DOGE-Train: Discrete Optimization on GPU with End-to-End Training

Ahmed Abbas[1], **Paul Swoboda**[1,2,3]

[1]Max Planck Institute for Informatics, Saarland Informatics Campus,
[2]University of Mannheim,
[3]Heinrich-Heine University Düsseldorf

Abstract

We present a fast, scalable, data-driven approach for solving relaxations of 0-1 integer linear programs. We use a combination of graph neural networks (GNN) and a Lagrange decomposition based algorithm. We make the latter differentiable for end-to-end training and use GNNs to predict its algorithmic parameters. This allows to retain the algorithm's theoretical properties including dual feasibility and guaranteed non-decrease in the lower bound while improving it via training. We overcome suboptimal fixed points of the basic solver by additional non-parametric GNN update steps maintaining dual feasibility. For training we use an unsupervised loss. We train on smaller problems and test on larger ones showing strong generalization performance with a GNN comprising only around $10k$ parameters. Our solver achieves significantly faster performance and better dual objectives than its non-learned version, achieving close to optimal objective values of LP relaxations of very large structured prediction problems and on selected combinatorial ones. In particular, we achieve better objective values than specialized approximate solvers for specific problem classes while retaining their efficiency. Our solver has better any-time performance over a large time period compared to a commercial solver.

1 Introduction

Integer linear programs (ILP) are a universal tool for solving combinatorial optimization problems. While great progress has been made on improving ILP solvers over the past several decades, there is recent interest in leveraging machine learning to enhance ILP algorithms. Almost all ILP solving subroutines, except ILP relaxation algorithms, have been recently shown to benefit from learning, including variable selection for branch-and bound (Nair et al. 2020) or cutting plane selection (Huang et al. 2022; Turner et al. 2022; Paulus et al. 2022). Moreover, a number of specialized heuristics as well as meta-algorithms using heuristics as subroutines (Sun and Yang 2023; Qiu, Sun, and Yang 2022) have used ML for greatly improving performance for some problem classes. However no general purpose ILP relaxation algorithm has yet benefited from machine learning.

We make a contribution towards general ML-enabled solvers for optimization by learning a problem-agnostic solver for LP-relaxations of ILPs. LP solving is a key step taking most time in traditional ILP pipelines. State of the art LP solvers (Gurobi Optimization, LLC 2021; Cplex, IBM ILOG 2019; FICO 2022; MOSEK ApS 2022; Gamrath et al. 2020) are not amenable to ML since they are non-differentiable, sequential and have very complex implementations. This makes utilization of neural networks and GPUs for solver improvement difficult. For these reasons we build upon the massively parallel FastDOG (Abbas and Swoboda 2022) solver and show that it can be made differentiable. This allows to train our problem agnostic solver for specific problem classes resulting in equal or better performance as compared to efficient hand-designed specialized solvers.

Contributions Our high-level contributions are conceptual and empirical: (i) We show that embedding good inductive biases coming from non-learned solvers (in our case highly parallel GPU-based block coordinate ascent (Abbas and Swoboda 2022) and subgradients) into neural networks leads to greatly improved performance. In particular, we give evidence to the hypothesis that similar to vision (convolutions) and NLP (sequence models) the right inductive biases coming from solver primitives are a promising way to use the potential of ML for optimization. (ii) Our approach is more economical as compared to developing efficient problem specific heuristics, as is customary for large scale problems in structured prediction tasks for ML (Haller et al. 2020; Hutschenreiter et al. 2021). Instead of spending much time and effort in designing and implementing new algorithms, one can train our problem agnostic solver with a few problem instances coming from the problem class of interest and obtain a state of the art GPU-enabled solver for it[1].

In detail, we propose to learn the Lagrange decomposition algorithm (Abbas and Swoboda 2022) for solving LP relaxations of ILP problems and show its benefits. In particular,

- We generalize the dual optimization algorithm of (Abbas and Swoboda 2022) to allow for a larger space of parameter updates.
- We make our dual optimization algorithm efficiently differentiable and embed it as a layer in a neural network. This enables us to predict parameters of the algorithm leading to faster convergence compared to manually designed rules.

[1]Code available at https://github.com/LPMP/BDD

- We train a predictor for arbitrary non-parametric updates that allow to escape suboptimal fixed points encountered by parametric update steps of (Abbas and Swoboda 2022).
- Our predictors for both of the above updates are trained in a fully unsupervised manner. Our loss optimizes for producing large improvements in the dual objective.
- We show the benefits of our learned approach on a wide range of problems. We have chosen structured prediction tasks including graph matching (Kainmueller et al. 2014) and cell tracking (Haller et al. 2020). From theoretical computer science we compare on the QAPLib (Burkard, Karisch, and Rendl 1997) dataset and on randomly generated independent set problems (Prouvost et al. 2020).

2 Related Work

2.1 Learning to Solve Combinatorial Optimization

ML has been used to improve various aspects of solving combinatorial problems. For the standard branch-and-cut ILP solvers the works (Gasse et al. 2019; Gupta et al. 2020; Nair et al. 2020; Scavuzzo et al. 2022) learn variable selection for branching. The approaches (Ding et al. 2020; Nair et al. 2020) learn to fix a subset of integer variables in ILPs to their hopefully optimal values to improve finding high quality primal solutions. The works (Sonnerat et al. 2021; Wu et al. 2021) learn variable selection for the large neighborhood search heuristic for obtaining primal solutions to ILPs. Selecting good cuts through scoring them with neural networks was investigated in (Huang et al. 2022; Turner et al. 2022). While all these approaches result in runtime and solution quality improvements, only a few works tackle the important task of speeding up ILP relaxations by ML. Specifically, the work (Cappart et al. 2019) used graph neural network (GNN) to predict variable orderings of decision diagrams representing combinatorial optimization problems. The goal is to obtain an ordering such that a corresponding dual lower bound is maximal. To our knowledge it is the only work that accelerates ILP relaxation computation with ML. For constraint satisfaction problems (Selsam et al. 2018; Cameron et al. 2020; Tönshoff et al. 2021) train GNN while the latter train in an unsupervised manner. Parameters of belief propagation are learned in (Deng et al. 2022) for speeding-up inference in graphical models, in a similar spirit to our work. Our method however is more generally applicable, allows to escape fixed-points , and is scalable to larger problems due to efficient implementation.

For narrow subclasses of problems primal heuristics have been augmented through learning some of their decisions, e.g. for capacitated vehicle routing (Nazari et al. 2018), graph matching (Wang, Yan, and Yang 2021) and traveling salesman (Xin et al. 2021). For a more complete overview of ML for combinatorial optimization we refer to the detailed surveys (Bengio, Lodi, and Prouvost 2021; Cappart et al. 2023).

2.2 Unrolling Algorithms for Parameter Learning

Algorithms containing differentiable iterative procedures are combined with neural networks for improving performance of such algorithms. Such approaches show more generalization power than pure neural networks based ones as shown in the survey (Monga, Li, and Eldar 2021). The work of (Gregor and LeCun 2010) embedded sparse coding algorithms in a neural network by unrolling. For solving inverse problems (Yang et al. 2020; Chen and Pock 2017) unroll through ADMM and non-linear diffusion resp. Lastly, neural networks were used to predict update directions for training other neural networks (e.g. in (Andrychowicz et al. 2016)).

3 Method

We first recapitulate the Lagrange decomposition approach to binary ILPs from (Lange and Swoboda 2021) and generalize the optimization scheme of (Abbas and Swoboda 2022) for faster convergence. Then we will show how to backpropagate through the optimization scheme allowing to train a graph neural network for predicting its parameters.

3.1 Lagrange Decomposition

Definition 1 (Binary Program (Lange and Swoboda 2021))**.** Let a linear objective $c \in \mathbb{R}^n$ and m variable subsets $\mathcal{I}_j \subset [n]$ of constraints with feasible set $\mathcal{X}_j \subset \{0,1\}^{\mathcal{I}_j}$ for $j \in [m]$ be given. The ensuing binary program is

$$\min_{x \in \{0,1\}^n} \langle c, x \rangle \quad \text{s.t.} \quad x_{\mathcal{I}_j} \in \mathcal{X}_j \quad \forall j \in [m]\,, \tag{BP}$$

where $x_{\mathcal{I}_j}$ is the restriction to variables in $\mathcal{I}_j$.

Any binary ILP $\min_{x \in \{0,1\}^n} \langle c, x \rangle$ s.t. $Ax \leq b$ where $A \in \mathbb{R}^{m \times n}$ can be written as (BP) by associating each constraint $a_j^T x \leq b_j$ for $j \in [m]$ with its own subproblem $\mathcal{X}_j$. In order to obtain a problem formulation amenable for parallel optimization we consider its Lagrange dual which decomposes the full problem (BP) into a series of coupled subproblems.

Definition 2 (Lagrangean dual problem (Lange and Swoboda 2021))**.** Define the set of subproblems that constrain variable i as $\mathcal{J}_i = \{j \in [m] \mid i \in \mathcal{I}_j\}$. Let the energy for subproblem $j \in [m]$ w.r.t. Lagrange variables $\lambda_{\bullet j} = (\lambda_{ij})_{i \in \mathcal{I}_j} \in \mathbb{R}^{\mathcal{I}_j}$ be

$$E^j(\lambda_{\bullet j}) = \min_{x \in \mathcal{X}_j} \langle \lambda_{\bullet j}, x \rangle\,. \tag{1}$$

Then the Lagrangean dual problem is defined as

$$\max_{\lambda} \sum_{j \in [m]} E^j(\lambda_{\bullet j}) \quad \text{s.t.} \quad \sum_{j \in \mathcal{J}_i} \lambda_{ij} = c_i \quad \forall i \in [n]. \tag{D}$$

The problem (D) provides a lower bound to the NP-Hard optimization problem (BP) and is also useful for primal recovery (Abbas and Swoboda 2022). Our goal is to learn a neural network for optimizing the dual (D) efficiently and to reach better objective values.

3.2 Optimization of Lagrangean Dual

The work of Abbas and Swoboda (2022) proposed a parallelization friendly iterative scheme for optimizing (D) with hand-designed parameters. We generalize their scheme in Algorithm 1, exposing a much larger set of parameters allowing more control over the optimization process. Since this large

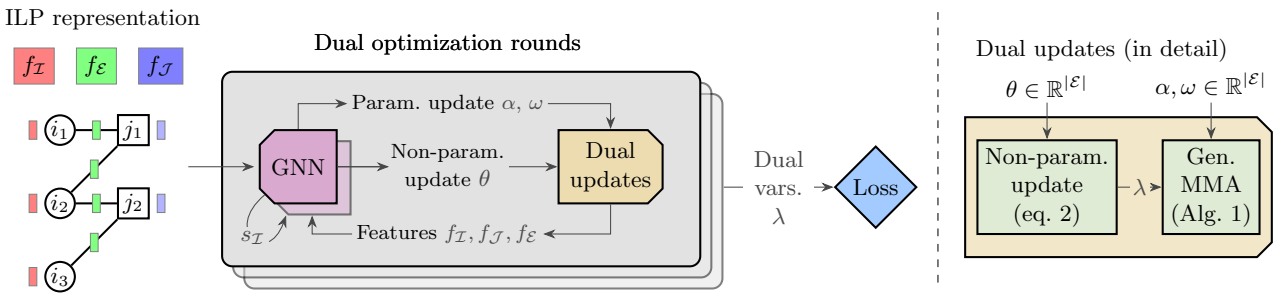


Figure 1: Our method for optimizing the Lagrangean dual (D). The dual problem is encoded on a bipartite graph containing features $f_{\mathcal{I}}$, $f_{\mathcal{J}}$ and $f_{\mathcal{E}}$ for primal variables, subproblems and dual variables resp. A graph neural network (GNN) predicts θ, α, ω for dual updates. In one dual update block (right), current set of Lagrange multipliers λ are first updated by the non-parametric update using θ. Afterwards parametric update is done via Alg. 1 using α, ω. The updated solver features f and LSTM cell states $s_{\mathcal{I}}$ are sent to the GNN in next optimization round. See Sec. 3.6 for further details.

parameter space is difficult to be tuned manually, we will employ a GNN for predicting these parameters.

In detail, Alg. 1 greedily assigns the Lagrange variables in u-many disjoint blocks $B_1, \dots, B_u$ in such a way that each block contains at most one Lagrange variable from each subproblem and all variables within a block are updated in parallel (same as (Abbas and Swoboda 2022)). The dual update scheme relies on computing min-marginal differences i.e., the difference of subproblem objectives when a certain variable is set to 1 minus its objective when the same variable is set to 0 (line 10). These min-marginal differences are averaged out across subproblems via updates to Lagrange variables (line 11). Our algorithm relies on two important set of parameters, damping factors and averaging weights. The damping factor ω_{ij} determine the fraction of min-marginal difference to subtract from variable i in subproblem j. The averaging weights α_{ij} parameterize the fraction of total min-marginal difference $\sum_{ik} M_{ik}$ variable i in subproblem j receives.

Remark. The deferred min-marginal averaging algorithm of (Abbas and Swoboda 2022) is a specialized form of our generalized Algorithm 1 if the parameters were set as $\omega_{ij} = 0.5$ and $\alpha_{ij} = 1/|\mathcal{J}_i|$ for all i, j.

We generalize the min-marginal update step by considering damping factors to be in $(0, 1)$ and averaging weights to be arbitrary convex combinations. We show that this generalized update step still preserves the desirable property of guaranteed non-improvement in the dual objective.

Proposition 1 (Dual Feasibility and Monotonicity of Generalized Min-marginal Averaging)**.** *For any $\alpha_{ij} \geq 0$ with $\sum_{j \in \mathcal{J}_i} \alpha_{ij} = 1$ and $\omega_{ij} \in [0, 1]$ the min-marginal averaging step in line 11 in Algorithm 1 retains dual feasibility and is non-decreasing in the dual lower bound.*

3.3 Backpropagation through Dual Optimization

We show below how to differentiate through Algorithm 1 with respect to its parameters α and ω. This will ultimately allow us to learn these parameters such that faster convergence is achieved. To this end we describe backpropagation for a block update (lines 8-12) of Alg. 1. All other operations can

Algorithm 1: Generalized Min-Marginal Averaging

Input: Lagrange variables $\lambda_{ij}\ \forall i \in [n], j \in \mathcal{J}_i$, damping factors $\omega_{ij} \in (0, 1)\ \forall i \in [n], j \in \mathcal{J}_i$, averaging weights $\alpha_{ij} \in (0, 1)\ \forall i \in [n], j \in \mathcal{J}_i$, max. number of iterations T.

1 Initialize deferred min-marginal diff. $M = \mathbb{0}$
2 **for** T *iterations* **do**
3 **for** *block* $B \in (B_1, \dots B_u)$ **do**
4 $\lambda, M \leftarrow \texttt{BlockUpdate}\,(B, \lambda, M, \alpha, \omega)$
5 **for** *block* $B \in (B_u, \dots B_1)$ **do**
6 $\lambda, M \leftarrow \texttt{BlockUpdate}\,(B, \lambda, M, \alpha, \omega)$
7 **return** λ, M
8 **Procedure** $\texttt{BlockUpdate}\,(B, \lambda^{\texttt{in}}, M^{\texttt{in}}, \alpha, \omega)$
9 **for** $ij \in B$ *in parallel* **do**
10 $M^{\texttt{out}}_{ij} = \omega_{ij}\big[\min_{\substack{x \in \mathcal{X}_j: \\ x_i = 1}} \langle \lambda^{\texttt{in}}_{\bullet j}, x\rangle - \min_{\substack{x \in \mathcal{X}_j: \\ x_i = 0}} \langle \lambda^{\texttt{in}}_{\bullet j}, x\rangle\big]$
11 $\lambda^{\texttt{out}}_{ij} = \lambda^{\texttt{in}}_{ij} - M^{\texttt{out}}_{ij} + \alpha_{ij} \sum_{k \in \mathcal{J}_i} M^{\texttt{in}}_{ik}$
12 **return** $\lambda^{\texttt{out}}$, $M^{\texttt{out}}$

be tackled by automatic differentiation. For a block B in $\{B_1, \dots, B_u\}$ we view the Lagrangean update as a mapping $\mathcal{H} : (\mathbb{R}^{|B|})^4 \to (\mathbb{R}^{|B|})^2$, $(\lambda^{\texttt{in}}, M^{\texttt{in}}, \alpha, \omega) \mapsto (\lambda^{\texttt{out}}, M^{\texttt{out}})$.

Given a loss function $\mathcal{L} : \mathbb{R}^N \to \mathbb{R}$ we denote $\partial \mathcal{L} / \partial x$ by $\dot{x}$. Algorithm 2 shows backpropagation through $\mathcal{H}$ to compute the gradients $\dot{\lambda}^{\texttt{in}}$, $\dot{M}^{\texttt{in}}$, $\dot{\alpha}$ and $\dot{\omega}$.

Proposition 2. *Alg. 2 performs backprop. through $\mathcal{H}$.*

Efficient Implementation Generally, the naive computation of min-marginal differences and its backpropagation are both expensive operations as they require solving two optimization problems for each dual variable. The works of Abbas and Swoboda (2022); Lange and Swoboda (2021) represent each subproblem by a binary decision diagram (BDD) for fast computation of min-marginal differences. Their algorithm results in a computation graph involving only elementary arithmetic operations and taking minima over several variables. Using this computational graph we

Algorithm 2: `BlockUpdate` backpropagation

Input: Forward pass inputs: $B, \lambda^{\text{in}}, M^{\text{in}}, \alpha, \omega$, gradients of forward pass output: $\dot{\lambda}^{\text{out}}, \dot{M}^{\text{out}}$, gradients of parameters $\dot{\alpha}, \dot{\omega}$

for $ij \in B$ *in parallel* **do**
- $\dot{M}^{\text{in}}_{ij} = \sum_{k \in \mathcal{J}_i} \dot{\lambda}^{\text{out}}_{ik} \alpha_{ik}$
- $\dot{M}^{\text{out}}_{ij} = \dot{M}^{\text{out}}_{ij} - \dot{\lambda}^{\text{out}}_{ij}$
- $\dot{\alpha}_{ij} = \dot{\alpha}_{ij} + \dot{\lambda}_{ij} \sum_{k \in \mathcal{J}_i} M^{\text{in}}_{ik}$
- $\dot{\omega}_{ij} = \dot{\omega}_{ij} + \dot{M}^{\text{out}}_{ij} [M^{\text{out}}_{ij} / \omega_{ij}]$
- Compute minimizers for $\beta \in \{0, 1\}$ $s^j(i, \beta) = \operatorname{argmin}_{x \in \mathcal{X}_j : x_i = \beta} \langle \lambda^{\text{in}}_{\bullet j}, x \rangle$
- $\dot{\lambda}^{\text{in}}_{pj} = \dot{\lambda}^{\text{out}}_{pj} + \dot{M}^{\text{out}}_{ij} \omega_{ij} [s^j_p(i, 1) - s^j_p(i, 0)]$, $\forall p \in \mathcal{I}_j$

return $\dot{\lambda}^{\text{in}}, \dot{M}^{\text{in}}, \dot{\alpha}, \dot{\omega}$

can implement the abstract Algorithm 2 efficiently on GPU. For further performance gains we implement custom backpropagation routines in CUDA for more than an order of magnitude decrease in runtime and memory usage as shown in Table 4 of the Appendix.

3.4 Non-parametric Update Steps

Although the min-marginal averaging scheme of Alg. 1 guarantees a non-decreasing lower bound, it can get stuck in suboptimal fixed points, see (Werner 2007) for a discussion for the special case of MAP inference in Markov Random Fields and (Werner, Prusa, and Dlask 2020) for a more general setting. To address this issue we allow arbitrary updates to Lagrange variables through a vector $\hat{\theta} \in \mathbb{R}^{|\lambda|}$ as

$$\lambda_{ij} \leftarrow \lambda_{ij} + \hat{\theta}_{ij} - \frac{1}{|\mathcal{J}_i|} \sum_{k \in \mathcal{J}_i} \hat{\theta}_{ik}, \forall i \in [n], j \in \mathcal{J}_i, \quad (2)$$

where the last term ensures feasibility of updated Lagrange variables w.r.t. the dual problem (D).

3.5 Graph Neural Network

We train a graph neural network (GNN) to predict the parameters $\alpha, \omega \in \mathbb{R}^{|\lambda|}$ of Alg. 1 and also the non-parametric update $\theta \in \mathbb{R}^{|\lambda|}$ for (2). To this end we encode the dual problem (D) on a bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Its nodes correspond to primal variables $\mathcal{I}$ and subproblems $\mathcal{J}$ i.e., $\mathcal{V} = \mathcal{I} \cup \mathcal{J}$ and edges $\mathcal{E} = \{ij \mid i \in \mathcal{I}, j \in \mathcal{J}_i\}$ correspond to Lagrange multipliers. We need to predict values of α_{ij}, ω_{ij} and θ_{ij} for each edge ij in $\mathcal{E}$. We associate features $f = (f_{\mathcal{I}}, f_{\mathcal{J}}, f_{\mathcal{E}})$ with each entity (nodes, edges) of the bipartite graph. Lagrange multipliers λ^{in} and deferred min-marginals M^{in} encode the current state of Alg. 1 as a part of edge features. Additionally, we encode a number of quantities as features which can allow the GNN to make better updates. Specifically, a subgradient of the dual problem (D) is encoded in the edge features $f_{\mathcal{E}}$ and a history of previous dual objectives for each subproblem is encoded in the constraint features $f_{\mathcal{J}}$. This enables our GNN to effectively utilize more information in parameter prediction than conventional hand-designed updates rules can manage. For example (Abbas and Swoboda 2022) can get stuck in suboptimal fixed points due to zero min-marginal differences (Werner, Prusa, and Dlask 2020). Since the GNN additionally has access to subgradient of the dual problem it can escape such fixed points. A complete list of features is provided in the Appendix.

Graph convolution We use the transformer based graph convolution scheme (Shi et al. 2021). We first compute embeddings of all subproblems j in $\mathcal{J}$ by receiving messages from adjacent nodes and edges as

$$\texttt{CONV}_{\mathcal{J}}(f_{\mathcal{I}}, f_{\mathcal{J}}, f_{\mathcal{E}}, \mathcal{E})_j = \mathbf{W_s} f_j + \sum_{i | ij \in \mathcal{E}} a_{ij}(f_j, f_{\mathcal{I}}, f_{\mathcal{E}}; \mathbf{W_a}) \left[\mathbf{W_t} f_i + \mathbf{W_e} f_{ij} \right], \quad (3)$$

where $\mathbf{W_a}, \mathbf{W_s}, \mathbf{W_t}, \mathbf{W_e}$ are trainable parameters and $a_{ij}(f_j, f_{\mathcal{I}}, f_{\mathcal{E}}; \mathbf{W_a})$ is the softmax attention weight between nodes i and j parameterized by $\mathbf{W_a}$. Afterwards we perform message passing in the reverse direction to compute embeddings for variables $\mathcal{I}$. A similar strategy for message passing on bipartite graphs was followed in (Gasse et al. 2019).

Recurrent connections Our default GNN as mentioned above only uses hand-crafted features to maintain a history of previous optimization rounds. To learn a summary of the past updates we optionally allow recurrent connections through an LSTM with forget gate (Gers, Schmidhuber, and Cummins 1999). The LSTM is only applied on primal variable nodes $\mathcal{I}$ and maintains cell states $s_{\mathcal{I}}$ which can be updated and used for parameter prediction in subsequent optimization rounds.

Prediction The learned embeddings from GNN, LSTM outputs and solver features are consumed by a multi-layer perceptron Φ to predict the required variables for each edge ij in $\mathcal{E}$. Afterwards we transform these outputs so that they satisfy Prop. 1. The exact sequence of operations performed by the graph neural network are shown in Alg. 3 where $[u_1, \ldots, u_k]$ denotes concatenation of vectors $u_1, \ldots, u_k$, `LN` denotes layer normalization (Ba, Kiros, and Hinton 2016) and $\texttt{LSTM}_{\mathcal{I}}$ stands for an LSTM cell operating on primal variables $\mathcal{I}$.

Loss Given the Lagrange variables λ we directly use the dual objective (D) as an unsupervised loss to train the GNN. Thus, we maximize the loss L defined as

$$\mathcal{L}(\lambda) = \sum_{j \in [m]} E^j(\lambda_{\bullet j}). \quad (4)$$

For a mini-batch of instances during training we take the mean of corresponding per-instance losses. For backpropagation, gradient of the loss $\mathcal{L}$ w.r.t. Lagrange variables of a subproblem j is computed by finding a minimizing assignment for that subproblem, written as $\left(\frac{\partial \mathcal{L}}{\partial \lambda}\right)_{\bullet j} = \operatorname{argmin}_{x \in \mathcal{X}_j} \langle \lambda_{\bullet j}, x \rangle \in \{0, 1\}^{\mathcal{I}_j}$. This gradient is then sent as input for backpropagation. For computing the minimizing assignment efficiently we use binary decision diagram representation of each subproblem as in (Abbas and Swoboda 2022; Lange and Swoboda 2021).

Algorithm 3: Parameter prediction by GNN

Input: Primal variable features $f_{\mathcal{I}}$ and cell states $s_{\mathcal{I}}$, Subproblem features $f_{\mathcal{J}}$, Dual variable (edge) features $f_{\mathcal{E}}$, Set of edges $\mathcal{E}$.

// Compute subproblems embeddings

$h_{\mathcal{J}} = \texttt{ReLU}\left(\texttt{LN}\left(\texttt{CONV}_{\mathcal{J}}\left(f_{\mathcal{I}}, f_{\mathcal{J}}, f_{\mathcal{E}}, \mathcal{E}\right)\right)\right)$

// Compute primal var embeddings

$h_{\mathcal{I}} = \texttt{ReLU}\left(\texttt{LN}\left(\texttt{CONV}_{\mathcal{I}}\left(f_{\mathcal{I}}, [f_{\mathcal{J}}, h_{\mathcal{J}}], f_{\mathcal{E}}, \mathcal{E}\right)\right)\right)$

// Compute output and cell state

$z_{\mathcal{I}}, s_{\mathcal{I}} = \texttt{LSTM}_{\mathcal{I}}(h_{\mathcal{I}}, s_{\mathcal{I}})$

// Prediction per edge

$(\hat{\alpha}, \hat{\omega}, \hat{\theta}) = \Phi\left([f_{\mathcal{I}}, h_{\mathcal{I}}, z_{\mathcal{I}}], [f_{\mathcal{J}}, h_{\mathcal{J}}], f_{\mathcal{E}}, \mathcal{E}\right)$

// Ensure non-decreasing obj., Prop 1:

$\alpha_{i\bullet} = \texttt{Softmax}(\hat{\alpha}_{i\bullet}), \forall i \in \mathcal{I}$, $\omega = \texttt{Sigmoid}(\hat{\omega})$

// Maintain dual feasibility

$\theta_{i\bullet} = \hat{\theta}_{i\bullet} - \frac{1}{|\mathcal{J}_i|}\sum_{k \in \mathcal{J}_i} \hat{\theta}_{ik}, \forall i \in \mathcal{I}$

return $\alpha, \omega, \theta, s_{\mathcal{I}}$

3.6 Overall Pipeline

We train our pipeline (Fig. 1) which contains multiple dual optimization rounds in a fashion similar to that of recurrent neural networks. One round of our dual optimization consists of message passing by GNN, a non-parametric update step and T iterations of generalized min-marginal averaging. For computational efficiency we run our pipeline for at most R dual optimization rounds during training. On each mini-batch we randomly sample a number of optimization rounds r in $[R]$, run $r-1$ rounds without tracking gradients and backpropagate through the last round by computing the loss (4). For the pipeline with recurrent connections we backpropagate through last 3 rounds and apply the loss after each of these rounds. Since the task of dual optimization is relatively easier in early rounds as compared to later ones we use two neural networks. The early stage network is trained if the randomly sampled r is in $[0, R/2]$ and the late stage network is chosen otherwise. During testing we switch to the later stage network when the relative improvement in the dual objective by the early stage network becomes less than 10^{-6}. For computational efficiency during testing we query the GNN for parameter updates only after $T \gg 1$ iterations of Alg. 1.

4 Experiments

4.1 Evaluation

As main evaluation metric we report convergence plots of the relative dual gap $g(t) \in [0,1]$ at time t by $g(t) = \min\left(\frac{d^* - d(t)}{d^* - d_{init}}, 1.0\right)$ where $d(t)$ is the dual objective at time t, d^* is the optimal (or best known) objective value of the Lagrange relaxation (D) and d_{init} is the objective before optimization as computed by (Abbas and Swoboda 2022). Additionally we also report per dataset averages of best objective value (E), time taken (t) to obtain best objective and relative dual gap integral $g_I = \int g(t)dt$ (Berthold 2013). The latter metric g_I allows to measure quality of the solution in conjunction with time take to obtain this solution.

4.2 Datasets

We evaluate our approach on a variety of datasets from different domains. For each dataset we train our pipeline on smaller instances and test on larger ones.

Cell tracking (CT): Instances of developing flywing tissue from cell tracking challenge (Ulman et al. 2017) processed by (Haller et al. 2020) and obtained from (Swoboda et al. 2022). We use the largest and hardest 3 instances, train on the 2 smaller instances and test on the largest one.

Graph matching (GM): Instances of graph matching for matching nuclei in 3D microscopic images (Long et al. 2009) processed by (Kainmueller et al. 2014) and made publicly available through (Swoboda et al. 2022) as ILPs. We train on 10 instances and test on the remaining 20.

Independent set (IS): Random instances of independent set problem generated using (Prouvost et al. 2020). For training we use 240 instances with $10k$ nodes each and test on 60 instances with $50k$ nodes.

QAPLib: The benchmark dataset for quadratic assignment problems used in the combinatorial optimization community (Burkard, Karisch, and Rendl 1997). The benchmark contains problems arising from a variety of domains e.g., keyboard design, hospital layout, circuit synthesis, facility location etc. We train on 61 instances having up to $0.6M$ Lagrange variables and test on 35 instances having up to $10.6M$ Lagrange variables. Conversion to ILP is done via (2.7)-(2.16) of (Loiola et al. 2007)

For each dataset the size of problems (D) are reported in Table 1. Due to varying instance sizes we use a separate set of hyperparameters for each dataset given in Table 5 of the Appendix[2]. For the *CT* dataset we only predict $\theta \in \mathbb{R}^{|\lambda|}$ for non-parametric update steps (2) and fix the parameters α, ω in Alg. 1 to their default values from (Abbas and Swoboda 2022). Learning these parameters gave slightly worse training loss at convergence possibly due to small training set size.

	# vars. (M)		# cons. (M)		# edges (M)	
	train	test	train	test	train	test
CT	3.1	10.1	0.6	2.2	8.5	27.5
GM	1.5	0.1	1.5	0.1	3.3	3.3
IS	0.01	0.05	0.04	0.4	0.1	1.1
QAPLib	0.1	2.5	0.02	0.2	0.6	10.6

Table 1: Size of datasets where the values are averaged within each train/test split (M: million). Number of edges in the GNN equal the number of Lagrange multipliers λ.

4.3 Algorithms

`Gurobi`: The dual simplex algorithm from the commercial solver (Gurobi Optimization, LLC 2021).

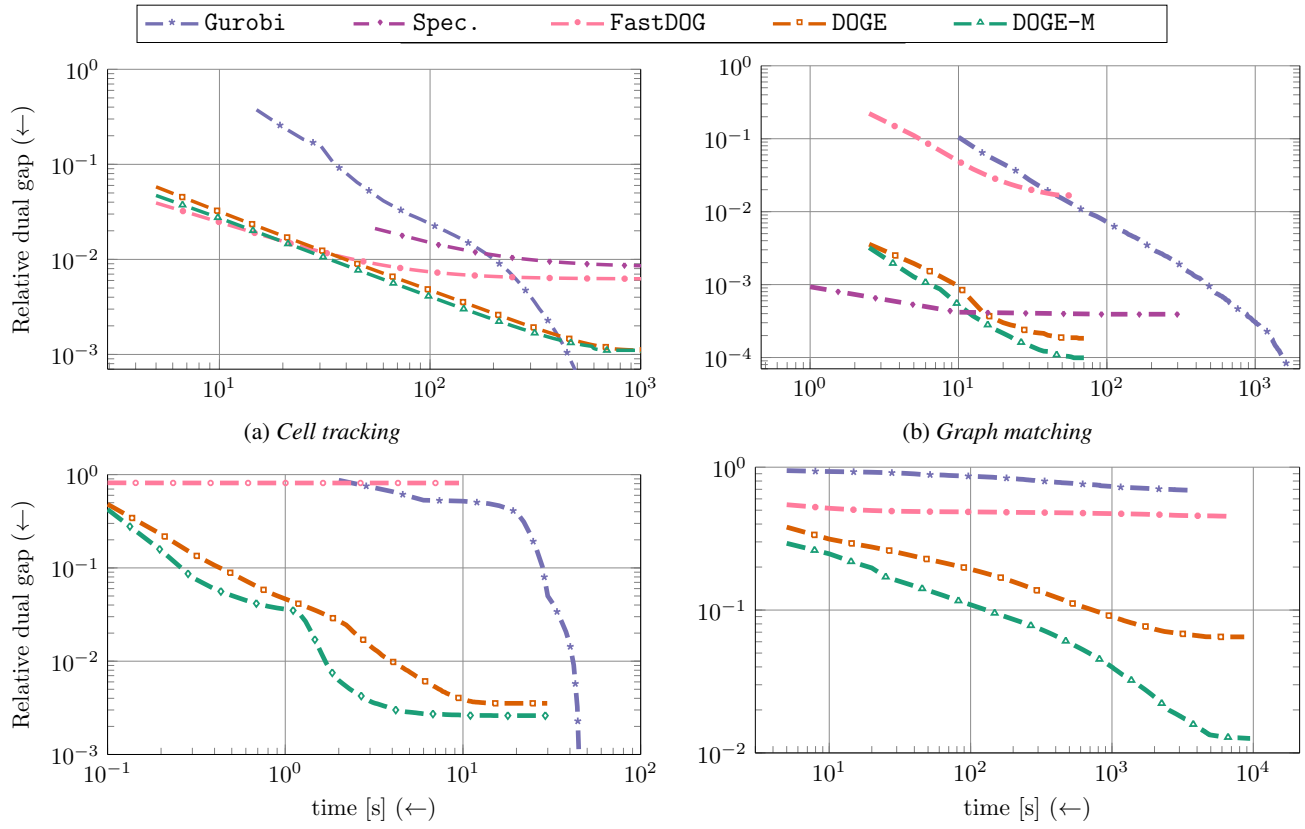
[2]In our full paper: https://arxiv.org/abs/2205.11638

	Cell tracking			*Graph matching*			*Independent set*			*QAPLib*		
	g_I	$E(\times 10^8)$	$t[s]$	g_I	$E(\times 10^4)$	$t[s]$	g_I	$E(\times 10^4)$	$t[s]$	g_I	$E(\times 10^6)$	$t[s]$
`Gurobi`	18	**−3.852**	809	9	**−4.8433**	278	14	**−2.4457**	52	3472	0.9	2618
`Spec.`	-	−3.866	1673	-	−4.8443	100	-	-	-	-	-	-
`FastDOG`	7	−3.863	1005	21	−4.8912	61	42	−2.4913	9	276	5.7	1680
`DOGE`	2.4	−3.854	1015	0.3	−4.8439	17	0.3	−2.4460	8	320	<u>12.1</u>	720
`DOGE-M`	**2.1**	<u>−3.854</u>	730	**0.2**	<u>−4.8436</u>	21	**0.2**	<u>−2.4459</u>	5	**131**	**14.5**	861

Table 2: Results on tests instances where the values are averaged within a dataset. Numbers in bold highlight the best performance and underlines indicate the second best objective.

	w/o learn. (`FastDOG`)	w/o GNN	same network	only non-param.	only param.	w/o α	w/o ω	`DOGE`	`DOGE-M`
g_I (↓)	21	0.42	0.95	2.3	0.7	0.36	0.35	0.33	**0.19**
E (↑)	−48912	−48440	−48444	−48476	−48444	−48439	−48439	−48439	**−48436**
$t[s]$ (↓)	61	29	24	51	74	30	30	17	21

Table 3: Ablation study results on the *Graph matching* dataset. w/o GNN: Use only the two predictors Φ without GNN for early and late stage optimization; same network: use one network (GNN, Φ) for both early and late stage; only non-param., param.: predict only the non-parametric update (2) or the parametric update (Alg. 1); w/o α, ω: does not predict α or ω resp.



(a) *Cell tracking*

(b) *Graph matching*

(c) *Independent set*

(d) *QAPLib*

Figure 2: Convergence plots for $g(t)$ the relative dual gap to the optimum (or maximum suboptimal objective among all methods) of the relaxation (D). X-axis indicates wall clock time and both axes are logarithmic. The value of $g(t)$ is averaged over all test instances in each dataset.

`Spec.`: For graph matching and cell tracking datasets we also report results of state-of-the-art dataset specific solvers. For cell tracking the solver of (Haller et al. 2020) and for graph matching the best performing solver (fm-bca) from recent benchmark (Haller et al. 2022).

`FastDOG`: The non-learned baseline (Abbas and Swoboda 2022) with their hand designed parameters $\omega_{ij} = 0.5$ and $\alpha_{ij} = 1/|\mathcal{J}_i|$ as a specialization of Alg. 1.

`DOGE`: Our approach where we learn to predict parametric and non-parametric updates by using two graph neural networks for early and late-stage optimization. Size of the learned embeddings h computed by the GNN in Alg. 3 is set to 16 for nodes and 8 for edges. For computing attention weights in (3) we use only one attention head for efficiency. The predictor Φ in Alg. 3 contains 4 linear layers with the ReLU activation. We train the networks using the Adam optimizer (Kingma and Ba 2014). To prevent gradient overflow we use gradient clipping on model parameters by an l^2 norm of 50. The number of trainable parameters is $8k$.

`DOGE-M`: Variant of our method where we additionally use recurrent connections using LSTM. The cell state vector s_i for each primal variable node $i \in \mathcal{I}$ has a size of 16. The number of trainable parameters is $12k$.

Note the test instances require millions of solver parameters to be predicted (ref. Table 1) while our largest GNN has $12k$ parameters.

For training we use PyTorch and implement Alg. 1,2 in CUDA. CPU solvers use AMD EPYC 7702 CPU with 16 threads. GPU solvers use one of RTX 8000 (48GB) or A100 (80GB) GPU depending on problem size.

4.4 Results

For each dataset we evaluate our methods on corresponding testing split. Convergence plots of relative dual gaps change (averaged over all test instances) are given in Figure 2. Other evaluation metrics are reported in Table 2. For further details we refer to the Appendix.

Discussion As compared to the non-learned baseline `FastDOG` we reach an order of magnitude more accurate relaxation solutions, almost closing the gap to optimum as computed by `Gurobi`. Even though given unlimited time `Gurobi` attains the optimum, we reach reasonably close values that are considered correct for practical purposes. For example the graph matching benchmark (Haller et al. 2022) considers a relative gap less than 10^{-3} as optimal (we achieve 10^{-4}). Moreover our learned solvers reach much better objective values as compared to specialized solvers. Using LSTM in `DOGE-M` further improves the performance especially on the most difficult *QAPLib* dataset. On *QAPLib* `Gurobi` does not converge on instances with more than 40 nodes within the time limit of one hour. We show convergence plots for smaller instances in the Appendix. The difference to `Gurobi` is most pronounced w.r.t. anytime performance measured by g_I, as our solver reaches good solutions relatively early.

Ablation study We evaluate the importance of various components in our approach. Starting from (Abbas and Swoboda 2022) as a baseline we first predict all parameters α, ω, θ through the two multi-layer perceptrons Φ for early and late stage optimization without using GNN. Next, we report results of using one network (instead of two) which is trained and tested for both early and later rounds of dual optimization. Lastly, we aim to seek the importance of learning parameters of Alg. 2 and the non-parametric update (2). To this end, we learn to predict only the non-parametric update and apply the loss directly on updated λ without requiring backpropagation through Alg. 1. We also try learning a subset of parameters i.e., not predicting averaging weights α or damping factors ω. Lastly, we report results of `DOGE-M` which uses recurrent connections. The results for *graph matching* dataset are in Table 3. Results on other datasets are provided in the Appendix.

Firstly, from our ablation study we observe that learning even one of the two types of updates i.e., non-parametric or parametric already gives better results than the non-learned solver `FastDOG`. This is because non-parametric update can help in escaping fixed-points when they occur and the parametric update can help Alg. 1 in avoiding such fixed-points. Combining both of these strategies further improves the results. Secondly, we observe that performing message passing with GNN gives improvement over only using the MLP Φ. Thirdly, we find using separate networks for early and late stage optimization gives better performance than using the same network for all stages. Lastly, using recurrent connections through an LSTM gives the best performance.

Limitations Easy problem classes, including small *cell tracking* (Haller et al. 2020) and easy Markov Random Field (MRF) inference (Kappes et al. 2013) do not benefit from learning, since `FastDOG` already solves the problem in few iterations. Some problem classes have sequential bottlenecks due to long subproblems, including MRFs for protein folding (Jaimovich et al. 2006) and shape matching (Windheuser et al. 2011a,b), which makes training difficult due to slow dual optimization. Although our method requires training for each problem class, the cost of training is manageable. Nonetheless devising a generalizable approach is an interesting research direction requiring at least: a large and diverse training set, powerful neural network and multi-GPU support.

5 Conclusion

We have proposed an self-supervised learning approach for solving relaxations to combinatorial optimization problems by backpropagating through and learning parameters for the dual LP solver (Abbas and Swoboda 2022). We demonstrated its potential in obtaining close to optimal solutions much faster than with traditional methods. Although our solvers require training as compared to conventional solvers, this overhead is negligible as compared to the human effort required for developing efficient specialized solvers (which are also often outperformed by our approach).

Our work generalizes efficient approximate solver development: Instead of developing a specialized solver we propose to use a generically applicable one and train it to obtain fast and accurate optimization algorithm. Going one step further and training a universal model that generalizes across different problem classes remains a challenge for future work.

Acknowledgements

We thank all anonymous reviewers for their feedback especially reviewer 4 for the detailed discussion and pointing out related work. We also thank Paul Roetzer for suggestions regarding writing.

References

Abbas, A.; and Swoboda, P. 2022. FastDOG: Fast Discrete Optimization on GPU. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Andrychowicz, M.; Denil, M.; Gomez, S.; Hoffman, M. W.; Pfau, D.; Schaul, T.; Shillingford, B.; and De Freitas, N. 2016. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29.

Ba, J. L.; Kiros, J. R.; and Hinton, G. E. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bengio, Y.; Lodi, A.; and Prouvost, A. 2021. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2): 405–421.

Berthold, T. 2013. Measuring the Impact of Primal Heuristics. *Oper. Res. Lett.*, 41(6): 611–614.

Burkard, R. E.; Karisch, S. E.; and Rendl, F. 1997. QAPLIB–a quadratic assignment problem library. *Journal of Global optimization*, 10(4): 391–403.

Cameron, C.; Chen, R.; Hartford, J.; and Leyton-Brown, K. 2020. Predicting Propositional Satisfiability via End-to-End Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04): 3324–3331.

Cappart, Q.; Chételat, D.; Khalil, E. B.; Lodi, A.; Morris, C.; and Velickovic, P. 2023. Combinatorial optimization and reasoning with graph neural networks. *J. Mach. Learn. Res.*, 24: 130–1.

Cappart, Q.; Goutierre, E.; Bergman, D.; and Rousseau, L.-M. 2019. Improving optimization bounds using machine learning: Decision diagrams meet deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 1443–1451.

Chen, Y.; and Pock, T. 2017. Trainable Nonlinear Reaction Diffusion: A Flexible Framework for Fast and Effective Image Restoration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6): 1256–1272.

Cplex, IBM ILOG. 2019. CPLEX Optimization Studio 12.10.

Deng, Y.; Kong, S.; Liu, C.; and An, B. 2022. Deep Attentive Belief Propagation: Integrating Reasoning and Learning for Solving Constraint Optimization Problems. *Advances in Neural Information Processing Systems*, 35: 25436–25449.

Ding, J.-Y.; Zhang, C.; Shen, L.; Li, S.; Wang, B.; Xu, Y.; and Song, L. 2020. Accelerating primal solution findings for mixed integer programs based on solution prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 1452–1459.

FICO. 2022. FICO Xpress Optimization Suite.

Gamrath, G.; Anderson, D.; Bestuzheva, K.; Chen, W.-K.; Eifler, L.; Gasse, M.; Gemander, P.; Gleixner, A.; Gottwald, L.; Halbig, K.; Hendel, G.; Hojny, C.; Koch, T.; Bodic, P. L.; Maher, S. J.; Matter, F.; Miltenberger, M.; Mühmer, E.; Müller, B.; Pfetsch, M.; Schlösser, F.; Serrano, F.; Shinano, Y.; Tawfik, C.; Vigerske, S.; Wegscheider, F.; Weninger, D.; and Witzig, J. 2020. The SCIP Optimization Suite 7.0. Technical Report 20-10, ZIB, Takustr. 7, 14195 Berlin.

Gasse, M.; Chételat, D.; Ferroni, N.; Charlin, L.; and Lodi, A. 2019. Exact combinatorial optimization with graph convolutional neural networks. *arXiv preprint arXiv:1906.01629*.

Gers, F.; Schmidhuber, J.; and Cummins, F. 1999. Learning to forget: continual prediction with LSTM. In *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, volume 2, 850–855 vol.2.

Gregor, K.; and LeCun, Y. 2010. Learning Fast Approximations of Sparse Coding. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, 399–406. Madison, WI, USA: Omnipress. ISBN 9781605589077.

Gupta, P.; Gasse, M.; Khalil, E.; Mudigonda, P.; Lodi, A.; and Bengio, Y. 2020. Hybrid models for learning to branch. *Advances in neural information processing systems*, 33: 18087–18097.

Gurobi Optimization, LLC. 2021. Gurobi Optimizer Reference Manual.

Haller, S.; Feineis, L.; Hutschenreiter, L.; Bernard, F.; Rother, C.; Kainmüller, D.; Swoboda, P.; and Savchynskyy, B. 2022. A Comparative Study of Graph Matching Algorithms in Computer Vision. In *Proceedings of the European Conference on Computer Vision*.

Haller, S.; Prakash, M.; Hutschenreiter, L.; Pietzsch, T.; Rother, C.; Jug, F.; Swoboda, P.; and Savchynskyy, B. 2020. A Primal-Dual Solver for Large-Scale Tracking-by-Assignment. In *AISTATS*.

Huang, Z.; Wang, K.; Liu, F.; Zhen, H.-L.; Zhang, W.; Yuan, M.; Hao, J.; Yu, Y.; and Wang, J. 2022. Learning to select cuts for efficient mixed-integer programming. *Pattern Recognition*, 123: 108353.

Hutschenreiter, L.; Haller, S.; Feineis, L.; Rother, C.; Kainmüller, D.; and Savchynskyy, B. 2021. Fusion moves for graph matching. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 6270–6279.

Jaimovich, A.; Elidan, G.; Margalit, H.; and Friedman, N. 2006. Towards an integrated protein–protein interaction network: A relational markov network approach. *Journal of Computational Biology*, 13(2): 145–164.

Kainmueller, D.; Jug, F.; Rother, C.; and Myers, G. 2014. Active graph matching for automatic joint segmentation and annotation of C. elegans. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 81–88. Springer.

Kappes, J.; Andres, B.; Hamprecht, F.; Schnorr, C.; Nowozin, S.; Batra, D.; Kim, S.; Kausler, B.; Lellmann, J.; Komodakis, N.; et al. 2013. A comparative study of modern inference techniques for discrete energy minimization problems. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1328–1335.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lange, J.-H.; and Swoboda, P. 2021. Efficient Message Passing for 0–1 ILPs with Binary Decision Diagrams. In *International Conference on Machine Learning*, 6000–6010. PMLR.

Loiola, E. M.; De Abreu, N. M. M.; Boaventura-Netto, P. O.; Hahn, P.; and Querido, T. 2007. A survey for the quadratic assignment problem. *European journal of operational research*, 176(2): 657–690.

Long, F.; Peng, H.; Liu, X.; Kim, S. K.; and Myers, E. 2009. A 3D digital atlas of C. elegans and its application to single-cell analyses. *Nature methods*, 6(9): 667–672.

Monga, V.; Li, Y.; and Eldar, Y. C. 2021. Algorithm Unrolling: Interpretable, Efficient Deep Learning for Signal and Image Processing. *IEEE Signal Processing Magazine*, 38(2): 18–44.

MOSEK ApS. 2022. *9.0.105*.

Nair, V.; Bartunov, S.; Gimeno, F.; von Glehn, I.; Lichocki, P.; Lobov, I.; O'Donoghue, B.; Sonnerat, N.; Tjandraatmadja, C.; Wang, P.; et al. 2020. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*.

Nazari, M.; Oroojlooy, A.; Snyder, L.; and Takác, M. 2018. Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems*, 31.

Paulus, M. B.; Zarpellon, G.; Krause, A.; Charlin, L.; and Maddison, C. 2022. Learning to cut by looking ahead: Cutting plane selection via imitation learning. In *International conference on machine learning*, 17584–17600. PMLR.

Prouvost, A.; Dumouchelle, J.; Scavuzzo, L.; Gasse, M.; Chételat, D.; and Lodi, A. 2020. Ecole: A Gym-like Library for Machine Learning in Combinatorial Optimization Solvers. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*.

Qiu, R.; Sun, Z.; and Yang, Y. 2022. DIMES: A Differentiable Meta Solver for Combinatorial Optimization Problems. In Oh, A. H.; Agarwal, A.; Belgrave, D.; and Cho, K., eds., *Advances in Neural Information Processing Systems*.

Scavuzzo, L.; Chen, F.; Chételat, D.; Gasse, M.; Lodi, A.; Yorke-Smith, N.; and Aardal, K. 2022. Learning to branch with tree mdps. *Advances in Neural Information Processing Systems*, 35: 18514–18526.

Selsam, D.; Lamm, M.; Bünz, B.; Liang, P.; de Moura, L.; and Dill, D. L. 2018. Learning a SAT solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*.

Shi, Y.; Huang, Z.; Feng, S.; Zhong, H.; Wang, W.; and Sun, Y. 2021. Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification. In Zhou, Z.-H., ed., *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, 1548–1554. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Sonnerat, N.; Wang, P.; Ktena, I.; Bartunov, S.; and Nair, V. 2021. Learning a Large Neighborhood Search Algorithm for Mixed Integer Programs. *arXiv preprint arXiv:2107.10201*.

Sun, Z.; and Yang, Y. 2023. DIFUSCO: Graph-based Diffusion Solvers for Combinatorial Optimization. *arXiv preprint arXiv:2302.08224*.

Swoboda, P.; Hornakova, A.; Roetzer, P.; and Abbas, A. 2022. Structured Prediction Problem Archive. *arXiv preprint arXiv:2202.03574*.

Turner, M.; Koch, T.; Serrano, F.; and Winkler, M. 2022. Adaptive Cut Selection in Mixed-Integer Linear Programming. *arXiv preprint arXiv:2202.10962*.

Tönshoff, J.; Ritzert, M.; Wolf, H.; and Grohe, M. 2021. Graph Neural Networks for Maximum Constraint Satisfaction. *Frontiers in Artificial Intelligence*, 3.

Ulman, V.; Maška, M.; Magnusson, K. E.; Ronneberger, O.; Haubold, C.; Harder, N.; Matula, P.; Matula, P.; Svoboda, D.; Radojevic, M.; et al. 2017. An objective comparison of cell-tracking algorithms. *Nature methods*, 14(12): 1141–1152.

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

Werner, T. 2007. A linear programming approach to max-sum problem: A review. *IEEE transactions on pattern analysis and machine intelligence*, 29(7): 1165–1179.

Werner, T.; Prusa, D.; and Dlask, T. 2020. Relative Interior Rule in Block-Coordinate Descent. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Windheuser, T.; Schlickewei, U.; Schmidt, F. R.; and Cremers, D. 2011a. Geometrically consistent elastic matching of 3d shapes: A linear programming solution. In *2011 International Conference on Computer Vision*, 2134–2141. IEEE.

Windheuser, T.; Schlickwei, U.; Schimdt, F. R.; and Cremers, D. 2011b. Large-scale integer linear programming for orientation preserving 3d shape matching. In *Computer Graphics Forum*, volume 30, 1471–1480. Wiley Online Library.

Wu, Y.; Song, W.; Cao, Z.; and Zhang, J. 2021. Learning Large Neighborhood Search Policy for Integer Programming. *Advances in Neural Information Processing Systems*, 34.

Xin, L.; Song, W.; Cao, Z.; and Zhang, J. 2021. NeuroLKH: Combining Deep Learning Model with Lin-Kernighan-Helsgaun Heuristic for Solving the Traveling Salesman Problem. *Advances in Neural Information Processing Systems*, 34.

Yang, Y.; Sun, J.; Li, H.; and Xu, Z. 2020. ADMM-CSNet: A Deep Learning Approach for Image Compressive Sensing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(3): 521–538.