

UNROLLED GRAPH NEURAL NETWORKS FOR CONSTRAINED OPTIMIZATION

Samar Hadou Alejandro Ribeiro

Department of Electrical and Systems Engineering, University of Pennsylvania, PA, USA

ABSTRACT

In this paper, we unroll the dynamics of the dual ascent (DA) algorithm in two coupled graph neural networks (GNNs) to solve constrained optimization problems. The two networks interact with each other at the layer level to find a saddle point of the Lagrangian. The primal GNN finds a stationary point for a given dual multiplier, while the dual network iteratively refines its estimates to reach an optimal solution. We force the primal and dual networks to mirror the dynamics of the DA algorithm by imposing descent and ascent constraints. We propose a joint training scheme that alternates between updating the primal and dual networks. Our numerical experiments demonstrate that our approach yields near-optimal near-feasible solutions and generalizes well to out-of-distribution (OOD) problems.

Index Terms— Unrolling, constrained optimization, GNNs, dual ascent, constrained learning

1. INTRODUCTION

Algorithm unrolling has emerged as a learning-to-optimize paradigm that embeds iterative solvers into neural network architectures, yielding learnable optimizers endowed with domain-specific knowledge [1, 2]. Despite the extensive literature on unrolling, most work focuses on unconstrained optimization problems, e.g., [2–11]. For constrained settings, a common approach is to incorporate constraints into the training loss as penalty terms [12–14]. Another line of work considers unrolling primal-dual algorithms [15–18], but these efforts typically restrict learning to a few scalar hyperparameters, such as the step size, rather than learning the full algorithmic dynamics. As a result, such approaches remain narrowly tailored to the designated problem classes.

In this paper, we unroll the dynamics of dual ascent (DA) into two coupled graph neural networks (GNNs). The models aim to collaboratively find the saddle point of the Lagrangian function associated with a constrained problem. The primal GNN predicts a layerwise trajectory towards a stationary point of the Lagrangian for a given multiplier, while the dual GNN uses the primal predictions to ascend the dual function toward its optimum. We cast the training problem as a nested optimization problem, where the inner level trains the primal network over a joint distribution of constrained problems and Lagrangian multipliers, while the outer level trains the dual network. Crucially, the multiplier distribution required by the primal network training is not known *a priori*, as it is induced by the multiplier trajectories generated by the forward pass of the dual network. Our nested formulation allows for alternating training steps to overcome this challenge. The dual network first generates multiplier trajectories to train the primal network, whose solutions are then used in the dual network updates. This procedure optimizes each network against the most recent outputs of the other until convergence.

Departing from traditional unrolling literature, which wires a specific iterative algorithm into the architecture, we instead use standard GNNs to model the primal and dual iterates. GNNs are well

suited to constrained problems, as variable-constraint relations can be naturally encoded in a graph adjacency matrix [19–21]. However, this design choice can further exacerbate the lack of monotonic descent often observed in hand-coded unrolled networks [7]. To address this limitation, we impose monotonic descent and ascent constraints on the outputs of the primal and dual layers, respectively, during training. These descent constraints improve the final-layer performance and enhance out-of-distribution (OOD) generalization, as illustrated in our numerical experiments.

In summary, the main contributions of this work are as follows.

- We design a pair of GNNs that interact at the layer level to solve the dual problem (Section 3).
- We formulate the training problem as a nested optimization problem with an alternating training scheme (Section 4).
- We impose descent and ascent constraints on the unrolled layers of the primal and dual networks, respectively, during training (Section 4).
- We empirically assess our approach on mixed-integer quadratic programs (MIQPs) with linear constraints (Section 5)

2. CONSTRAINED OPTIMIZATION

Consider a constrained problem that poses the task of minimizing a scalar objective function $f_0 : \mathbb{R}^n \rightarrow \mathbb{R}$ subject to m constraints, formulated as

$$P^*(\mathbf{z}) = \min_{\mathbf{x} \in \mathbb{R}^n} f_0(\mathbf{x}; \mathbf{z}) \quad \text{s.t.} \quad \mathbf{f}(\mathbf{x}; \mathbf{z}) \leq \mathbf{0}, \quad (1)$$

where $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is a vector-valued function representing the problem constraints, and \mathbf{z} represents a *problem instance*. Moving to the dual domain, the Lagrangian function, $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}_+^m \rightarrow \mathbb{R}$, associated with (1) is

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}; \mathbf{z}) = f_0(\mathbf{x}; \mathbf{z}) + \boldsymbol{\lambda}^\top \mathbf{f}(\mathbf{x}; \mathbf{z}), \quad (2)$$

where $\boldsymbol{\lambda}$ contains the dual multipliers. The dual problem is defined as

$$D^*(\mathbf{z}) = \max_{\boldsymbol{\lambda} \in \mathbb{R}_+^m} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}; \mathbf{z}), \quad (3)$$

and the duality theory affirms that $D^*(\mathbf{z}) \leq P^*(\mathbf{z})$ [22]. The equality holds under strong duality, which applies to convex problems. Under this assumption, the Lagrangian has a saddle point $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$, with \mathbf{x}^* and $\boldsymbol{\lambda}^*$ optimal for (1) and (3), respectively. The dual ascent (DA) algorithm retrieves the dual optimum $\boldsymbol{\lambda}^*$ through the iterations:

$$\begin{aligned} \mathbf{x}_l^*(\boldsymbol{\lambda}_l) &\in \underset{\mathbf{x}}{\operatorname{argmin}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}_l; \mathbf{z}), \\ \boldsymbol{\lambda}_{l+1} &= \left[\boldsymbol{\lambda}_l + \eta \mathbf{f}(\mathbf{x}_l^*, \mathbf{z}) \right]_+, \end{aligned} \quad (4)$$

where η is a step size, and $[\cdot]_+$ denotes a projection onto \mathbb{R}_+^m . In (4), a Lagrangian stationary point is attained for the recent multiplier $\boldsymbol{\lambda}_l$, before a (projected) gradient ascent step is taken in the dual domain. The primal optimum \mathbf{x}^* is then recovered from $\mathbf{x}^* \in \mathbf{x}^*(\boldsymbol{\lambda}^*)$.

3. UNROLLED NETWORKS FOR CONSTRAINED OPTIMIZATIONS

We design a pair of unrolled GNNs that jointly find the saddle point of (3), mimicking the DA dynamics, as illustrated in Fig. 1. The primal network, Φ_P , emulates the minimization problem with respect to \mathbf{x} , while the dual network, Φ_D , emulates the gradient ascent steps in the dual domain across its layers. Each unrolled layer consists of T graph sub-layers, each comprising a graph filter and a nonlinearity [23], followed by a readout that outputs a primal or dual estimate.

The primal network, denoted by $\Phi_P(\cdot; \cdot; \theta_P)$, predicts a K -step trajectory from an initial point $\tilde{\mathbf{x}}_0$ towards $\tilde{\mathbf{x}}_K \approx \mathbf{x}^*(\boldsymbol{\lambda})$ across its K unrolled layers. For a given dual multiplier $\boldsymbol{\lambda}$ and a problem instance \mathbf{z} , the k -th primal layer refines the estimate $\tilde{\mathbf{x}}_{k-1}$ into

$$\tilde{\mathbf{x}}_k = \Phi_P^k(\tilde{\mathbf{x}}_{k-1}, \boldsymbol{\lambda}, \mathbf{z}; \theta_P^k), \quad (5)$$

where θ_P^k contains the parameters of the graph sub-layers and readout. We use the tilde notation to distinguish the intermediate estimates produced by the primal layers from other occurrences of \mathbf{x} . As a general design principle, we incorporate residual connections between unrolled layers as they naturally mimic gradient-based updates.

The dual network, denoted by $\Phi_D(\cdot; \cdot; \theta_D)$, has L layers whose outputs constitute a trajectory in the dual domain starting from an initial point $\boldsymbol{\lambda}_0$ and ending at an estimate of the optimal multiplier, $\boldsymbol{\lambda}_L \approx \boldsymbol{\lambda}^*$. The l -th dual layer is defined as

$$\boldsymbol{\lambda}_l = \Phi_D^l(\boldsymbol{\lambda}_{l-1}, \Phi_P(\boldsymbol{\lambda}_{l-1}, \mathbf{z}; \theta_P), \mathbf{z}; \theta_D^l), \quad (6)$$

where θ_D^l is the learnable parameters. In (6), the l -th dual layer queries the primal network for its estimate $\Phi_P(\boldsymbol{\lambda}_{l-1}, \mathbf{z}; \theta_P) \approx \mathbf{x}^*(\boldsymbol{\lambda}_{l-1})$, denoted as \mathbf{x}_{l-1} . Consequently, a single forward pass of the dual network triggers L forward passes of the primal network, as illustrated in Fig. 1. The nonlinearity at the end of each dual layer is chosen as a relu function to ensure that the predicted multipliers are nonnegative.

Finally, the solution to (1) is obtained by feeding the final dual estimate, $\boldsymbol{\lambda}_L = \Phi_D(\mathbf{z}; \theta_D, \theta_P^*)$, to the primal network,

$$\mathbf{x}_L = \Phi_P(\Phi_D(\mathbf{z}; \theta_D, \theta_P), \mathbf{z}; \theta_P). \quad (7)$$

Our goal is to train the primal and dual networks such that the output of the primal network satisfies $\tilde{\mathbf{x}}_K \approx \mathbf{x}^*(\boldsymbol{\lambda})$ for any $\boldsymbol{\lambda}$, and the output of the dual network satisfies $\boldsymbol{\lambda}_L \approx \boldsymbol{\lambda}^*$ for a family of optimization problems. In addition, the predicted trajectories should follow descent and ascent dynamics with respect to the corresponding optimization variable.

4. TRAINING WITH DESCENT CONSTRAINTS

We formulate the unsupervised training problem as a nested optimization problem that mirrors the dual problem. The outer problem is the dual network training problem over a distribution of problem instances, while the inner problem trains the primal network over a joint distribution of problem instances and multipliers. Formally, the nested training problem is defined as

$$\theta_D^* \in \operatorname{argmax}_{\theta_D} \mathbb{E}_{\mathbf{z}} \left[\mathcal{L}(\Phi_P(\boldsymbol{\lambda}_L, \mathbf{z}; \theta_P^*), \boldsymbol{\lambda}_L; \mathbf{z}) \right], \quad (8)$$

$$\text{with } \theta_P^* \in \operatorname{argmin}_{\theta_P} \mathbb{E}_{\boldsymbol{\lambda}, \mathbf{z}} \left[\mathcal{L}(\Phi_P(\boldsymbol{\lambda}, \mathbf{z}; \theta_P), \boldsymbol{\lambda}; \mathbf{z}) \right], \quad (9)$$

where $\boldsymbol{\lambda}_L = \Phi_D(\mathbf{z}; \theta_D, \theta_P^*)$ is the final output of the dual network. The joint training objective in (8)–(9) guides the final outputs but

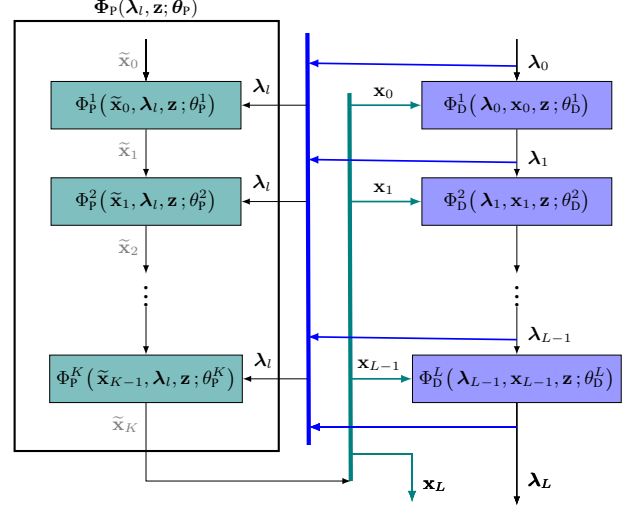


Fig. 1: The structure of the unrolled primal GNN (left) and dual GNN (right). The forward pass alternates between: i) each dual layer Φ_D^l outputs $\boldsymbol{\lambda}_l$, which is sent to the primal network Φ_P to query for the Lagrangian minimizer $\mathbf{x}^*(\boldsymbol{\lambda}_l)$, and ii) the primal network performs its internal forward pass and returns $\mathbf{x}_l \approx \mathbf{x}^*(\boldsymbol{\lambda}_l)$.

does not guarantee that the intermediate layer trajectories are monotonically descending (primal) or ascending (dual).

To address this challenge, we impose descent constraints on the primal network such that the gradient norm of the Lagrangian with respect to \mathbf{x} is forced to decrease, on average, across the layers. The primal network training is then cast as

$$\begin{aligned} \theta_P^* \in \operatorname{argmin}_{\theta_P} \mathbb{E} \left[\mathcal{L}(\Phi_P(\boldsymbol{\lambda}, \mathbf{z}; \theta_P), \boldsymbol{\lambda}; \mathbf{z}) \right], \\ \text{s.t. } \mathbb{E} \left[\|\nabla_{\mathbf{x}} \mathcal{L}(\tilde{\mathbf{x}}_k, \boldsymbol{\lambda}; \mathbf{z})\| - \alpha_k \|\nabla_{\mathbf{x}} \mathcal{L}(\tilde{\mathbf{x}}_{k-1}, \boldsymbol{\lambda}; \mathbf{z})\| \right] \leq 0, \forall k \end{aligned} \quad (10)$$

where α_k is a design parameter that controls the descent rate and $\tilde{\mathbf{x}}_0$ is initialized randomly. Analogously, we impose ascent constraints on the dual network to ensure that the gradient norm of the Lagrangian with respect to $\boldsymbol{\lambda}$, i.e., $\|\mathbf{f}(\cdot; \mathbf{z})\|$, decreases across the layers. This is a proxy for decreasing the gradient of the dual function (Danskin's theorem [24]). The dual network training is then defined as

$$\begin{aligned} \theta_D^* \in \operatorname{argmax}_{\theta_D} \mathbb{E} \left[\mathcal{L}(\Phi_P(\boldsymbol{\lambda}_L, \mathbf{z}; \theta_P^*), \boldsymbol{\lambda}_L; \mathbf{z}) \right], \\ \text{s.t. } \mathbb{E} \left[\|\mathbf{f}(\mathbf{x}_l; \mathbf{z})\| - \beta_l \|\mathbf{f}(\mathbf{x}_{l-1}; \mathbf{z})\| \right] \leq 0, \forall l, \end{aligned} \quad (11)$$

where β_l is a design parameter and $\boldsymbol{\lambda}_0$ is randomly initialized. Problems (10) and (11) are constrained nonconvex programs with small duality gap [25]. Therefore, they can be tackled in the dual domain with meta dual variables, $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$, via Algorithms 1 and 2, respectively, in which the hat notation denotes the empirical averages.

In principle, the primal network should be trained first on the joint problem-multiplier distribution before training the dual network with the primal parameters held fixed. However, the multiplier distribution is generally unknown *a priori* as it depends on the multipliers generated by the dual network during execution. The nested formulation in (8)–(9) provides a solution to this challenge by enabling an alternating training scheme. First, we freeze the primal parameters and train

Algorithm 1 Primal Network Training

```

1: Inputs:  $\theta_P, \theta_D, \mu, \epsilon_P, \eta_P$ 
2: for each epoch do
3:   for each primal batch do
4:     Sample  $\{\mathbf{z}_{(j)}\}_{j=1}^N \sim \mathcal{D}_Z$ 
5:     Sample  $\{\lambda_{(i,j)}\}_{i=1, j=1}^{M,N}$  from the trajectories by  $\theta_D$ 
6:     Execute the primal network to generate  $\{\tilde{\mathbf{x}}_{k,(i,j)}\}_{k,i,j}$ 
7:      $\ell(\theta_P) \leftarrow \hat{\mathcal{L}}(\tilde{\mathbf{x}}_K, \lambda; \mathbf{z})$ 
8:      $\mathcal{C}_k(\theta_P, \mu) \leftarrow \mu_k \left( \|\hat{\nabla} \mathcal{L}(\tilde{\mathbf{x}}_k, \lambda; \mathbf{z})\| - \alpha_k \|\hat{\nabla} \mathcal{L}(\tilde{\mathbf{x}}_{k-1}, \lambda; \mathbf{z})\| \right)$ 
9:      $\theta_P \leftarrow \theta_P - \epsilon_P \cdot \left( \nabla \ell(\theta_P) + \nabla_{\theta_P} \sum_k \mathcal{C}_k(\theta_P, \mu) \right)$ 
10:     $\mu \leftarrow [\mu + \eta_P \cdot \nabla_{\mu} \mathcal{C}(\theta_P, \mu)]_+$ 
11:   end for
12: end for
13: return  $\theta_P, \mu$ 

```

the dual network on the outer objective for a fixed number of epochs, producing layerwise multiplier trajectories. These trajectories then serve as training data for the primal network, which we update for a fixed number of epochs. We keep alternating between training the two networks to ensure that each is optimized with data generated by the current state of the other.

Note that although we replace the original convex problem with nonconvex training problems solved with the same algorithm we unroll, the computational cost of the iterative algorithm is now paid offline. The iterative algorithm is used once during training, after which each new instance is solved by a single forward pass of a neural network, significantly lowering per-instance latency and compute.

5. NUMERICAL RESULTS

We consider a mixed-integer quadratic program (MIQP) with linear inequality constraints, formulated as:

$$\min_{\mathbf{x}} \quad \frac{1}{2} \mathbf{x}^\top \mathbf{P} \mathbf{x} + \mathbf{q}^\top \mathbf{x} \quad (12)$$

$$\text{s.t.} \quad \bar{\mathbf{A}} \mathbf{x} \leq \bar{\mathbf{b}}, \quad (13)$$

$$x_i \in \{-1, 1\}, \forall i \in \mathcal{I}, \quad (14)$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{q} \in \mathbb{R}^n$, $\bar{\mathbf{A}} \in \mathbb{R}^{m \times n}$, $\bar{\mathbf{b}} \in \mathbb{R}^m$, $\mathbf{P} \in \mathbb{R}^{n \times n}$ is PSD, and \mathcal{I} is a set that contains the indices of the binary variables with cardinality $|\mathcal{I}| = r \leq n$. Solving the MIQP in (12)–(14) is NP-hard because of the binary constraints. To mitigate this issue, we consider a convex relaxation of (14) in the form of box constraints, i.e., $-1 \leq x_i \leq 1$, $\forall i \in \mathcal{I}$. The linear constraints in (13) can be combined with the new ones, producing the relaxed MIQP problem,

$$\min_{\mathbf{x}} \quad \frac{1}{2} \mathbf{x}^\top \mathbf{P} \mathbf{x} + \mathbf{q}^\top \mathbf{x} \quad \text{s.t.} \quad \mathbf{A} \mathbf{x} \leq \mathbf{b}, \quad (15)$$

where $\mathbf{A} \in \mathbb{R}^{(m+2r) \times n}$ and $\mathbf{b} \in \mathbb{R}^{m+2r}$. Specifically, we define $\mathbf{A} = [\bar{\mathbf{A}}; \mathbf{M}; -\mathbf{M}]$ and $\mathbf{b} = [\bar{\mathbf{b}}; \mathbf{1}_r; \mathbf{1}_r]$. Here, $\mathbf{M} \in \{0, 1\}^{r \times n}$ is a selection matrix whose j -th row is the standard basis vector $\mathbf{e}_{i_j}^\top$ for $i_j \in \mathcal{I}$, and $\mathbf{1}_r$ is an r -dimensional all-ones vector. The Lagrangian function associated with the relaxed problem (15) is defined as $\mathcal{L}(\mathbf{x}, \lambda) = \frac{1}{2} \mathbf{x}^\top \mathbf{P} \mathbf{x} + \mathbf{q}^\top \mathbf{x} + \lambda^\top (\mathbf{A} \mathbf{x} - \mathbf{b})$.

To facilitate using primal and dual GNNs, we model the MIQP problem with the graph adjacency, where the n variable nodes preceding the $m + 2r$ constraint nodes for notational convenience,

$$\mathbf{S} = \begin{bmatrix} \mathbf{P} & \mathbf{A}^\top \\ \mathbf{A} & \mathbf{0} \end{bmatrix}. \quad (16)$$

Algorithm 2 Dual Network Training

```

1: Inputs:  $\theta_P, \theta_D, \nu, \epsilon_D, \eta_D$ 
2: for each epoch do
3:   for each dual batch do
4:     Sample  $\{\mathbf{z}_{(i)}\}_{i=1}^N \sim \mathcal{D}_Z$ 
5:     Execute the networks to generate  $\{(\mathbf{x}_{l,(i)}, \lambda_{l,(i)})\}_{l,i}$ 
6:      $\ell(\theta_D) \leftarrow -\hat{\mathcal{L}}(\mathbf{x}_L, \lambda_L; \mathbf{z})$ 
7:      $\mathcal{C}(\theta_D, \nu) \leftarrow \sum_l \nu_l \cdot \mathbb{E} \left[ \|\mathbf{f}(\mathbf{x}_l; \mathbf{z})\| - \beta_l \|\mathbf{f}(\mathbf{x}_{l-1}; \mathbf{z})\| \right]$ 
8:      $\theta_D \leftarrow \theta_D - \epsilon_D \cdot \left( \nabla \ell(\theta_D) + \nabla_{\theta_D} \mathcal{C}(\theta_D, \nu) \right)$ 
9:      $\nu \leftarrow [\nu + \eta_D \cdot \nabla_{\nu} \mathcal{C}(\theta_D, \nu)]_+$ 
10:   end for
11: end for
12: return  $\theta_D, \nu$ 

```

In our models, the ℓ -th unrolled layer consists of a cascade of T graph convolutional sub-layers [26]. The t -th sub-layer filters the previous output $\mathbf{X}_{t-1}^{(\ell)}$ and maps it to

$$\mathbf{X}_t^{(\ell)} = \varphi \left(\sum_{h=0}^{K_h} \mathbf{S}^h \mathbf{X}_{t-1}^{(\ell)} \Theta_{t,h}^{(\ell)} \right), \quad (17)$$

where $\Theta_{t,h}^{(\ell)} \in \mathbb{R}^{F_{t-1} \times F_t}$ is the set of learnable parameters, K_h represents the filter taps, and φ is a nonlinear activation function.

In the primal network, the input to the k th unrolled layer is

$$\tilde{\mathbf{X}}_0^{(k)} = \begin{bmatrix} \tilde{\mathbf{x}}_{k-1} & \mathbf{q} \\ \lambda & \mathbf{b} \end{bmatrix}, \quad (18)$$

where $\tilde{\mathbf{x}}_{k-1}$ is the output of the previous unrolled layer, and λ, \mathbf{q} and \mathbf{b} are input data. The output of the unrolled layer is then

$$\tilde{\mathbf{x}}_k = \tilde{\mathbf{x}}_{k-1} + \mathbf{M}_P \tilde{\mathbf{X}}_T^{(k)} \mathbf{W}_k + \mathbf{c}_k, \quad (19)$$

where $\tilde{\mathbf{X}}_T^{(k)}$ is the output of the T -th graph sub-layer, and $\mathbf{W}_k \in \mathbb{R}^{F_T}$ and $\mathbf{c}_k \in \mathbb{R}^n$ are the parameters of the readout layer. The selection matrix \mathbf{M}_P extracts the outputs associated with the n variable nodes. Similarly, the input to each unrolled dual layer is constructed as

$$\mathbf{X}_0^{(l)} = \begin{bmatrix} \mathbf{x}_{l-1} & \mathbf{q} \\ \lambda_{l-1} & \mathbf{b} \end{bmatrix}, \quad (20)$$

where λ_{l-1} is the previous dual estimate and \mathbf{x}_{l-1} is the corresponding estimate of the primal network. The output of the unrolled layer is expressed as

$$\lambda_l = \varphi_{\text{relu}} \left(\mathbf{y}_{l-1} + \mathbf{M}_D \mathbf{X}_T^{(l)} \mathbf{W}_l + \mathbf{c}_l \right), \quad (21)$$

where \mathbf{M}_D selects the constraint-node values, and $\mathbf{W}_l \in \mathbb{R}^{F_T}$ and $\mathbf{c}_l \in \mathbb{R}^{m+2r}$ are learnable parameters—distinct from those of the primal layers despite the shared notation.

Experiment Setup. We construct a dataset of 800 MIQP problems with $n = 80$ decision variables, $m = 45$ linear inequality constraints and $r = 10$ binary-valued variables. Both primal and dual networks consist of $K = 14$ unrolled layers with $T = 3$ graph sub-layers. In both networks, each graph filter aggregates information from $K_h = 1$ hop neighbors, processes $F_l = F_k = 32$ hidden features, and is followed by a tanh activation function. We set the learning rates to: $\epsilon_P = 10^{-4}$, $\epsilon_D = 7 \times 10^{-4}$, $\eta_P = 10^{-4}$ and

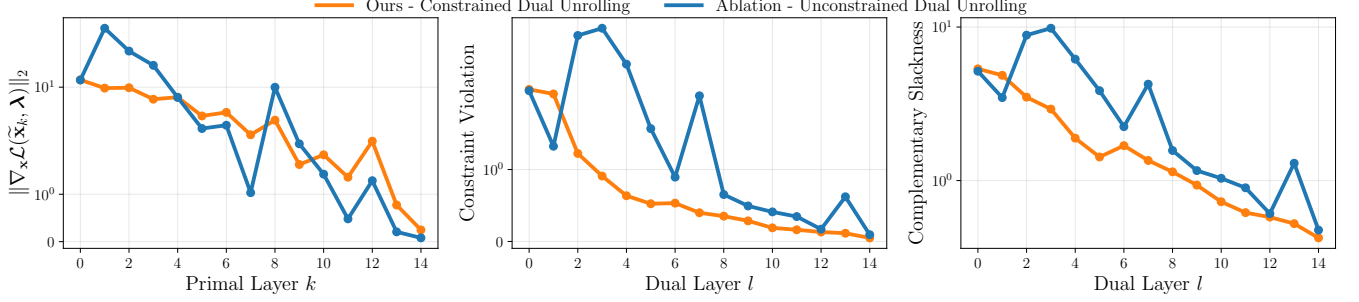


Fig. 2: Descent Guarantees. (Left) The gradient norm of the Lagrangian across the primal layers. (Middle) The constraint violation across the dual layers. (Right) The complementary slackness $\lambda_L^T \mathbf{f}(\mathbf{x}_l)$ across the unrolled dual layers. The constrained model exhibits a consistent decrease in all three quantities across layers, whereas the unconstrained model shows a more oscillatory pattern.

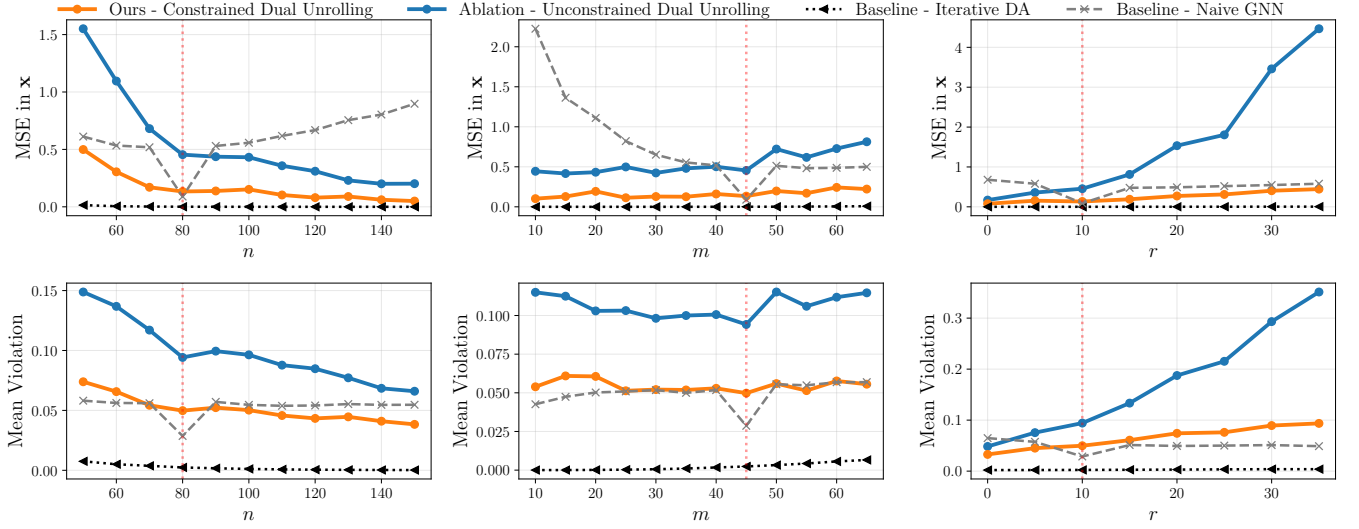


Fig. 3: Robustness under OOD problems, varying (left) the number of optimization variables n , (middle) the number of linear constraints m , and (right) the number of integer-valued variables r . The red dotted line represents the in-distribution scenario: $n = 80$, $m = 45$, and $r = 10$. Our constrained dual unrolling outperforms the other learning-based methods in optimality (top) and feasibility (bottom) across all OOD scenarios.

$\eta_D = 10^{-3}$. The constraint parameters $\alpha_k = 0.98$ and $\beta_l = 0.95$ are shared across the layers.

Performance. We evaluate the trained primal and dual GNNs over a testset of unseen 400 problems. Fig. 2 shows that our models satisfy the descent constraints on the testset as they exhibit a consistent decrease in the Lagrangian gradient norm and in the mean constraint violations across the layers. In contrast, the ablated model, trained without descent constraints, exhibits more pronounced oscillations. These differences are not aesthetic, as the unconstrained model does not reach the same performance as the constrained one in predicting the true $(\mathbf{x}^*, \lambda^*)$ pairs. This gap is more evident in Fig. 3 (the red dotted lines), where the constrained model attains a mean-squared error (MSE) of 0.133 in \mathbf{x} and a mean constraint violation of 0.049, compared to 0.454 and 0.094 for the unconstrained model.

Fig. 3 also shows OOD results obtained by varying one problem parameter while keeping the others fixed. The distribution shift intensifies as $(m + 2r)/n$ increases, yielding more challenging instances. Across all OOD scenarios, our constrained model consistently outperforms its unconstrained counterpart in optimality and feasibility. The performance gap widens as the constraint-to-variable

ratio $(m + 2r)/n$ increases, suggesting that the descent constraints confer OOD robustness. The constrained model also surpasses a standard GNN trained with a supervised loss [27] to predict the primal solutions. This is attributed to our unsupervised training objectives and to jointly training a dual network that predicts the optimal dual variable along with the primal solution. When the ODD multiplier distribution remains close to that of the in-distribution case, our model can produce solutions that match—or even exceed—in-distribution performance, whereas the naive GNN does not realize these gains.

6. CONCLUSIONS

This paper introduced two coupled GNNs trained to jointly find the Lagrangian saddle point and forced to mimic the DA dynamics by imposing layerwise monotonic descent and ascent constraints. Once trained, solving new instances reduces to a single forward pass, significantly cutting per-instance latency and compute. In MIQP experiments, the constrained models consistently outperformed both unconstrained unrolling and supervised GNN in OOD robustness (e.g., lower MSE and constraint violation).

7. REFERENCES

- [1] Vishal Monga, Yuelong Li, and Yonina C. Eldar, “Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing,” *IEEE Signal Processing Magazine*, vol. 38, no. 2, pp. 18–44, Mar. 2021.
- [2] Karol Gregor and Yann LeCun, “Learning fast approximations of sparse coding,” in *Proceedings of the 27th International Conference on Machine Learning*, June 2010, ICML’10, pp. 399–406.
- [3] Kai Zhang, Luc Van Gool, and Radu Timofte, “Deep unfolding network for image super-resolution,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 3217–3226.
- [4] Chong Mou, Qian Wang, and Jian Zhang, “Deep generalized unfolding networks for image restoration,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 17399–17410.
- [5] Marion Savanier, Emilie Chouzenoux, Jean-Christophe Pesquet, and Cyril Riddell, “Deep unfolding of the dbfb algorithm with application to roi ct imaging with limited angular density,” *IEEE Transactions on Computational Imaging*, vol. 9, pp. 502–516, 2023.
- [6] Ping Wang, Lishun Wang, Gang Qu, Xiaodong Wang, Yulun Zhang, and Xin Yuan, “Proximal algorithm unrolling: Flexible and efficient reconstruction networks for single-pixel imaging,” in *Proceedings of the Computer Vision and Pattern Recognition Conference*, 2025, pp. 411–421.
- [7] Samar Hadou, Navid NaderiAlizadeh, and Alejandro Ribeiro, “Robust stochastically-descending unrolled networks,” *IEEE Transactions on Signal Processing*, vol. 72, pp. 5484–5499, 2024.
- [8] Xiaoyang Wang and Hongping Gan, “UFC-NET: Unrolling fixed-point continuous network for deep compressive sensing,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 25149–25159.
- [9] Qingping Zhou, Jiayu Qian, Junqi Tang, and Jinglai Li, “Deep unrolling networks with recurrent momentum acceleration for nonlinear inverse problems,” *Inverse Problems*, vol. 40, no. 5, pp. 055014, 2024.
- [10] Chengyu Fang, Chunming He, Fengyang Xiao, Yulun Zhang, Longxiang Tang, Yuelin Zhang, Kai Li, and Xiu Li, “Real-world image dehazing with coherence-based pseudo labeling and cooperative unfolding network,” *Advances in Neural Information Processing Systems*, vol. 37, pp. 97859–97883, 2024.
- [11] Chunming He, Rihan Zhang, Fengyang Xiao, Chenyu Fang, Longxiang Tang, Yulun Zhang, Linghe Kong, Deng-Ping Fan, Kai Li, and Sina Farsiu, “RUN: Reversible unfolding network for concealed object segmentation,” in *Proceedings of the 42nd International Conference on Machine Learning*, 2025.
- [12] Masako Kishida and Masaki Ogura, “Temporal deep unfolding for constrained nonlinear stochastic optimal control,” *IET Control Theory & Applications*, vol. 16, no. 2, pp. 139–150, 2022.
- [13] Carla Bertocchi, Emilie Chouzenoux, Marie-Caroline Corbineau, Jean-Christophe Pesquet, and Marco Prato, “Deep unfolding of a proximal interior point method for image restoration,” *Inverse Problems*, vol. 36, no. 3, pp. 034005, 2020.
- [14] Samar Hadou, Navid NaderiAlizadeh, and Alejandro Ribeiro, “Stochastic unrolled federated learning,” *arXiv preprint arXiv:2305.15371*, 2023.
- [15] Masatoshi Nagahama, Koki Yamada, Yuichi Tanaka, Stanley H Chan, and Yonina C Eldar, “Graph signal restoration using nested deep algorithm unrolling,” *IEEE transactions on signal processing*, vol. 70, pp. 3296–3311, 2022.
- [16] Jianjun Zhang, Christos Masouros, Fan Liu, Yongming Huang, and A. Lee Swindlehurst, “Low-complexity joint radar-communication beamforming: From optimization to deep unfolding,” *IEEE Journal of Selected Topics in Signal Processing*, pp. 1–16, 2025.
- [17] Junwen Yang, Ang Li, Xuewen Liao, and Christos Masouros, “ADMM-SLPNet: A model-driven deep learning framework for symbol-level precoding,” *IEEE Transactions on Vehicular Technology*, vol. 73, no. 1, pp. 1376–1381, 2024.
- [18] Bingheng Li, Linxin Yang, Yupeng Chen, Senmiao Wang, Haitao Mao, Qian Chen, Yao Ma, Akang Wang, Tian Ding, Jiliang Tang, et al., “PDHG-unrolled learning-to-optimize method for large-scale linear programming,” in *Proceedings of the 41st International Conference on Machine Learning*. PMLR, 2024, pp. 29164–29180.
- [19] Quentin Cappart, Didier Chételat, Elias B Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković, “Combinatorial optimization and reasoning with graph neural networks,” *Journal of Machine Learning Research*, vol. 24, no. 130, pp. 1–61, 2023.
- [20] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi, “Exact combinatorial optimization with graph convolutional neural networks,” in *Advances in neural information processing systems*, 2019, vol. 32.
- [21] Ziang Chen, Jialin Liu, Xinshang Wang, and Wotao Yin, “On representing linear programs by graph neural networks,” in *International Conference on Learning Representations*, 2023.
- [22] Stephen P Boyd and Lieven Vandenbergh, *Convex optimization*, Cambridge university press, 2004.
- [23] Samar Hadou, Charilaos I Kanatsoulis, and Alejandro Ribeiro, “Space-time graph neural networks with stochastic graph perturbations,” in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023, pp. 1–5.
- [24] Dimitri P Bertsekas, “Nonlinear programming,” *Journal of the Operational Research Society*, vol. 48, no. 3, pp. 334–334, 1997.
- [25] Luiz FO Chamon, Santiago Paternain, Miguel Calvo-Fullana, and Alejandro Ribeiro, “Constrained learning with non-convex losses,” *IEEE Transactions on Information Theory*, 2022.
- [26] Fernando Gama, Elvin Isufi, Geert Leus, and Alejandro Ribeiro, “Graphs, convolutions, and neural networks: From graph filters to graph neural networks,” *IEEE Signal Processing Magazine*, vol. 37, pp. 128–138, Nov. 2020.
- [27] Ziang Chen, Xiaohan Chen, Jialin Liu, Xinshang Wang, and Wotao Yin, “Expressive power of graph neural networks for (mixed-integer) quadratic programs,” in *Proceedings of the 42nd International Conference on Machine Learning*, 2025, pp. 1–32.