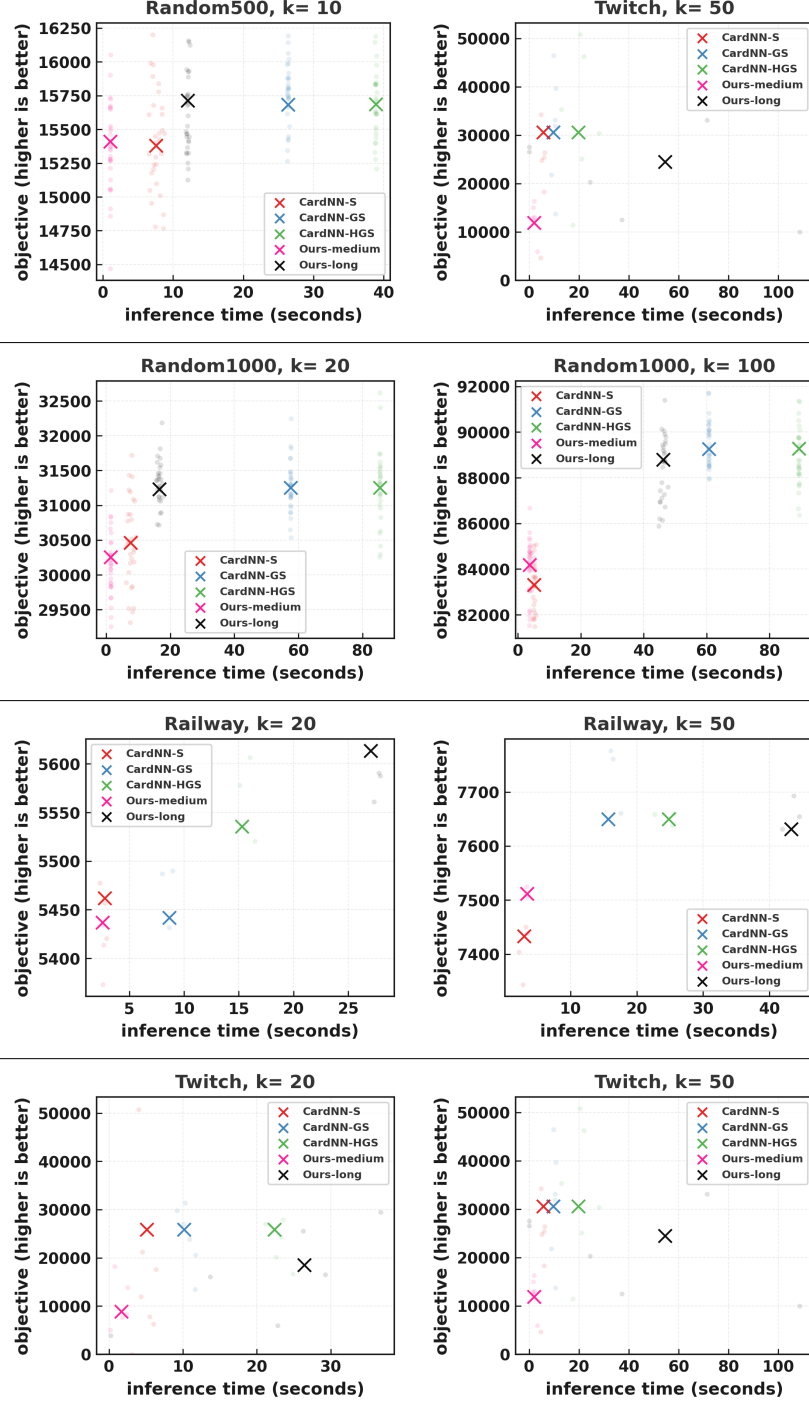


Table 7: Performance comparison of our method against baseline approaches across non-Learning/traditional methods on multiple datasets. Metrics used are inference time (lower is better) and objective value (higher is better). While greedy is an efficient baseline with a strong approximation guarantee, our method performs competitively and is capable of outperforming it on datasets like Random500 for larger values of k . We are also able to outperform UCOM in several cases (e.g., Random1000 and Railway), though there are instances where greedy and/or UCOM perform the best, such as the twitch dataset.

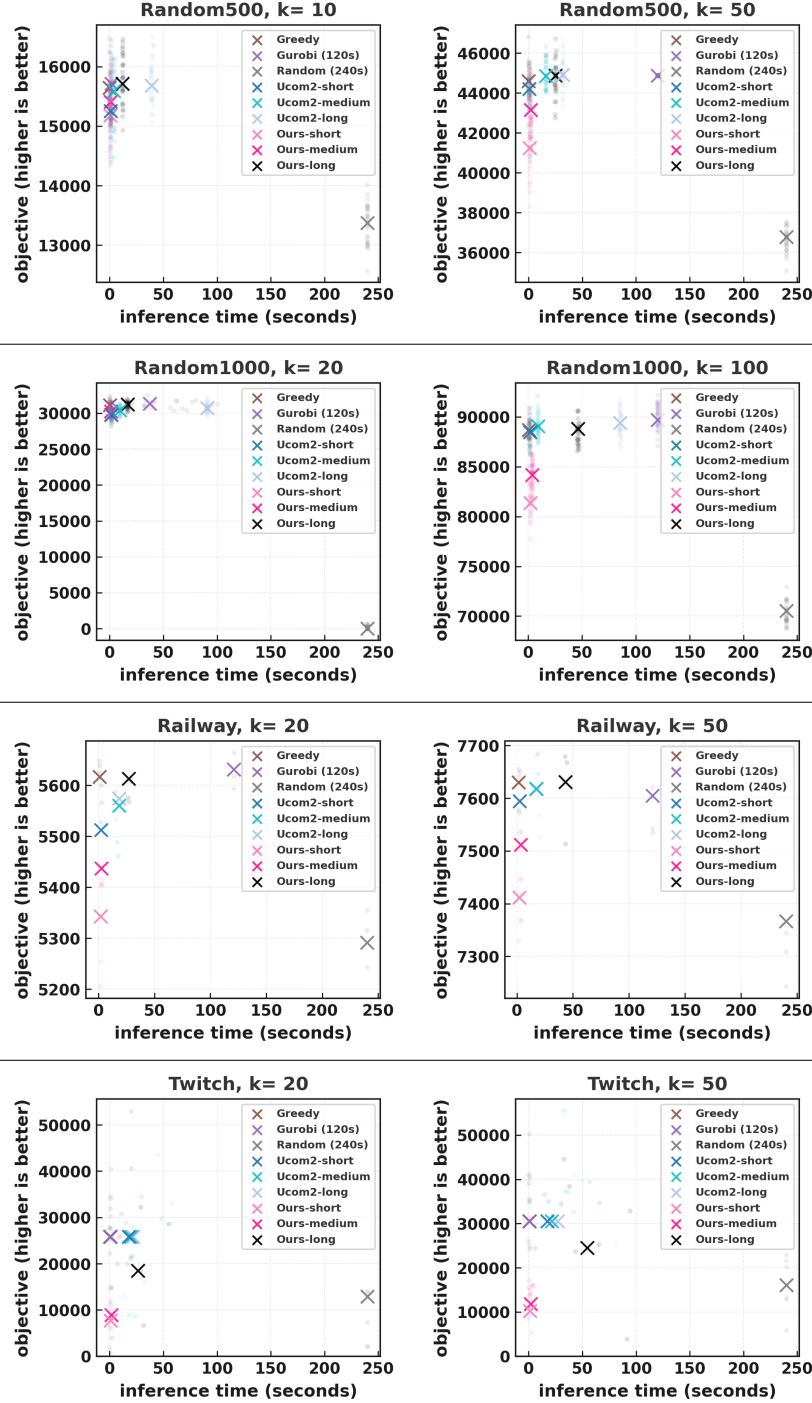


Table 8: Performance comparison of our method against all baseline approaches. Metrics used are inference time (lower is better) and objective value (higher is better).

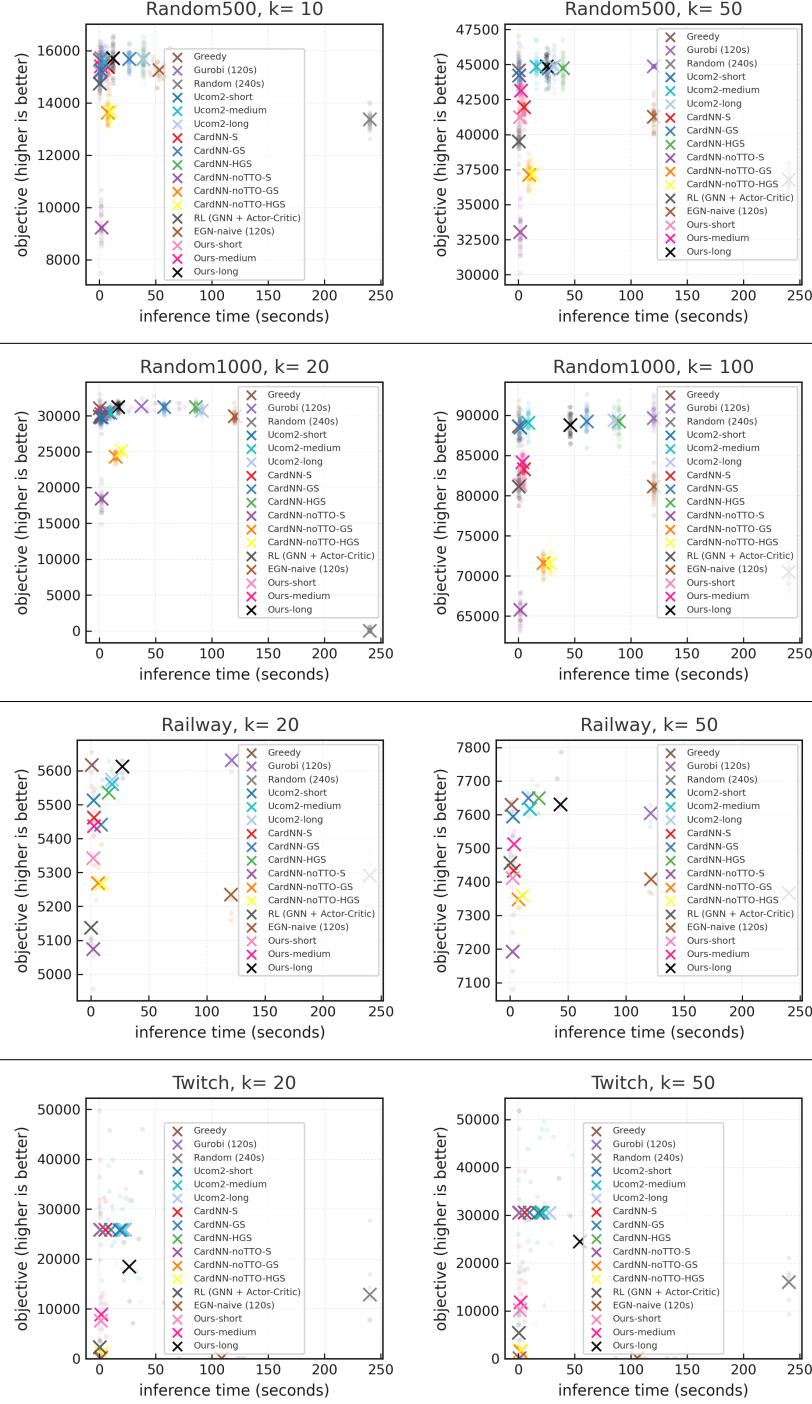


Table 9: Raw numerical results on maximum coverage problem, comparing our method against baseline approaches across Learning without TTO, on multiple datasets. Running time (time): smaller the better. Objective (obj): the larger the better.

Method	rand500				rand1000				railway				twitch			
	k=10		k=50		k=20		k=100		k=20		k=50		k=20		k=50	
	Time↓	Obj↑	Time↓	Obj↑	Time↓	Obj↑	Time↓	Obj↑	Time↓	Obj↑	Time↓	Obj↑	Time↓	Obj↑	Time↓	Obj↑
CardNN-noTTO-S	1.6880s ± 0.0791s	9231.54 ± 827.09	1.7211s ± 0.0661s	33055.87 ± 1211.03	1.8693s ± 0.0018s	18458.92 ± 1283.61	1.8810s ± 0.0453s	65793.40 ± 1478.50	1.9287s ± 0.2161s	5074.33 ± 59.41	2.1892s ± 0.2505s	7193.00 ± 80.72	0.8636s ± 0.2075s	77.0000 ± 51.4704	0.9635s ± 0.4585s	170.50 ± 158.15
CardNN-noTTO-GS	7.4571s ± 0.0300s	13635.05 ± 303.36	9.524s ± 0.5770s	37120.46 ± 610.25	14.1629s ± 0.0301s	24309.37 ± 465.91	22.3352s ± 0.0942s	71620.38 ± 1026.81	5.9863s ± 0.5143s	5269.00 ± 68.83	7.3275s ± 0.3261s	7348.33 ± 80.31	1.5749s ± 0.3842s	376.0000 ± 194.6679	1.6334s ± 0.6782s	1574.33 ± 631.04
CardNN-noTTO-HGS	9.6954s ± 0.0429s	13671.99 ± 302.84	12.2502s ± 0.1068s	37164.81 ± 678.19	19.3888s ± 0.0532s	25135.52 ± 436.21	28.3368s ± 0.1218s	71604.61 ± 1010.17	8.8791s ± 0.7784s	5266.67 ± 76.01	10.8211s ± 0.6323s	7359.00 ± 70.89	2.1831s ± 0.7328s	1242.33 ± 499.23	2.3674s ± 0.7858s	1898.00 ± 506.67
EGN-naive(120s)	53.0409s ± 5.3879s	15262.76 ± 401.10	120.1361s ± 0.0796s	41272.68 ± 737.36	120.0105s ± 0.3980s	29968.04 ± 563.02	120.3565s ± 0.1600s	81166.12 ± 1391.02	120.6763s ± 0.2931s	5234.67 ± 52.47	121.3351s ± 0.9133s	7408.33 ± 72.82	108.3030s ± 24.3569s	152.67 ± 12.22	105.4605s ± 20.7886s	247.17 ± 31.88
RL(GNN+Actor-Critic)	0.0696 ± 0.0028s	14741.26 ± 322.51	0.2471 ± 0.029s	39510.96 ± 975.21	0.1021 ± 0.0029s	29912.14 ± 571.12	0.4652 ± 0.0029s	81158.54 ± 1215.23	0.2191 ± 0.0030s	5137 ± 52.01	0.2191 ± 0.0031s	7456.67 ± 57.14	0.1694 ± 0.0028s	2379.5 ± 1471.50	0.2905 ± 0.0030s	5435.5 ± 3318.67
Ours-short	0.5786s ± 0.0094s	15177.77 ± 344.68	1.1078s ± 0.0149s	41247.80 ± 965.48	0.8248s ± 0.0312s	29810.12 ± 586.05	1.9076s ± 0.0284s	81357.29 ± 1277.84	1.8559s ± 0.0458s	5343.00 ± 89.17	2.0838s ± 0.0142s	7411.67 ± 55.08	1.0975s ± 0.7446s	7689.00 ± 5712.56	1.1485s ± 0.7596s	10227.00 ± 5537.29

Table 10: Raw numerical results on maximum coverage problem, comparing our method against baseline approaches with TTO. Running time (time): smaller the better. Objective (obj): the larger the better.

Method	rand500				rand1000				railway				twitch			
	k=10		k=50		k=20		k=100		k=20		k=50		k=20		k=50	
	Time↓	Obj↑	Time↓	Obj↑	Time↓	Obj↑	Time↓	Obj↑	Time↓	Obj↑	Time↓	Obj↑	Time↓	Obj↑	Time↓	Obj↑
CardNN-S	7.5387s ±	15381.60 ±	4.9843 ±	41971.22 ± 814.87	7.6054s ±	30461.73 ±	5.0909s ±	83319.37 ± 1324.81	2.7580s ±	5462.00 ±	3.0035s ±	7433.33 ±	5.1258s ±	25863.83 ±	5.5767s ±	30556.00 ±
CardNN-GS	0.6576s 26.3746	394.79 15683.40	0.2941 26.8944s	44724.42 ± 773.70	0.6608s 57.5445s	590.50 31250.56	0.3016s 60.7239s	89269.41 ± 1253.68	0.2705s 8.6626s	56.00 5441.67	0.2782s 15.6866s	63.34 7650.00	0.6503s 10.1648s	11198.96 25864.00	0.5135s 9.5794	13063.12 30560.67
CardNN-HGS	0.0831 38.8875s	350.79 15685.39	0.1131s 39.6041s	44745.20 ± 769.93	0.0670s 85.4505s	528.72 31250.47	0.0665s 89.3385s	89280.17 ± 1261.34	0.5672s 15.2983s	58.77 5535.67	1.0987s 24.7941s	73.33 7650.00	1.1928s 22.3602s	11198.73 25864.00	1.5230 19.7813s	13061.60 30560.83
	0.1065s	349.83	0.1122s		0.0952s	526.03	0.1235s		0.5441s	64.52	0.9907s	88.71	1.2962s	11198.73	3.4102s	13061.35
Ours-medium	1.0546s ±	15410.23 ±	2.1212s ±	43152.87 ± 1004.95	1.4175s ±	30257.05 ±	3.6669s ±	84186.79 ± 1336.48	2.5628s ±	5437.00 ±	3.4385s ±	7512.00 ±	1.6693s ±	8887.83 ±	1.9401s ±	11858.50 ±
Ours-long	0.0146s 12.0878s	353.91 15714.63	0.0096s 24.9885s	44866.84 ± 876.78	0.0633s 16.6007s	606.07 31230.26	0.0582s 46.1200s	88796.03 ± 1327.83	0.0418s 27.0344s	66.84 5613.33	0.0532s 43.2777s	63.85 7631.00	0.9268s 26.4401s	5087.20 18526.83	1.0147s 54.4778s	4793.61 24509.83
	0.1365s	354.22	0.3543s		0.4632s	476.25	0.6998s		0.9617s	55.58	1.3728s	75.19	15.9632s	8671.00	32.5091s	11171.33

Table 11: Raw numerical results on maximum coverage problem, comparing our method against non-learning/traditional baseline approaches. Running time (time): smaller the better. Objective (obj): the larger the better.

Method	rand500				rand1000				railway				twitch			
	k=10		k=50		k=20		k=100		k=20		k=50		k=20		k=50	
	Time↓	Obj↑	Time↓	Obj↑	Time↓	Obj↑	Time↓	Obj↑	Time↓	Obj↑	Time↓	Obj↑	Time↓	Obj↑	Time↓	Obj↑
Random (240s)	240.0000	13372.76	240.0000	36786.89	240.0000	24133.50	240.0000	70527.31	240.0000	5291.67	240.0000	7367.00	240.0000	12889.17	240.0000	16050.50
Gurobi (120s)	±	±	±	±	±	±	±	±	±	±	±	±	±	±	±	±
	0.000s	308.83	0.000s	± 655.45	0.000s	390.27	0.000s	± 1051.87	0.000s	73.92	0.000s	76.97	0.000s	5636.10	0.000s	6715.32
Greedy	1.5744	15714.90	120.0651	44880.59	37.2985	31347.6200	0.1395	89696.83	121.0590	5631.67	121.0329	7604.67	0.8221	25864.00	0.7961s	30560.33
	±	±	±	± 7.132	±	±	±	± 1231.76	±	±	±	62.38	±	±	±	±
Ucom2-short	0.5376s	346.84	0.0268s	±	30.0300s	509.75	0.0526s	±	0.0318s	28.77	0.0613s	±	0.5700s	10223.00	0.6577s	11922.45
	0.0572s	15640.99	0.1210s	44597.56	0.1905s	31105.89	0.4609s	88685.40	0.7271s	5617.00	1.3196s	7630.00	0.3970s	25855.50	0.6553s	30542.33
Ucom2-medium	±	±	±	± 848.28	±	±	±	± 1225.67	±	±	±	72.19	±	±	±	±
	0.0502s	360.60	0.0563s	44208.95	0.0998s	506.23	0.1074s	88472.61	0.0251s	49.00	0.5774s	7594.67	0.2536s	11207.33	0.3883s	13055.67
Ucom2-long	1.0411s	15253.35	0.7392s	± 768.68	1.7342	29791.66	1.6216	± 1261.36	2.2988	5512.67	2.4537	68.72	17.8031	25833.00	17.3739	30526.00
	±	±	±	±	±	±	±	±	±	±	±	±	±	±	±	±
Ours-short	0.0827s	370.00	0.0150s	44852.82	0.0677s	678.02	0.0644s	89072.92	0.2026s	56.89	0.3920s	7618.00	18.4783s	11212.59	18.1940s	13043.86
	4.1891	15589.25	16.0005	± 765.11	9.5523	30420.69	8.9956	± 1248.58	18.2307	5560.33	17.1635	79.76	19.6806	25858.17	21.6114	30544.17
Ours-medium	±	±	±	±	±	±	±	±	±	±	±	±	±	±	±	±
	0.1150s	363.43	0.0408s	44906.50	0.0620s	552.66	0.089s	89408.28	2.4105s	44.56	2.2504s	81.19	19.8771s	11206.89	21.3478s	13052.91
Ours-long	39.2018	15679.45	31.7587	± 761.75	90.7475	30744.61	85.2663	± 1232.74	18.3035	5574.33	17.1988	7617.00	22.0589	25858.17	27.2170	30544.17
	±	±	±	±	±	±	±	±	±	±	±	±	±	±	±	±
Ours-short	0.9474s	358.41	0.0811s	±	0.3342s	512.61	0.6164s	±	2.4544s	58.79	2.2001s	±	21.6886s	11206.89	25.6604s	13052.91
	±	±	±	±	±	±	±	±	±	±	±	±	±	±	±	±
Ours-medium	0.5786s	15177.77	1.1078s	41247.80	0.8248s	29810.12	1.9076s	81357.29	1.8559s	5343.00	2.0838s	7411.67	1.0975s	7689.00	1.1485s	10227.00
	±	±	±	± 965.48	±	±	±	± 1277.84	±	±	±	55.08	±	±	±	±
Ours-long	0.0094s	344.68	0.0149s	43152.87	0.0312s	586.05	0.0284s	84186.79	0.0458s	89.17	0.0142s	7512.00	0.7446s	5712.56	0.7596s	5537.29
	1.0546s	15410.23	2.1212s	± 1004.95	1.4175s	30257.05	3.6669s	± 1336.48	2.5628s	5437.00	3.4385s	63.85	1.6693s	8887.83	1.9401s	11858.50
Ours-long	±	±	±	±	±	±	±	±	±	±	±	±	±	±	±	±
	0.0146s	353.91	0.0096s	44866.84	0.0633s	606.07	0.0582s	88796.03	0.0418s	66.84	0.0532s	7631.00	0.9268s	5087.20	1.0147s	4793.61
Ours-long	12.0878s	15714.63	24.9885s	± 876.78	16.6007s	31230.26	46.1200s	± 1327.83	27.0344s	5613.33	43.2777s	75.19	26.4401s	18526.83	54.4778s	24509.83
	±	±	±	±	±	±	±	±	±	±	±	±	±	±	±	±
Ours-long	0.1365s	354.22	0.3543s	±	0.4632s	476.25	0.6998s	±	0.9617s	55.58	1.3728s	±	15.9632s	8671.00	32.5091s	11171.33
	±	±	±	±	±	±	±	±	±	±	±	±	±	±	±	±

1179 H Max-Cut Ablation

1180 **Datasets.** We conduct experiments on two datasets. The IMDB-BINARY dataset [91] consists
 1181 of 1000 graphs. The Erdős–Rényi dataset consists of 1000 synthetic graphs generated under the
 1182 $G(n, p)$ model. For each graph, the number of nodes n is drawn uniformly from $\{50, \dots, 100\}$ and
 1183 each candidate edge is included independently with probability $p = 0.15$. Our experiments use the
 1184 following split for both datasets. 60% of the graphs are allocated for training, 20% for validation,
 1185 and the final 20% for testing. Within each split, any graphs with fewer than k nodes are excluded.
 1186 We compare in two settings, one of small and one of large k , where k is selected as a fraction of
 1187 the average number of nodes in the dataset. For small k we pick the fraction to be 0.25 of the node
 1188 average and for the large k we pick 0.75.

1189 **Direct optimization.** In the direct optimization approach, we assign vector $\mathbf{x} \in \mathbb{R}^n$ using the
 1190 perturbation method described in Section 4.1 and treat it as a perturbation from the uniform point (i.e.
 1191 $(k/n, k/n, \dots, k/n)$) in $\Delta_{n,k}$. We optimize it directly with Adam [39]. For IMDB-BINARY, we set
 1192 the learning rate to 0.015, and for Erdős–Rényi, the learning rate was set to 0.012. Each optimization
 1193 run consists of 150 update steps.

1194 **Neural net architecture.** For the network based approach, from the uniform vector $\mathbf{x} \in \mathbb{R}^n$ such
 1195 that $x_i = k/n$, we generate a perturbation vector as described in Section 4.1. The perturbed interior
 1196 point is the starting point in the hypercube for the decomposition, and gradient descent is used to
 1197 optimize the perturbation. Here, the neural network is two eight-layer GatedGraphConv networks [49]
 1198 followed by two linear layers. Node features include random-walk positional encodings generated
 1199 with AddRandomWalkPE [18] with walk length 10. The results are presented in the table below.

1200 **Standard SSL.** In the standard SSL baseline, we train the same architecture on the training data and
 1201 do model selection using the validation set. We observe that while the trained SSL baseline is not able
 1202 to match the performance of a neural net directly optimized on the test set, it consistently produces
 1203 performance that is competitive with direct optimization without a neural net and competitive with
 1204 greedy for large k .

Table 12: Test-set performance (mean \pm std) on IMDB-BINARY graphs (avg. 20 nodes) and Erdős–Rényi graphs (avg. 75 nodes, edge density 0.15).

Method	IMDB-BINARY		Erdős–Rényi	
	$k = 5$	$k = 15$	$k = 15$	$k = 60$
Greedy Algorithm	1.000 ± 0.002	0.801 ± 0.195	0.985 ± 0.015	0.900 ± 0.047
Random Sampling + Decomp	0.881 ± 0.140	0.850 ± 0.179	0.753 ± 0.038	0.761 ± 0.041
Direct optimization + Decomp	0.960 ± 0.064	0.922 ± 0.158	0.833 ± 0.025	0.844 ± 0.043
NN + Direct optimization + Decomp	0.971 ± 0.041	0.932 ± 0.089	0.910 ± 0.045	0.902 ± 0.041
Standard SSL + Decomp	0.956 ± 0.049	0.905 ± 0.117	0.899 ± 0.035	0.889 ± 0.044

1205 NeurIPS Paper Checklist

1206 1. Claims

1207 Question: Do the main claims made in the abstract and introduction accurately reflect the
1208 paper's contributions and scope?

1209 Answer: [Yes]

1210 Justification: We claim to provide an end-to-end pipeline for neural combinatorial optimiza-
1211 tion with constraints (section 4), which has applications to several fundamental constraint
1212 classes (section 4.2.1 and 4.2.2), all of which are clearly given in the paper. In section 5, we
1213 support our claims with experimental results.

1214 Guidelines:

- 1215 • The answer NA means that the abstract and introduction do not include the claims
1216 made in the paper.
- 1217 • The abstract and/or introduction should clearly state the claims made, including the
1218 contributions made in the paper and important assumptions and limitations. A No or
1219 NA answer to this question will not be perceived well by the reviewers.
- 1220 • The claims made should match theoretical and experimental results, and reflect how
1221 much the results can be expected to generalize to other settings.
- 1222 • It is fine to include aspirational goals as motivation as long as it is clear that these goals
1223 are not attained by the paper.

1224 2. Limitations

1225 Question: Does the paper discuss the limitations of the work performed by the authors?

1226 Answer: [Yes]

1227 Justification: The limitations are discussed in the experiment section and also in the con-
1228 clusion. Also, when we discuss the general pipeline we do not claim it covers all possible
1229 constrained CO problems.

1230 Guidelines:

- 1231 • The answer NA means that the paper has no limitation while the answer No means that
1232 the paper has limitations, but those are not discussed in the paper.
- 1233 • The authors are encouraged to create a separate "Limitations" section in their paper.
- 1234 • The paper should point out any strong assumptions and how robust the results are to
1235 violations of these assumptions (e.g., independence assumptions, noiseless settings,
1236 model well-specification, asymptotic approximations only holding locally). The authors
1237 should reflect on how these assumptions might be violated in practice and what the
1238 implications would be.
- 1239 • The authors should reflect on the scope of the claims made, e.g., if the approach was
1240 only tested on a few datasets or with a few runs. In general, empirical results often
1241 depend on implicit assumptions, which should be articulated.
- 1242 • The authors should reflect on the factors that influence the performance of the approach.
1243 For example, a facial recognition algorithm may perform poorly when image resolution
1244 is low or images are taken in low lighting. Or a speech-to-text system might not be
1245 used reliably to provide closed captions for online lectures because it fails to handle
1246 technical jargon.
- 1247 • The authors should discuss the computational efficiency of the proposed algorithms
1248 and how they scale with dataset size.
- 1249 • If applicable, the authors should discuss possible limitations of their approach to
1250 address problems of privacy and fairness.
- 1251 • While the authors might fear that complete honesty about limitations might be used by
1252 reviewers as grounds for rejection, a worse outcome might be that reviewers discover
1253 limitations that aren't acknowledged in the paper. The authors should use their best
1254 judgment and recognize that individual actions in favor of transparency play an impor-
1255 tant role in developing norms that preserve the integrity of the community. Reviewers
1256 will be specifically instructed to not penalize honesty concerning limitations.

1257 3. Theory assumptions and proofs

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

Answer: [Yes]

Justification: Assumptions are stated in theorem statements and all proofs are provided in the appendix.

Guidelines:

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

4. Experimental result reproducibility

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

Answer: [Yes]

Justification: All the details of the experiments such as training and dataset details are provided either in the main body of the paper or appendix.

Guidelines:

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

5. Open access to data and code

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

Answer: [Yes]

Justification: All the datasets are public. Our code is available https://anonymous.4open.science/r/Neural_Combinatorial_Optimization_with_Constraints-3485/README.md.

Guidelines:

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

6. Experimental setting/details

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

Answer: [Yes]

Justification: All the details of the experiments such as training and dataset details are provided either in the main body of the paper or appendix.

Guidelines:

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

7. Experiment statistical significance

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

Answer: [No]

Justification: We do not report error bars or statistical significance due to computational and time constraints; as is common in combinatorial optimization, we report mean results over multiple problem instances.

Guidelines:

- The answer NA means that the paper does not include experiments.

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

8. Experiments compute resources

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

Answer: [Yes]

Justification: Details are provided in the appendix.

Guidelines:

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

9. Code of ethics

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

Answer: [Yes]

Justification: Given the focus on foundations, we do not have human subjects or sensitive data and we do not perceive risks of harm or misuse.

Guidelines:

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

10. Broader impacts

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

Answer: [NA]

Justification: The societal impact of improved combinatorial optimization or neural combinatorial optimization techniques are minimal.

Guidelines:

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

11. Safeguards

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

Answer: [NA]

Justification: The paper does not contain data or models that have a high risk for misuse.

Guidelines:

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

12. Licenses for existing assets

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

Answer: [Yes]

Justification: We provide citations to all data and code we use. All material is licensed for such use.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. New assets

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

Answer: [NA]

Justification: There are no new assets released in this paper.

Guidelines:

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

14. Crowdsourcing and research with human subjects

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

Answer: [NA]

Justification: There were no human subjects in this work.

Guidelines:

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

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

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

Answer: [NA]

Justification: No human subjects or study participants were used in this research.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

- 1518 • Depending on the country in which research is conducted, IRB approval (or equivalent)
1519 may be required for any human subjects research. If you obtained IRB approval, you
1520 should clearly state this in the paper.
- 1521 • We recognize that the procedures for this may vary significantly between institutions
1522 and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the
1523 guidelines for their institution.
- 1524 • For initial submissions, do not include any information that would break anonymity (if
1525 applicable), such as the institution conducting the review.

1526 16. **Declaration of LLM usage**

1527 Question: Does the paper describe the usage of LLMs if it is an important, original, or
1528 non-standard component of the core methods in this research? Note that if the LLM is used
1529 only for writing, editing, or formatting purposes and does not impact the core methodology,
1530 scientific rigorousness, or originality of the research, declaration is not required.

1531 Answer: [NA]

1532 Justification: LLMs were not used in the core methods of this research.

1533 Guidelines:

- 1534 • The answer NA means that the core method development in this research does not
1535 involve LLMs as any important, original, or non-standard components.
- 1536 • Please refer to our LLM policy (<https://neurips.cc/Conferences/2025/LLM>)
1537 for what should or should not be described.