Unsupervised Learning for Combinatorial Optimization with Principled Objective Design

Anonymous Author(s)
Affiliation
Address
email

Abstract

Using machine learning to solve combinatorial optimization (CO) problems is challenging, especially when the data is unlabeled. This work proposes an unsupervised learning framework for CO problems. Our framework follows the standard relaxation-plus-rounding approach and adopts neural networks to parameterize the relaxed solutions so that simple back-propagation can train the model end-to-end. Our key contribution is the observation that if the relaxed objective satisfies entry-wise concavity, a low optimization loss guarantees the quality of the final integral solutions. This observation broadly generalizes the applicability of a previous framework inspired by Erdos' probabilistic method [1]. In particular, this observation can further guide the design of objective models in the applications where the objectives are not given explicitly while requiring being modeled in prior. We evaluate our framework by solving a synthetic graph optimization problem, and two real-world applications including resource allocation in circuit design and approximate computing. Our framework largely outperforms the baselines based on naïve relaxation, reinforcement learning and Gumbel-softmax tricks.

1 Introduction

Combinatorial optimization (CO) with the goal of finding the optimal solution from a discrete space is a fundamental problem in many scientific and engineering applications [2–4]. Most CO problems are NP-complete. Traditional methods efficient in practice often use heuristics or produce approximation solutions. Designing these approaches requires considerable insights into the problem. Recently, machine learning has revolutionized this traditional way to develop CO algorithms by leveraging neural networks (NNs) to extract heuristics from the data [5–7]. Promising learning for CO (LCO) approaches have already been developed for SAT [8–10], mixed integer linear programming [11–13], vertex covering [14,15] and routing problems [16–23].

Another promising, if not more promising usage of machine learning techniques is to assist the applications where the evaluation of the CO objective for each tentative solution could be expensive and time-consuming [24–27]. For example, in hardware/system design, the actual computation latency, power efficiency [28], and resource consumption [29–31] cannot be known before running complex simulators. Also, in molecule design, the desired properties such as protein fluorescence or DNA binding may only get evaluated via costly simulations or living experiments [32–34]. Therefore, proxies of their objectives need to be learned at first. Then, solving these Proxy-based CO (PCO) can be further conducted by following traditional LCO schemes [31]. Note that learning for PCO is even in greater need compared to traditional CO problems because commercial CO solvers such as Gurobi can never be applied in PCO due to the in-availability of closed-form objectives. Generic solvers such as simulated annealing [35] may be applied while they could be extremely slow.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

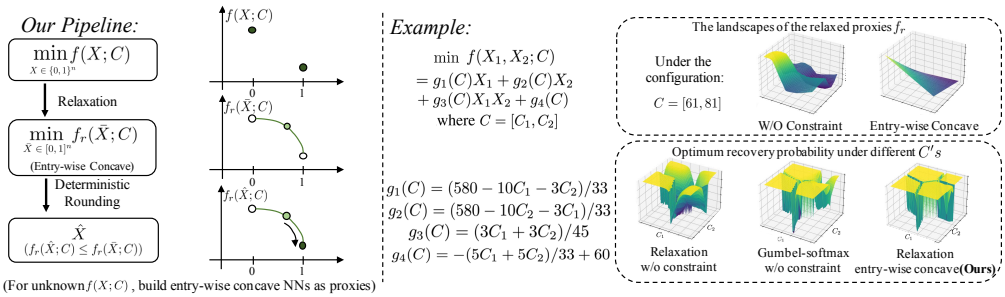


Figure 1: The pipeline and an example. The optimization objective $f(X;C)$, $X \in \{0,1\}^n$ is relaxed to $f_r(\bar{X};C)$, $\bar{X} \in [0,1]^n$. Here, C is the problem configuration such as an attributed graph in a graph optimization problem. The entry-wise concave structure of f_r provides a performance guarantee in our deterministic rounding procedure (in Def. 1). When $f(X;C)$ is not explicitly given, f_r will be modeled as NNs and learned. In the toy example, we first learn $f(\bar{X};C)$ by proxies with or without the entry-wise concave constraint. We compare their landscapes in the top-right figure. We further optimize both proxies and round the obtained soft solutions to integral solutions. The bottom-right figure shows the optimum-recovery probabilities of different methods under different C's.

In this work, we are to study the unsupervised LCO framework. Our findings are applied to general CO problems while exhibiting extraordinary promise for PCO problems. Unsupervised LCO has recently attracted great attentions [1,9,20,36,37]. Compared to supervised learning that gets criticized for the dependence on huge amounts of labeled data [38], and reinforcement learning (RL) that suffers from notoriously unstable training [39], unsupervised LCO has shown great advantages due to its faster training, good generalization via accessing large unlabeled data, and its capability of dealing with large-scale problems [1]. On the other hand, for PCO problems, unsupervised learning has never been investigated before. Previous works for PCO problems were all on RL, e.g., hardware design [30,31]. This work provides the first unsupervised framework for PCO problems.

Our framework follows a relaxation-plus-rounding approach. We optimize a carefully-designed continuous relaxation of the cost model (penalized with constraints if any) and obtain a soft solution. Then, we decode the soft assignment to have the final discrete solution. This follows a common approach in traditional approximation algorithm design [40,41]. However, the strategy here to obtain soft assignments is based on a model optimized based on the historical (unlabeled) data via gradient descent. Learning from historical data is expected to facilitate the model understanding the data distribution, which helps with avoiding local minima and achieving fast inference. An illustration of the framework with a toy-example is shown in Fig. 1.

Our method shares a similar spirit with [1] while making the following significant contributions. We abandon the probabilistic guarantee in [1], because it is hard to use when we have general or proxy-based objectives. Instead, we design a deterministic objective relaxation principle that allows performance guarantee. We prove that if the objective relaxation is entry-wise concave w.r.t. the binary optimization variables, a low-cost soft solution plus deterministic sequential decoding guarantees generating a valid and low-cost integral solution. This principle significantly broadens the applicability of this unsupervised learning framework. In particular, it guides the design of model architectures to learn the objectives in PCO problems. We further justify the wide applicability of the entry-wise concave principle in both theory and practice.

We evaluate our framework over three PCO applications including a feature-based edge covering & node matching problems, and two real-world applications, imprecise functional unit assignment in approximate computing (AxC) [42–46] and resource allocation in circuit design [24,30]. In all three applications, we observe a significant performance boost of our framework compared to previous RL-based approaches and relaxed gradient approaches based on the Gumbel-softmax trick [47–49].

1.1 Further Discussion on Related Works

Most previous LCO approaches are based on RL [13,14,16–18,22,50–52] or supervised learning [11,12,38], as these two frameworks do not hold much constraints on the formulation of CO problems.

However, they often suffer from the issues of training instability and subpar generalization. Previous works on unsupervised learning for CO have studied satisfaction problems [9,36], while applying them to general CO problems requires problem reductions. Others have considered max-cut [37] and TSP problems [20], while these works depend on carefully selected problem-specific objectives. The work most relevant to ours is [1] and we give detailed comparison in Sec. 3. Note that all previous works on unsupervised learning for CO do not apply to PCO as they need an explicit objective to manipulate. For PCO problems, previous studies focus on how to learn more generalizable proxies of the costs, such as via Bayesian learning [53,54] and adverserial training [55,56]. Once proxies are learned, direct objective relaxation [55] or RL [30,31] is often adopted. Studying generalization of proxies is out of the scope of this work while entry-wise concave proxies seem smoother than those without constraints (See Fig. 1) and thus have the potential to be more generalizable.

2 Preliminaries and Problem Formulation

In this section, we define several useful concepts and notations.

Combinatorial Optimization (CO). Let $C \in \mathcal{C}$ denote a data-based configuration such as a graph with weighted edges. Let Ω be a finite set of all feasible combinatorial objects and each object has a binary vector embedding $X = (X_i)_{1 \le i \le n} \in \{0,1\}^n$. For example, in the node matching problem, each entry of X corresponds to an edge to denote whether this edge is selected or not. Note that such binary embeddings are applicable even when the choice is not naturally binary: Choosing at most one element from a tuple $(1,2,3)$ can be represented as a 3-dim binary vector (X_1, X_2, X_3) with the constraint $X_1 + X_2 + X_3 \le 1$. W.l.o.g, we assume an algebraic form of the feasible set $\Omega \triangleq \{X \in \{0,1\}^n : g(X;C) < 1\}$ where $g(X;C) \ge 0$ for all $X \in \{0,1\}^n$ [1]. For notational simplicity, we only consider one inequality constraint while our later discussion in Sec. 3 and our case studies in Sec. 4 may contain multiple inequalities. Given a configuration C and a constraint Ω, a combinatorial optimization (CO) is to minimize a cost $f(\cdot;C)$ by solving

$$\min_{X \in \{0,1\}^n} f(X;C), \quad \text{s.t.} \quad g(X;C) < 1. \tag{1}$$

Proxy-based CO (PCO). In the many applications, the cost or the constraint may not be cheaply evaluated. Some proxies of the cost f or the constraint g often need to be learned from the historical data. With some abuse of notations, we interchangably use f (g, resp.) to denote the objective (the constraint, resp.) and its proxy.

Learning for CO/PCO (LCO). A LCO problem is to learn an algorithm $\mathcal{A}_\theta(\cdot) : \mathcal{C} \to \{0,1\}^n$, say a neural network (NN) parameterized by θ to solve CO or PCO problems. Given a configuration $C \in \mathcal{C}$, we expect $\mathcal{A}_\theta$ to (a) generate a valid solution $\hat{X} = \mathcal{A}_\theta(C) \in \Omega$ and (b) minimize $f(\hat{X};C)$.

There are different approaches to learn $\mathcal{A}_\theta$. Our focus is unsupervised learning approaches where given a configuration C, the ground-truth solution X^* is not accessible during the training. θ can only be optimized based on the knowledge of the cost and the constraint, or their proxies.

Erdős' Probabilistic Method (EPM). The EPM has recently been brought for LCO [1]. Specifically, The EPM formulates $\mathcal{A}_\theta(C)$ as a randomized algorithm that essentially gives a probabilistic distribution over the solution space $\{0,1\}^n$, which solves the optimization problem:

$$\min_\theta \quad \mathbb{E}\left[l(X;C)\right], \text{ where } l(X;C) \triangleq f(X;C) + \beta 1_{g(X;C) \ge 1},\ X \sim \mathcal{A}_\theta(C) \text{ and } \beta > 0\,. \tag{2}$$

Karalias & Loukas proved that with $\beta > \max_{X \in \Omega} f(X;C)$ and a small expected loss $\mathbb{E}\left[l(X,C)\right] < \beta$, sampling a sufficiently large number of $\hat{X} \sim \mathcal{A}_\theta(C)$ guarantees the existance of a feasible $\hat{X} \in \Omega$ that achieves the cost $f(\hat{X};C) \le \mathbb{E}\left[l(X,C)\right]$ [1]. Although this guarantee makes EPM intriguing, applying EPM in practice is non-trivial. We will explain the challenge in Sec. 3.1, which inspires our solutions and further guides the objective design for general CO and PCO problems.

[1] Normalization $(g(\cdot;C) - g_{\min})/(g^+_{\min} - g_{\min})$ where $g^+_{\min} = \min_{X \in \{0,1\}^n \setminus \Omega} g(X;C)$ and $g_{\min} = \min_{X \in \{0,1\}^n} g(X;C)$ always satisfies the property. $g^+_{\min}$, $g_{\min}$ often can be easily estimated in practice.

3 The Relaxation Principle for Unsupervised LCO

In this section, we start with the practical issues of EPM. Then, we introduce our solutions by proposing a relaxation principle of the objectives, which gives performance guarantee for general practical unsupervised LCO.

3.1 Motivation: The Practical Issues of EPM

Applying EPM in practice has two fundamental difficulties. First, optimizing θ in Eq.(2) is generally hard as the gradient $\frac{dX}{d\theta}$ does not generally exist so the chain rule cannot be used. We discuss the potential solutions to this problem in Sec. 3.4. Second, EPM needs to sample a large number of $X \sim \mathcal{A}_\theta(C)$ for evaluation to achieve the performance guarantee in [1]. This is not acceptable where the evaluation per sample is time-consuming and expensive.

So, in practice, Karalias & Loukas consider a deterministic method. They view $\mathcal{A}_\theta(C) \in [0,1]^n$ as the parameters of Bernouli distributions to generate the entries of X so $\mathbb{E}[X] = \mathcal{A}_\theta(C)$. First, they optimize $\min_\theta l(\mathcal{A}_\theta(C), C)$ instead of $\min_\theta \mathbb{E}[l(X, C)]$, and then, sequentially round the probability $\mathcal{A}_\theta(C)$ to discrete $X \in \{0,1\}^n$ by comparing conditional expectations, e.g., $\mathbb{E}[l(X,C)|X_1 = 0]$ v.s. $\mathbb{E}[l(X,C)|X_1 = 1]$ to decide X_1. However, such conditional expectations for general l cannot be efficiently computed unless one uses Monte-Carlo sampling. The two special cases in [1] on the max-clique and graph-partition problems seem to have special structures. This blocks the applicability of this framework, especially for the PCO problems where the objectives l are learned as models.

3.2 Our Approach: Relaxation plus Rounding, and Performance Guarantee

Our solution does not use the probabilistic modeling but directly adopts a relaxation-plus-rounding approach. We optimize a relaxation of the objective l_r and obtain a soft solution $\bar{X} \in [0,1]^n$. Then, we deterministically round the entries in $\bar{X}$ to a solution in the discrete space $\{0,1\}^n$. The question is whether the obtained solutions may still achieve the guarantee as EPM does. Our key observation is that such success essentially depends on how to relax the objective l.

Therefore, our first contribution beyond [1] is to propose the principle (Def. 2) to relax general costs and constraints. With this principle, the unsupervised LCO framework can deterministically yield valid and low-cost solutions (Thm. 1) as the EPM guarantees, and is applied to any objective l.

First, we introduce the pipeline. Consider a relaxation of a deterministic upper bound of Eq.(2):

$$\min_\theta \; l_r(\theta; C) \triangleq f_r(\bar{X}; C) + \beta g_r(\bar{X}; C), \text{ where } \bar{X} = \mathcal{A}_\theta(C) \in [0,1]^n \text{ , } \beta > 0. \tag{3}$$

Here $f_r(\cdot; C) : [0,1]^n \to \mathbb{R}$ is the relaxation of $f(\cdot; C)$, which satisfies $f_r(X; C) = f(X; C)$ for $X \in \{0,1\}^n$. The relation between g_r and g is similar, i.e., $g_r(X; C) = g(X; C)$ for $X \in \{0,1\}^n$. Here, we also use the fact that $g_r(X; C)$ provides a natural upper bound $1_{g(X;C)\geq 1} \leq g_r(X; C)$ for $X \in \{0,1\}^n$ given the normalization of $g(X; C)$ adopted in Sec. 2.

Now, suppose the parameter θ gets optimized so that $l_r(\theta; C)$ is small. Further, we adopt the sequential rounding in Def. 1 to adjust the continuous solution $\bar{X} = \mathcal{A}_\theta(C)$ to discrete solution X.

Definition 1 (Rounding)**.** *Given a continuous vector $\bar{X} \in [0,1]^n$ and an arbitrary order of the entries, w.o.l.g., $i = 1, 2, ..., n$, round $\bar{X}_i$ into 0 or 1 and fix all the other variables un-changed. Set $X_i = \arg\min_{j=0,1} f_r(X_1, ..., X_{i-1}, j, \bar{X}_{i+1}, ..., \bar{X}_n; C) + \beta g_r(X_1, ..., X_{i-1}, j, \bar{X}_{i+1}, ..., \bar{X}_n; C)$, replace $\bar{X}_i$ with X_i and repeat the above procedure until all the variables become discrete.*

Note that our rounding procedure does not need to evaluate any conditional expectations $\mathbb{E}[l(X; C)|X_1]$ which EPM in [1] requires. Instead, we ask both relaxations f_r and g_r to satisfy the principle in Def. 2. With this principle, the pipeline allows achieving a valid and low-cost solution X, as proved in Theorem 1. We leave the proof in Appendix A.1.

Definition 2 (The Entry-wise Concave Principle)**.** *For any $C \in \mathcal{C}$, $h_r(\cdot; C) : [0,1]^n \to \mathbb{R}$ is entry-wise concave if for any $\gamma \in [0,1]$ and any $\bar{X}, \bar{X}' \in [0,1]^n$ that are only different in one entry,*

$$\gamma h_r(\bar{X}; C) + (1-\gamma) h_r(\bar{X}'; C) \leq h_r(\gamma \bar{X} + (1-\gamma)\bar{X}'; C).$$

Note that entry-wise concavity is much weaker than concavity. For example, the function $h_r(\bar{X}_1, \bar{X}_2) = -\text{Relu}(\bar{X}_1\bar{X}_2)$, $\bar{X}_1, \bar{X}_2 \in \mathbb{R}$ is entry-wise concave but not concave.

Theorem 1 (Performance Guarantee)**.** *Let $\beta > \max_{X\in\Omega} f(X;C)$ and $\min_{X\in\Omega} f(X;C) \geq 0$ in Eq.(3). Suppose the relaxed cost f_r and constraint g_r are entry-wise concave, and the learned parameter θ achieves $l_r(\theta;C) < \beta$. Then, rounding (Def. 1) the relaxed solution $\bar{X} = \mathcal{A}_\theta(C)$ generates a valid discrete solution $X \in \Omega$ such that $f(X;C) < l_r(\theta;C)$.*

When there are multiple normalization constraints $g^{(j)}(X;C) < 1$ for $j = 1, 2, ...$, we may use relaxation $\beta \sum_j g_r^{(j)}(X;C)$ as the penalty term in Eq.(3), where $g_r^{(j)}$ is a relaxation of $g^{(j)}$. It can be shown that if $\sum_j g_r^{(j)}$ satisfies the entry-wise concave condition, the guarantee of Thm. 1 still applies.

3.3 The Wide Applicability of Entry-wise Concave Relaxations

We have introduced the entry-wise concave principle to relax the objective to associate our framework with performance guarantee. The question is how widely applicable this principle could be.

Actually, every function with binary inputs can be relaxed as an entry-wise affine function with the exactly same values at the discrete inputs, as shown in Theorem 2. Note that entry-wise affinity is a special case of entry-wise concavity. In Sec. 4, we will provide the design of NN architecture (for PCO) and math derivation (for CO) that guarantee formulating an entry-wise concave function.

Theorem 2 (Wide Applicability)**.** *For any binary-input function $h(\cdot) : \{0,1\}^n \to \mathbb{R}$, there exists a relaxation $h_r(\cdot) : [0,1]^n \to \mathbb{R}$ such that (a) $h_r(X) = h(X)$ for $X \in \{0,1\}^n$ and (b) h_r is entry-wise affine, i.e., for any $\gamma \in [0,1]$ and any $\bar{X}, \bar{X}' \in [0,1]^n$ that are only different in one entry,*

$$\gamma h_r(\bar{X}) + (1-\gamma)h_r(\bar{X}') = h_r(\gamma\bar{X} + (1-\gamma)\bar{X}').$$

Proof sketch. Set $h_r(\bar{X}) = \sum_{X\in\{0,1\}^n} h(X) \prod_{j=1}^n \bar{X}_j^{X_j}(1-\bar{X}_j)^{(1-X_j)}$, which satisfies (a) and (b). The detailed proof is in Appendix A.2. □

Although Theorem 2 shows the existence of entry-wise affine relaxations, the constructed representation in the proof depends on higher-order moments of the input entries, which make it often intractable to implement, especially via a NN architecture. Therefore, we also propose to use entry-wise concave functions to implicitly generate higher-order moments. For example, when $n = 2$, we could use the composition of $-\text{Relu}(\cdot)$ and affine operators (only 1st-order moments) to achieve universal representation (See Prop. 1 and the proof in Appendix A.3). For general n, we leave as a future study.

Proposition 1. *For any binary-input function $h(X_1, X_2)$, there exists parameters $\{w_{ij}\}$ such that an entry-wise concave function $h_r(\bar{X}_1, \bar{X}_2) = w_{00} - \sum_{i=1}^3 \textit{Relu}(w_{i1}\bar{X}_1 + w_{i2}\bar{X}_2 + w_{i0})$ satisfies $h_r(X_1, X_2) = h(X_1, X_2)$ for any $X_1, X_2 \in \{0,1\}$.*

3.4 Discussion: Methods to Directly Optimize the Randomized Objective in EPM Eq.(2)

The naïve way to optimize the randomized objective in Eq.(2) without worrying about the specific form of the objective l is based on the policy gradient in RL via the logarithmic trick, i.e., estimating the gradient $\frac{dl}{d\theta}$ via $(f(X;C) + \beta 1_{g(X;C)\geq 1}) \log \mathbb{P}(X)$ by sampling $X \sim \mathcal{A}_\theta(C)$. However, the policy gradient suffers from notoriously large variance [39] and makes RL hard to converge. Therefore, methods such as actor critic [57] or subtracting some baselines $l(X;C) - b$ [58] have been proposed.

	RL	Gumbel-softmax	Ours
Objective	No Limit	No Limit	Entry-wise Concave
Optimizer	Log Trick	Gumbel Trick	No Limit
Inference	Sampling	Sampling	Deter. Rounding
Train. Time	Slow	Fast	Fast
Convergence	Hard	Medium	Easy
Infer. Time	Slow	Slow	Fast

Table 1: The comparison among RL (policy gradient), Gumbel-softmax methods and our principled objective relaxation. Our methods are in need of much less training time and inference time.

Another way to solve Eq.(2) is based on reparameterization tricks to reduce the variance of gradients [59, 60]. Specifically, we set the entries of output $\bar{X} = \mathcal{A}_\theta(C) \in [0,1]^n$ as the parameters of Bernoulli distributions to generate X, i.e., $X_i \sim \text{Bern}(\bar{X}_i)$, for $1 \leq i \leq n$. To make $dX_i/d\bar{X}_i$

computable, we may use the Gumble-softmax trick [47–49]. However, this approach suffers from two issues. First, the estimation of the gradient is biased. Second, as $\mathcal{A}_\theta(C)$ is essentially a randomized algorithm, sampling sufficiently many $X \sim \mathcal{A}_\theta(C)$ is needed to guarantee a valid and low-cost solution. However, such evaluation is costly as discussed in Sec. 3.1. We compare different aspects of RL, Gumbel-softmax tricks and our relaxation approach in Table 1.

4 Applying Our Relaxation Principle to Learning for PCO

In this section, we apply our relaxation principle to three PCO applications: (I) feature-based edge covering & node matching, (II) resource allocation in circuit design, and (III) imprecise functional unit assignment in approximate computing. All the applications have graph-based configurations C. So later, we first introduce how to use graph neural networks (GNNs) to build proxies that satisfy our relaxation principle. Such GNN-based proxies will be used as the cost function relaxation f_r in all the applications. Our principle can also guide the relaxation of explicit CO objectives. The constraints in applications (I)(III) are explicit and their relaxation can be written into the entry-wise affine form. The constraint in (II) needs another GNN-based entry-wise concave proxy to learn.

4.1 GNN-based Entry-wise Concave Proxies

We consider the data configuration C as an attributed graph (V, E, Z) where V is the node set, $E \subseteq V \times V$ is the edge set and Z is the node attributes. We associate each node with a binary variable and group them together $X :\in \{0,1\}^{|V|}$. where for each $v \in V$, $X_v = 1$ indicates the choice of the node v. Note that our approach can be similarly applied to edge-level variables (see Appendix C.2), which is used in application (I). Let $\bar{X}$ still denote the relaxation of X.

To learn a discrete function $h : \{0,1\}^{|V|} \times \mathcal{C} \to \mathbb{R}$, we adopt a GNN as the relaxed proxy of h. We first define a latent graph representation in $\mathbb{R}^F$ whose entries are all entry-wise affine mappings of X.

$$\textbf{Latent representation:} \qquad \phi(\bar{X}; C) = W + \sum_{v \in V} U_v \bar{X}_1 + \sum_{v,u \in V, (v,u) \in E} Q_{v,u} \bar{X}_v \bar{X}_u \tag{4}$$

where W is the graph representation, U_v's are node representations and $Q_{v,u}$ are edge representations. These representations do not contain X and are given by GNN encoding C. Here, we consider at most 2nd-order moments based on adjacent nodes as they can be easily implemented via current GNN platforms [61,62]. Then, we use ϕ to generate entry-wise affine & concave proxies as follows.

$$\textbf{Entry-wise Affine Proxy (AFF):} \qquad h_r^{\mathrm{a}}(\bar{X}; C) = \langle w^a, \phi(\bar{X}; C) \rangle. \tag{5}$$

$$\textbf{Entry-wise Concave Proxy (CON):} \qquad h_r^{\mathrm{c}}(\bar{X}; C) = \langle w^c, -\mathrm{Relu}(\phi(\bar{X}; C)) \rangle + b. \tag{6}$$

where $w^a, w^c \in \mathbb{R}^F, b \in \mathbb{R}$ are learnt parameters and $w^c \geq 0$ guarantees entry-wise concavity.

4.2 The Setting up of the Experiments

Training & Evaluation Pipeline. In all the applications, we adopt the following training & evaluation pipeline. First, we have a set of observed configurations $\mathcal{D}_1 \subset \mathcal{C}$. Each $C \in \mathcal{D}_1$ is paired with one $X \in \{0,1\}^n$. We use the costs $f(X, C)$ (and constraints $g(X, C)$) to train the relaxed proxies $f_r(X, C)$ (and $g_r(X, C)$, if cannot be derived explicitly), where the relaxed proxies follow either Eq.(5) (named AFF) or Eq.(6) (named CON). Then, we parameterize the LCO algorithm $\mathcal{A}_\theta(C) \in [0,1]^n$ via another GNN. Based on the learned (or derived) f_r and g_r, we optimize θ by minimizing $\sum_{C \in \mathcal{D}_1} l_r(\theta; C)$, where l_r is defined according to Eq.(3). We will split $\mathcal{D}_1$ into a training set and a validation set for hyperparameter-tuning of proxies and $\mathcal{A}_\theta$. We have another set of configurations $\mathcal{D}_2 \subset \mathcal{C}$ used for testing. For each $C \in \mathcal{D}_2$, we use the relaxation $\bar{X} = \mathcal{A}_\theta(C)$ plus our rounding to evaluate the learned algorithm $\mathcal{A}_\theta(\cdot)$. We follow [1] and do not consider fine-tuning $\mathcal{A}_\theta$ over the testing dataset $\mathcal{D}_2$ to match the potential requirement of the fast inference.

Baseline	f_r, g_r	$\mathcal{A}_\theta$	Inference
Naïve + R	no limit	no limit	rounding
RL	no limit	RL	sampling
GS-Tr+S	no limit	GS	sampling
GS-Tr+R	no limit	GS	rounding

Table 2: The baselines in the paper.

Baselines. We consider 4 common baselines that is made up of different learnable relaxed proxies f_r, g_r, algorithms $\mathcal{A}_\theta$ and inference approaches as shown in Table 2. For the proxies f_r, g_r for

baselines, we apply GNNs without the entry-wise concave constraint and use X as one node attribute while keeping all other hyper-parameters exactly the same as ours (See details in Appendix. C); For the algorithm $\mathcal{A}_\theta$, we provide the Gumbel-softmax trick based methods (GS-Tr) [48, 49], the actor-critic-based RL method [57] (RL) and the naïve relaxation method (Naïve); For the inference approaches, we consider Monte Carlo sampling (S) and our proposed rounding (R) procedure. Although the baselines adopt proxies that are different from ours, we guarantee that their proxies approximate the ground-truth f, g over the validation dataset at least no worse than ours. In application II, we also consider two non-learnable algorithms to optimize the proxies without relaxation constraints, simulated annealing (SA) [35] and genetic algorithms (GA) [63, 64]. In application III, we put all of the required AxC units either close to the input (C-In) or close to the output (C-Out) of the approximating computing circuit as additional baselines. More details of the experiments setups and hyperparameter tuning can be found in Appendix C. We also obtain the optimal solutions (OPT) for applications I and III via brute-force search for comparison.

4.3 Application I: Feature-based Edge Covering & Node Matching in Graphs

This application is inspired by [65]. Here, each configuration C is a 4×4 grid graph whose node attributes are two-digit images generated by random combinations of the pictures in MNIST [66]. We associated each edge with variables $X \in \{0,1\}^{|E|}$. The objective is the sum of edge weights $f(X;C) = \sum_{e \in E} w_e X_e$ where w_e is unknown in prior and needed to be learned. The ground truth of w_e is a multiplication of the numbers indicated by the images on the two adjacent nodes. We adopt ResNet-50 [67] (to refine node features) plus GraphSAGE [68] to encode C. We consider using both Eq.(5) and Eq.(6) to formulate the relaxed cost $f_r(\bar{X};C)$. Training and validating f_r are based on 100k randomly sampled C paired with randomly sampled X. Note that 100k is much smaller than the entire space $\{0,1\}^{|E|} \times \mathcal{C}$ is of size $2^{24} \times 100^{16}$.

Next, as the constraint here is explicit, we can derive the relaxation of the constraints for this application. First, the constraint relaxation of the edge covering problem can be written as

$$\textbf{Edge Covering Constraint:} \quad g_r(\bar{X};C) = \sum_{v \in V} \prod_{e: v \in e} (1 - \bar{X}_e). \tag{7}$$

Each production term in Eq.(7) indicates that for each node, at least one edge is selected. We can easily justify that g_r is entry-wise affine and $\Omega = \{X \in \{0,1\}^{|E|} : g_r(X;C) < 1\}$ exactly gives the feasible solutions to the edge covering problem.

Similarly, we can derive the constraint for node matching by adding a further term to Eq.(7).

$$\textbf{Node Matching Constraint:} \; g_r(\bar{X};C) = \sum_{v \in V} [\prod_{e: v \in e} (1 - \bar{X}_e) + \prod_{e_1, e_2: v \in e_1, e_2, e_1 \neq e_2} \bar{X}_{e_1} \bar{X}_{e_2}]. \tag{8}$$

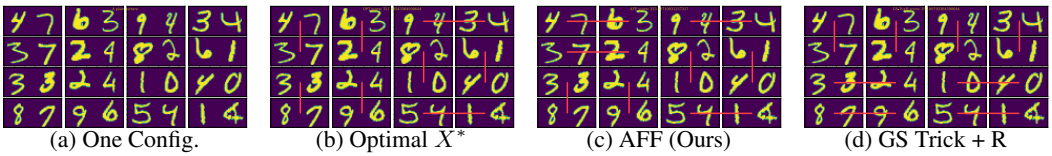
Here, the second term indicates that no two edges adjacent to the same node can be selected. We can easily justify that g_r is entry-wise affine and $\Omega = \{X \in \{0,1\}^{|E|} : g_r(X;C) < 1\}$ groups exactly the feasible solutions to the node matching problem.

Note that our above derivation also generalizes the node-selection framework in [1] to edge selection. With the learned f_r and the derived g_r, we further train and validate $\mathcal{A}_\theta$ over the 100k sampled (X, C)'s and test on another 500 randomly sampled C's.

Evaluation. Table 3 shows the evaluation results. In the GS-Tr+S method, the number of sampling is set to 120 (about 2.5 times the inference time of our deterministic rounding). Note that for node matching, GS-Tr+S could hardly sample a feasible node matching solution within 120 samples. The experiment results show that our principled proxy design exceeds the other baselines on both tasks. Also, we observe that AFF outperforms CON, which results from the fact that $f(X;C)$ in these two problems are naturally in entry-wise affine forms

Method	Edge covering	Node matching
Naive+R	68.52	429.12
RL	51.29	426.97
GS-Tr+S	63.36	-
GS-Tr+R	46.91	429.39
CON(ours)	49.59	422.47
AFF(ours)	**44.55**	**418.96**
OPT(gt)	42.69	416.01

Table 3: Performance on application I (graph optimization).



(a) One Config. (b) Optimal X^* (c) AFF (Ours) (d) GS Trick + R

Figure 2: The visualization for node matching in Application I. Our method avoids large multiplications $87 * 96$ and $94 * 82$ where GS-Trick cannot, and generate a solution different but close to OPT.

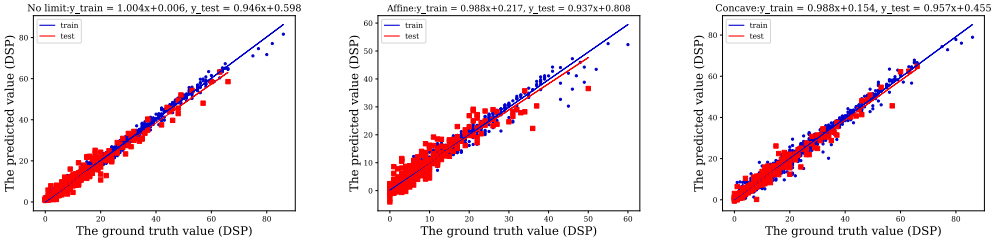


Figure 3: Comparing different proxies for learning DSP usage. Left, no constraint; Middle, entry-wise affine constraint (Eq. (5)); Right, entry-wise concave constraint (Eq.(6))

with low-order (1st-order) moments. One instance of node matching randomly selected from the test set is shown in Fig. 2. More visualization results can be found in Fig. 5 in the appendix.

4.4 Application II: Resource Allocation in Circuit Design

Resource allocation in field-programmable gate array (FPGA) design is a fundamental problem which can lead to largely varied circuit quality after synthesis, such as area, timing, and latency. In this application, we follow the problem formulation in [24, 30], where the circuit is represented as a data flow graph (DFG), and each node represents an arithmetic operation such as multiplication or addition. The goal is to find a resource allocation for each node to be either digital signal processor (DSP) or look-up table (LUT), such that the final circuit area (i.e., actual DSP and LUT usage) after synthesis is minimized. Notably, different allocation solutions result in greatly varied DSP/LUT usage due to complicated synthesis process, which cannot be simply summed up over each node. To obtain precise DSP/LUT usage, one must run high-level synthesis (HLS) [69] and place-and-route [70] tools, which can take up to hours [24, 30].

In this application, each configuration C is a DFG with > 100 nodes, where each node is allocated to either DSP or LUT. Node attributes include operation type (i.e., multiplication or addition) and data bitwidth. More details about the dataset can be found in Appendix C.4. Let $X \in \{0, 1\}^{|V|}$ denote the mapping to DSP or LUT. Let f_r and g_r denote the proxies of actual LUT and actual DSP usage, respectively. Note that given different constraints on the DSP usage, we will normalize g_r as introduced in Sec. 2. We train and validate $f_r, g_r, \mathcal{A}_\theta$ on $8,000$ instances that consist of 40 DFGs (C), each DFG with 200 different mappings (X), and test $\mathcal{A}_\theta$ over 20 DFGs. Note that the actual LUT and DSP usages of each training instance has been collected by running HLS in prior. We also run HLS to evaluate the actual LUT and DSP usages for the testing cases given the learned mappings.

Evaluation. We rank each method's best actual LUT usage under the constraint of different percentages (40% - 70%) of the maximum DSP usage in each testing instance, then calculate the averaged ranks. Fig. 4 shows the results. Our entry-wise concave proxy achieves the best performance. GS-Tr+R is slightly better than RL, and both of them exceed SA and GA. We do not include our entry-wise affine proxy in the ranking list, because the affine proxy could be much less accurate than the proxy without constraints and the entry-wise concave proxy. The comparison between these proxies on learning DSP usage (& LUT usage) is shown in Fig. 3 (& Fig. 7 in the appendix, respectively). The gap between different proxies indicates the FPGA circuit contains high-order moments of the input optimization variables and 2-order entry-wise affine proxy cannot model well. We do not include the result of GS-Tr+S and Naive+R, because these methods perform poor and could hardly generate feasible solutions given a constraint of DSP usage. We leave their results in Table. 5 in the appendix. Moreover, we compare the training time between different methods. To

DSP usage	40%	45%	50%	55%	60%	65%	70%	rank-avg
SA	3.50	3.25	3.42	3.17	3.50	4.08	4.00	3.56
GA	2.70	2.92	3.17	3.08	3.42	3.25	3.25	3.11
RL	3.20	3.67	3.67	3.17	2.83	2.58	**2.33**	3.06
GS-Tr+R	3.50	3.00	2.50	3.08	**2.17**	**2.50**	2.75	2.79
CON	**2.10**	**2.17**	**2.25**	**2.25**	3.00	**2.50**	2.50	**2.40**

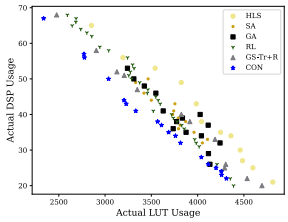


Figure 4: The left table shows averaged ranks of the LUT usage given by different methods with the constraint of different percentage of DSP usage in Application II (resource allocation). The right figure shows the DSP-LUT usage amount relationship on one test configuration.

Threshold θ	C-In	C-Out	Naïve	RL	GS-Tr+S	GS-Tr+R	CON	AFF	OPT
3 AxC units	12.42	12.44	10.59	3.62	4.87	3.24	3.18	**3.10**	2.77
5 AxC units	14.68	14.65	12.28	6.20	8.03	5.86	**5.13**	5.38	4.74
8 AxC units	17.07	17.04	15.17	11.12	12.65	10.62	10.17	**10.04**	8.56

Table 4: Relative errors of different methods with the AxC unit constraint as 3,5,8 in Application III.

be fair, all methods run on the same server with a Quadro RTX 6000 GPU. The RL based optimizer takes 22 GB GRAM, while other optimizers only take 7 GB on average. Fig. 8 in the appendix further demonstrates that our methods and GS-T methods require much less training time than RL.

4.5 Application III: Imprecise Functional Unit Assignment in Approximate Computing

One fundamental problem in approximate computing (AxC) is to assign imprecise functional units (a.k.a., AxC units) to execute operations such as multiplication or addition [42–46], aiming to significantly reduce circuit energy with tolerable error. We follow the problem formulation in [45], where given a computation graph, each node represents either multiplication or addition. The incoming edges of a node represent its two operands. The goal is to assign AxC units to a certain number of nodes while minimizing the expected relative error of the output of the computation graph.

In this application, each configuration C is a computation graph with 15 nodes (either multiplication or addition) that maps a vector in $\mathbb{R}^{16}$ to $\mathbb{R}$. A fixed number θ of nodes are assigned to AxC units with produce 10% relative error. Let $X \in \{0,1\}^{|V|}$ denote whether a node is assigned to an AxC unit or not; the proxy of the objective f_r is the expected relative error at the output. We use 100k (X, C) as the training dataset and the entire solution space is $2^{15} \times 2^{15}$. For each (X, C), the ground-truth, i.e., expected relative error, is computed by averaging 1k inputs sampled uniformly at random from $[0, 10]^{16}$. The constraint g_r is $\sum_{v \in V} X_v \geq \theta$ with normalization, where $\theta \in \{3, 5, 8\}$. We test the learned $\mathcal{A}_\theta$ on 500 unseen configurations.

Evaluation. Table. 4 shows the averaged relative errors of the assignments by different methods. The problem is far from trivial. Intuitively, assigning AxC units closed to the output, we may expect small error. However, C-Out performs bad. Our proxies AFF and CON obtain comparable best results. The MAE loss values of the two proxies are also similar, as shown in Table 8 in the appendix. The reason is that the circuit is made up of 4 layers in total which leads to at most 4-order moments in the objective function, which is in a medium-level complexity. Training time is also studied for this application, resulting in the same conclusion as application II (See Table 7 in the appendix).

5 Conclusion

This work introduces an unsupervised end-to-end framework to resolve LCO problems based on the relaxation-plus-rounding technique. With our entry-wise concave architecture, our framework guarantees that a low objective value could lead to qualified discrete solutions. Our framework is particularly good at solving PCO problems where the objectives need to be modeled and learned. Real-world applications demonstrate the superiority of our method over RL and gradient-relaxation approaches in both optimization performance and training efficiency. In the future, we aim to further broaden our framework so that binary embeddings of the optimization variables are not needed.

References

[1] N. Karalias and A. Loukas, "Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[2] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[3] G. Naseri and M. A. Koffas, "Application of combinatorial optimization strategies in synthetic biology," *Nature communications*, vol. 11, no. 1, 2020.

[4] Y. Crama, "Combinatorial optimization models for production scheduling in automated manufacturing systems," *European Journal of Operational Research*, vol. 99, no. 1, 1997.

[5] J. J. Hopfield and D. W. Tank, ""neural" computation of decisions in optimization problems," *Biological Cybernetics*, vol. 52, no. 3, 1985.

[6] K. A. Smith, "Neural networks for combinatorial optimization: a review of more than a decade of research," *INFORMS Journal on Computing*, vol. 11, no. 1, 1999.

[7] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[8] D. Selsam, M. Lamm, B. Benedikt, P. Liang, L. de Moura, D. L. Dill *et al.*, "Learning a sat solver from single-bit supervision," in *International Conference on Learning Representations*, 2018.

[9] S. Amizadeh, S. Matusevych, and M. Weimer, "Learning to solve circuit-sat: An unsupervised differentiable approach," in *International Conference on Learning Representations*, 2018.

[10] E. Yolcu and B. Póczos, "Learning local search heuristics for boolean satisfiability," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[11] E. Khalil, P. Le Bodic, L. Song, G. Nemhauser, and B. Dilkina, "Learning to branch in mixed integer programming," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[12] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, "Exact combinatorial optimization with graph convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[13] A. Delarue, R. Anderson, and C. Tjandraatmadja, "Reinforcement learning with combinatorial actions: An application to vehicle routing," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[14] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[15] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[16] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *International Conference on Learning Representations (Workshop)*, 2017.

[17] X. Chen and Y. Tian, "Learning to perform local rewriting for combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[18] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *International Conference on Learning Representations*, 2018.

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

[20] B. Hudson, Q. Li, M. Malencia, and A. Prorok, "Graph neural network guided local search for the traveling salesperson problem," *International Conference on Learning Representations*, 2022.

[21] Y. Ma, J. Li, Z. Cao, W. Song, L. Zhang, Z. Chen, and J. Tang, "Learning to iteratively solve routing problems with dual-aspect collaborative transformer," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[22] Y.-D. Kwon, J. Choo, I. Yoon, M. Park, D. Park, and Y. Gwon, "Matrix encoding networks for neural combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[23] M. Kim, J. Park *et al.*, "Learning collaborative policies to solve np-hard routing problems," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[24] N. Wu, H. Yang, Y. Xie, p. Li, and C. Hao, "High-level synthesis performance prediction using gnns: Benchmarking, modeling, and advancing," in *Proceedings of IEEE/ACM Design Automation Conference (DAC), 2022.*, 2022.

[25] C. Mendis, A. Renda, S. Amarasinghe, and M. Carbin, "Ithemal: Accurate, portable and fast basic block throughput estimation using deep neural networks," in *International Conference on Machine Learning*, 2019.

[26] G. Zhang, H. He, and D. Katabi, "Circuit-gnn: Graph neural networks for distributed circuit design," in *International Conference on Machine Learning*. PMLR, 2019.

[27] S. Vasudevan, W. J. Jiang, D. Bieber, R. Singh, C. R. Ho, C. Sutton *et al.*, "Learning semantic representations to verify hardware designs," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[28] N. Mishra, C. Imes, J. D. Lafferty, and H. Hoffmann, "Caloree: Learning control for predictable latency and low energy," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018.

[29] A. Renda, Y. Chen, C. Mendis, and M. Carbin, "Difftune: Optimizing cpu simulator parameters with learned differentiable surrogates," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020.

[30] N. Wu, Y. Xie, and C. Hao, "Ironman: Gnn-assisted design space exploration in high-level synthesis via reinforcement learning," in *Proceedings of the 2021 on Great Lakes Symposium on VLSI*, 2021.

[31] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi *et al.*, "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, 2021.

[32] K. Chen and F. H. Arnold, "Enzyme engineering for nonaqueous solvents: random mutagenesis to enhance activity of subtilisin e in polar organic media," *Bio/Technology*, vol. 9, no. 11, 1991.

[33] C. Angermueller, D. Dohan, D. Belanger, R. Deshpande, K. Murphy, and L. Colwell, "Model-based reinforcement learning for biological sequence design," in *International Conference on Learning Representations*, 2019.

[34] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik, "Automatic chemical design using a data-driven continuous representation of molecules," *ACS Central Science*, vol. 4, no. 2, 2018.

[35] D. Bertsimas and J. Tsitsiklis, "Simulated annealing," *Statistical Science*, vol. 8, no. 1, 1993.

[36] J. Toenshoff, M. Ritzert, H. Wolf, and M. Grohe, "Run-csp: unsupervised learning of message passing networks for binary constraint satisfaction problems," 2019.

[37] W. Yao, A. S. Bandeira, and S. Villar, "Experimental performance of graph neural networks on random instances of max-cut," in *Wavelets and Sparsity XVIII*, vol. 11138. International Society for Optics and Photonics, 2019.

[38] G. Yehuda, M. Gabel, and A. Schuster, "It's not what machines can learn, it's what we cannot teach," in *International Conference on Machine Learning*. PMLR, 2020.

[39] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, 2015.

[40] R. Gandhi, S. Khuller, S. Parthasarathy, and A. Srinivasan, "Dependent rounding and its applications to approximation algorithms," *Journal of the ACM (JACM)*, vol. 53, no. 3, 2006.

[41] J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità, "Steiner tree approximation via iterative randomized rounding," *Journal of the ACM (JACM)*, vol. 60, no. 1, 2013.

[42] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symposium (ETS)*. IEEE, 2013.

[43] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, 2016.

[44] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "Macaco: Modeling and analysis of circuits for approximate computing," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2011.

[45] D. Ma, R. Thapa, X. Wang, X. Jiao, and C. Hao, "Workload-aware approximate computing configuration," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021.

[46] C. Li, W. Luo, S. S. Sapatnekar, and J. Hu, "Joint precision optimization and high level synthesis for approximate computing," in *Proceedings of the 52nd Annual Design Automation Conference*, 2015.

[47] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[48] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *International Conference on Learning Representations*, 2017.

[49] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," *International Conference on Learning Representations*, 2017.

[50] Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min, "Pomo: Policy optimization with multiple optima for reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[51] R. Wang, Z. Hua, G. Liu, J. Zhang, J. Yan, F. Qi, S. Yang, J. Zhou, and X. Yang, "A bi-level framework for learning to solve combinatorial optimization on graphs," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[52] Y. Nandwani, D. Jindal, P. Singla *et al.*, "Neural learning of one-of-many solutions for combinatorial problems in structured output spaces," in *International Conference on Learning Representations*, 2021.

[53] A. Kumar and S. Levine, "Model inversion networks for model-based optimization," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[54] D. Brookes, H. Park, and J. Listgarten, "Conditioning by adaptive sampling for robust design," in *International Conference on Machine Learning*. PMLR, 2019.

[55] B. Trabucco, A. Kumar, X. Geng, and S. Levine, "Conservative objective models for effective offline model-based optimization," in *International Conference on Machine Learning*. PMLR, 2021.

[56] A. Kumar, A. Yazdanbakhsh, M. Hashemi, K. Swersky, and S. Levine, "Data-driven offline optimization for architecting hardware accelerators," 2022.

[57] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in Neural Information Processing Systems*, vol. 12, 1999.

[58] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2016.

[59] M. Paulus, D. Choi, D. Tarlow, A. Krause, and C. J. Maddison, "Gradient estimation with stochastic softmax tricks," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[60] K. Struminsky, A. Gadetsky, D. Rakitin, D. Karpushkin, and D. P. Vetrov, "Leveraging recursive gumbel-max trick for approximate inference in combinatorial spaces," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[61] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[62] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," *arXiv preprint arXiv:1909.01315*, 2019.

[63] S. Mirjalili, "Genetic algorithm," in *Evolutionary algorithms and neural networks*. Springer, 2019.

[64] D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, no. 2, 1994.

[65] M. V. Pogančić, A. Paulus, V. Musil, G. Martius, and M. Rolinek, "Differentiation of blackbox combinatorial solvers," in *International Conference on Learning Representations*, 2019.

[66] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, 2012.

[67] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[68] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[69] AMD/Xilinx, "Vitis ai development environment," https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html.

[70] ——, "Vivado development tool," https://www.xilinx.com/products/design-tools/vivado.html.

[71] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[72] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[73] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[74] C. Data61, "Stellargraph machine learning library," https://github.com/stellargraph/stellargraph, 2018.

[75] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković, "Principal neighbourhood aggregation for graph nets," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Sec. 5.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Appendix D.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Sec 2 and Sec 3.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix A.1, A.2 and A.3.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See the experiment details in Appendix C.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix C.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] The dataset in application II is adopted from [30].
   (b) Did you mention the license of the assets? [Yes] See Appendix E.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] All the new assets are publicly available.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] All the assets obtained from others are publicly available.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] See Appendix E and our data contains no human information or offensive content.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]