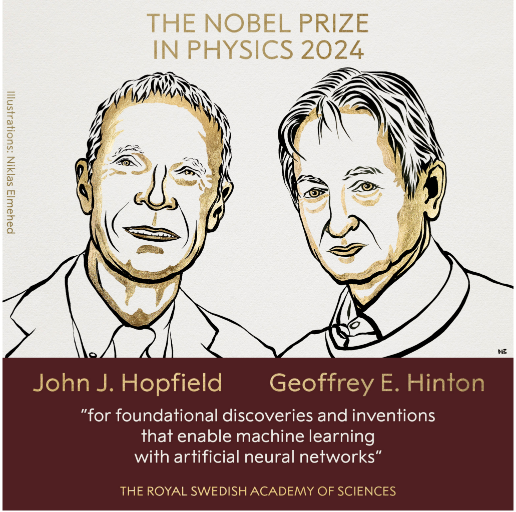
Learning-based Approaches for Linear Programming: A Survey

Zhonglin Xie

Beijing International Center for Mathematical Research
Peking University

October 9, 2024

ANNs are powerful tools in physics and other scientific disciplines



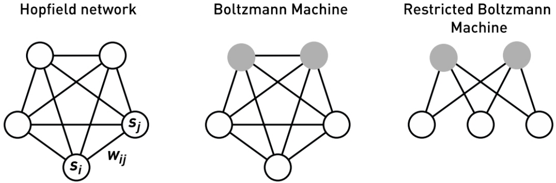


Figure: Recurrent networks of N binary nodes s_i (0 or 1), with connection weights w_{ij}.

- Hopfield Network: $E = -\frac{1}{2}\sum_{i,j} w_{ij} s_i s_j$
- Boltzmann Machine:
  $E = -\sum_{i,j} w_{ij} s_i s_j - \sum_i \theta_i s_i$
- Restricted Boltzmann Machine (RBM):
  $E = -\sum_{i,j} w_{ij} v_i h_j - \sum_i b_i v_i - \sum_j c_j h_j$

Outline

Standard Form of a Linear Programming Problem

$$\begin{aligned} \min_{x} \quad & c^\top x, \\ \text{s.t.} \quad & Ax \leq b, \\ & x \geq 0. \end{aligned} \tag{SF}$$

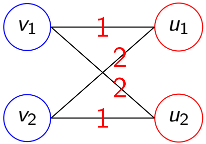
- $x \in \mathbb{R}^n$ is the decision variable
- $A \in \mathbb{R}^{m \times n}$ is a matrix of constraint coefficients
- $c \in \mathbb{R}^n$ is a vector of coefficients for the objective function
- we denote the instance (SF) as $I = (A, b, c)$

Weighted Bipartite Graphs

- A weighted bipartite graph is a tuple (U, V, E, w)
- U and V are two disjoint sets, and $U \cup V$ contains all vertices
- $E \subseteq U \times V$ is the set of edges, where each edge connects a vertex in U to a vertex in V
- $w : E \to \mathbb{R}$ is a function that assigns the weight for each edge

$$\begin{aligned} \min_{x \in \mathbb{R}^2} \quad & 2x_1 + 3x_2, \\ \text{s.t.} \quad & x_1 + 2x_2 \leq 1, \\ & 2x_1 + x_2 \leq 2, \\ & x_1 \geq 0, \ x_2 \geq 0. \end{aligned}$$

$h_1^V = (2)$ $h_2^V = (3)$



$h_1^U = (1)$ $h_2^U = (2)$

Figure: An example of LP-graph

Encoding LPs as Graphs

- $U = \{u_1, u_2, \dots, u_m\}$ represents the n dimensions of the decision variable x
- $V = \{v_1, v_2, \dots, v_n\}$ corresponds to the m inequality constraints
- The set E contains $m \times n$ edges. We denote $E_{i,j} = (u_i, v_j)$
- The function w assigns the weights according to $w(E_{i,j}) = A_{i,j}$
- For i-th constraint, the node u_i is associated with a feature vector $h_i^U = (b_i) \in \mathcal{H}^U$
- For j-variable, the node v_j has a feature vector $h_j^V = (c_j) \in \mathcal{H}^V$

Graph neural networks for LP

- Embedding: Given learnable functions $f_{\rm in}^U : \mathcal{H}^U \to \mathbb{R}^{d_0}$ and $f_{\rm in}^V : \mathcal{H}^V \to \mathbb{R}^{d_0}$,

$$h_i^{0,U} = f_{\rm in}^U(h_i^U), \quad h_j^{0,V} = f_{\rm in}^V(h_j^V), \quad i = 1, 2, \ldots, m, \; j = 1, 2, \ldots, n$$

- Update the hidden states: Let $f_l^U, f_l^V : \mathbb{R}^{d_{l-1}} \to \mathbb{R}^{d_l}$ and $g_l^U, g_l^V : \mathbb{R}^{d_{l-1}} \times \mathbb{R}^{d_l} \to \mathbb{R}^{d_l}$,

$$h_i^{l,U} = g_l^U\left(h_i^{l-1,U}, \sum_{j=1}^{n} E_{i,j} f_l^V(h_j^{l-1,V})\right), \quad i = 1, 2, \ldots, m, \quad l = 1, 2, \ldots, L$$

$$h_j^{l,V} = g_l^V\left(h_j^{l-1,V}, \sum_{i=1}^{m} E_{i,j} f_l^U(h_i^{l-1,U})\right), \quad j = 1, 2, \ldots, n, \quad l = 1, 2, \ldots, L$$

- Output layer of the single-output GNN: Learnable function $f_{\rm out} : \mathbb{R}^{d_L} \times \mathbb{R}^{d_L} \to \mathbb{R}$:

$$y_{\rm out} = f_{\rm out}\left(\sum_{i=1}^{m} h_i^{L,U}, \sum_{j=1}^{n} h_j^{L,V}\right)$$

Graph Neural Networks for LP

- Output of the vertex-output GNN is defined with $f_{\text{out}}^V : \mathbb{R}^{d_L} \times \mathbb{R}^{d_L} \times \mathbb{R}^{d_L} \to \mathbb{R}$:

$$y_{\text{out}}(v_j) = f_{\text{out}}^V\left(\sum_{i=1}^m h_i^{L,U}, \sum_{j=1}^n h_j^{L,V}, h_j^{L,V}\right), \quad j = 1, 2, \cdots, n$$

- Denote collections of single-output and vertex-output GNNs with $\mathcal{F}_{\text{GNN}}$ and $\mathcal{F}_{\text{GNN}}^V$:

$$\mathcal{F}_{\text{GNN}} = \{F : \mathcal{G}_{m,n} \times \mathcal{H}_m^U \times \mathcal{H}_n^V \to \mathbb{R} \mid F \text{ yields single-output } \}$$
$$\mathcal{F}_{\text{GNN}}^V = \{F : \mathcal{G}_{m,n} \times \mathcal{H}_m^U \times \mathcal{H}_n^V \to \mathbb{R}^n \mid F \text{ yields vertex-output } \}$$

Definition

- **Feasibility mapping.** The feasibility mapping is a classification function
$$\Phi_{\text{feas}} : \mathcal{G}_{m,n} \times \mathcal{H}_m^U \times \mathcal{H}_n^V \to \{0, 1\},$$
where $\Phi_{\text{feas}}(G, H) = 1$ if the LP is feasible and $\Phi_{\text{feas}}(G, H) = 0$ otherwise

- **Optimal objective value mapping.** Denote
$$\Phi_{\text{obj}} : \mathcal{G}_{m,n} \times \mathcal{H}_m^U \times \mathcal{H}_n^V \to \mathbb{R} \cup \{\infty, -\infty\},$$
as the optimal objective value mapping. For any $(G, H) \in \mathcal{G}_{m,n} \times \mathcal{H}_m^U \times \mathcal{H}_n^V$, $\Phi_{\text{obj}}(G, H)$ is the optimal objective value of the LP problem associated with (G, H)

- **Optimal solution mapping.** For any $(G, H) \in \Phi_{\text{obj}}^{-1}(\mathbb{R})$. The mapping
$$\Phi_{\text{solu}} : \Phi_{\text{obj}}^{-1}(\mathbb{R}) \to \mathbb{R}^n,$$
maps $(G, H) \in \Phi_{\text{obj}}^{-1}(\mathbb{R})$ to the optimal solution with the smallest ℓ_2-norm

GNN has strong enough separation power to represent LP

Theorem

Given any two LP instances $(G, H), (\hat{G}, \hat{H}) \in \mathcal{G}_{m,n} \times \mathcal{H}_m^U \times \mathcal{H}_n^V$*, if* $F(G, H) = F(\hat{G}, \hat{H})$ *for all* $F \in \mathcal{F}_{GNN}$*, then they share some common characteristics:*

(i) *Both LP problems are feasible or both are infeasible, i.e.,* $\Phi_{feas}(G, H) = \Phi_{feas}(\hat{G}, \hat{H})$*.*

(ii) *The two LP problems have the same optimal objective value:* $\Phi_{obj}(G, H) = \Phi_{obj}(\hat{G}, \hat{H})$*.*

(iii) *If both problems are feasible and bounded, they have the same optimal solution with the smallest* ℓ_2*-norm up to a permutation, i.e.,* $\Phi_{solu}(G, H) = \sigma_V(\Phi_{solu}(\hat{G}, \hat{H}))$ *for some* $\sigma_V \in S_n$*.*

Furthermore, if $F_V(G, H) = F_V(\hat{G}, \hat{H}), \forall F_V \in \mathcal{F}_{GNN}^V$*, then (iii) holds without taking permutations, i.e.,* $\Phi_{solu}(G, H) = \Phi_{solu}(\hat{G}, \hat{H})$*.*

GNN is a good classifier for LP instances

Theorem

Given any measurable $X \subset \mathcal{G}_{m,n} \times \mathcal{H}_m^U \times \mathcal{H}_n^V$ with finite measure, for any $\epsilon > 0$, there exists some $F \in \mathcal{F}_{GNN}$, such that

$$Meas\left(\{(G, H) \in X : \mathbb{I}_{F(G,H)>1/2} \neq \Phi_{feas}(G, H)\}\right) < \epsilon,$$

where $\mathbb{I}_{\cdot}$ is the indicator function, i.e., $\mathbb{I}_{F(G,H)>1/2} = 1$ if $F(G, H) > 1/2$ and $\mathbb{I}_{F(G,H)>1/2} = 0$ otherwise.

Corollary

For any $\mathcal{D} \subset \mathcal{G}_{m,n} \times \mathcal{H}_m^U \times \mathcal{H}_n^V$ with finite instances, there exists $F \in \mathcal{F}_{GNN}$ that

$$\mathbb{I}_{F(G,H)>1/2} = \Phi_{feas}(G, H), \quad \forall\ (G, H) \in \mathcal{D}.$$

GNN can approximate optimal objective value and solution mappings

Corollary

For any $\mathcal{D} \subset \mathcal{G}_{m,n} \times \mathcal{H}_m^U \times \mathcal{H}_n^V$ *with finite instances, there exists* $F_1 \in \mathcal{F}_{GNN}$ *such that*

$$\mathbb{I}_{F_1(G,H)>1/2} = \mathbb{I}_{\Phi_{obj}(G,H)\in\mathbb{R}}, \quad \forall\, (G,H) \in \mathcal{D},$$

and for any $\delta > 0$*, there exists* $F_2 \in \mathcal{F}_{GNN}$*, such that*

$$|F_2(G,H) - \Phi_{obj}(G,H)| < \delta, \quad \forall\, (G,H) \in \mathcal{D} \cap \Phi_{obj}^{-1}(\mathbb{R}).$$

Corollary

Given any $\mathcal{D} \subset \Phi_{obj}^{-1}(\mathbb{R}) \subset \mathcal{G}_{m,n} \times \mathcal{H}_m^U \times \mathcal{H}_n^V$ *with finite instances, for any* $\delta > 0$*, there exists* $F_V \in \mathcal{F}_{GNN}^V$*, such that*

$$\|F(G,H) - \Phi_{solu}(G,H)\| < \delta, \quad \forall\, (G,H) \in \mathcal{D}.$$

Outline

An ODE-based Method for Solving LPs

- Consider the LP of the general from

$$\begin{aligned} \min_{x} \quad & c^\top x, \\ \text{s.t.} \quad & Ax \leq b \end{aligned}$$

- The corresponding Lagrangian is

$$L(x, u) = c^\top x + u^\top (Ax - b)$$

- The corresponding KKT conditions are

$$\begin{aligned} c + A^\top u &= 0, \\ u^\top (Ax - b) &= 0, \\ Ax - b &\leq 0, \\ u &\geq 0 \end{aligned}$$

ODE system for modeling LPs

- Let $x(t) : \mathbb{R} \to \mathbb{R}^n$, $u(t) : \mathbb{R} \to \mathbb{R}^m$, and $y(t) = \left(x(t)^\top, u(t)^\top\right)^\top$
- The following ODE system models the LP problem via the KKT conditions

$$\frac{\mathrm{d}y}{\mathrm{d}t} = \Phi(y) = \begin{pmatrix} \frac{\mathrm{d}x}{\mathrm{d}t} \\ \frac{\mathrm{d}u}{\mathrm{d}t} \end{pmatrix} = \begin{pmatrix} -\left(c + A^\top (u + Ax - b)^+\right) \\ (u + Ax - b)^+ - u \end{pmatrix}, \quad y(t_0) = y_0, \quad t \in [0, T]$$

Theorem

$y^* = \left((x^*)^\top, (u^*)^\top\right)^\top$ is an equilibrium point of the ODE system if and only if it is a satisfied point of the KKT conditions. Furthermore, given arbitrary initial point, the ODE system satisfies

$$\lim_{t\to\infty} \mathit{dist}(y(t), \Theta^*) = 0, \ \textit{where } \Theta^* = \{y^* | y^* = (x^*)^\top, (u^*)^\top \textit{ solves the KKT conditions }\}.$$

PINNs for Linear Programming: A Proof of Concept

- Let the neural network model be defined as

$$\hat{y}(t, I; \mathbf{w}) = (1 - e^{-t})\mathrm{NN}(t, I; \mathbf{w}), \quad t \in [0, T]$$

- The multiplier $(1 - e^{-t})$ guarantees $\hat{y}(0, I; \mathbf{w}) = 0$ regardless of weights $\mathbf{w}$
- The endpoint

$$\hat{y}(T, I; \mathbf{w}) = (1 - e^{-T})\mathrm{NN}(T, I; \mathbf{w})$$

represents an approximate solution to the KKT conditions associated with instance I

- The loss function is defined as

$$E(\mathbf{w}) = \frac{1}{|D| * |\mathbb{T}|} \sum_{I_j \in D} \sum_{t_i \in \mathbb{T}} \ell\left(\frac{\partial \hat{y}(t_i, I_j; \mathbf{w})}{\partial t}, \Phi_j(\hat{y}(t_i, I_j; \mathbf{w}))\right)$$

where D refers to the set of instances, and each $I_j \in \Theta$ relate to an ODE system Φ_j

Algorithmic Framework

Algorithm Solving LP problems by neural networks

Require: A time range $[0, T]$
Require: A probability distribution P for generating I
  Function Main:
  while True **do**
    Generate D, a set of $I \sim P$ according to the given probability distribution P
    Uniformly sample $\mathbb{T}$, a batch of $t \sim U(0, T)$ from the interval $[0, T]$
    Forward propagation: Compute $E(w)$ based on the D and $\mathbb{T}$
    Backward propagation: Update w by $\nabla E(w)$
    Stopping criteria check
  end while

Outline

Interior-point Methods (IPMs) for Linear Optimization

- An instance I of an LP is a tuple $(A, \mathbf{b}, \mathbf{c})$, where $A \in \mathbb{Q}^{m \times n}$, and $\mathbf{b} \in \mathbb{Q}^m$ and $\mathbf{c} \in \mathbb{Q}^n$
- Linear optimization: finding a vector $\mathbf{x}^*$ in $\mathbb{Q}^n$ that minimizes $\mathbf{c}^\top \mathbf{x}^*$ over the *feasible set*

$$F(I) = \{\mathbf{x} \in \mathbb{Q}^n \mid A_j \mathbf{x} \leq b_j \text{ for } j \in [m] \text{ and } x_i \geq 0 \text{ for } i \in [n]\}$$

- Consider a perturbed version of the LP for some $\mu > 0$:

$$\min_{\mathbf{x} \in \mathbb{Q}^n} \mathbf{c}^\top \mathbf{x} - \mu[\mathbf{1}^\top \log(\mathbf{b} - A\mathbf{x}) + \mathbf{1}^\top \log(\mathbf{x})]$$

- Let $s_i = \mu / x_i$, $r_j = A_j \mathbf{x} - b_j$, and $w_j = \mu / r_j$. The first-order optimality conditions write

$$\begin{aligned} Ax^* - r^* &= b, & & \\ A^\top w^* + s^* &= c, & x^*, w^*, s^*, r^* &\geq 0, \\ x_i^* s_i^* &= \mu, & i &\in [n], \\ w_i^* r_i^* &= \mu, & j &\in [m] \end{aligned}$$

Interior-point Methods (IPMs) for Linear Optimization

1. Let $\sigma \in (0,1)$. IPMs start from an initial positive point $(x_0, w_0, s_0, r_0) > 0$

2. Compute the Newton step for the perturbed problem at barrier parameter $\sigma\mu$

$$\begin{bmatrix} A & 0 & 0 & -I \\ 0 & A^\top & I & 0 \\ D(s) & 0 & D(x) & 0 \\ 0 & D(r) & 0 & D(w) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta w \\ \Delta s \\ \Delta r \end{bmatrix} = \begin{bmatrix} b - Ax + r \\ c - A^\top w - s \\ \sigma\mu\mathbf{1} - D(x)D(s)\mathbf{1} \\ \sigma\mu\mathbf{1} - D(w)D(r)\mathbf{1} \end{bmatrix}$$

3. Take a step in that direction with length $\alpha > 0$, such that the resulting point

$$(x', w', s', r') = (x, w, s, r) + \alpha(\Delta x, \Delta w, \Delta s, \Delta r) \text{ satisfies } (x', w', s', r') > 0$$

Interior-point Methods (IPMs) for Linear Optimization

- The above system can be simplified as follows. First, we can infer that

$$\Delta s = \sigma\mu D(x)^{-1}\mathbf{1} - s - D(x)^{-1}D(s)\Delta x,$$
$$\Delta r = \sigma\mu D(w)^{-1}\mathbf{1} - r - D(w)^{-1}D(r)\Delta w,$$

which implies that

$$A\Delta x + D(w)^{-1}D(r)\Delta w = b - Ax + \sigma\mu D(w)^{-1}\mathbf{1},$$
$$A^\top \Delta w - D(x)^{-1}D(s)\Delta x = c - A^\top w - \sigma\mu D(x)^{-1}\mathbf{1}$$

- Therefore, for $Q = AD(s)^{-1}D(x)A^\top + D(w)^{-1}D(r)$, we only need to compute

$$\Delta x = D(s)^{-1}D(x)[A^\top \Delta w - c + A^\top w + \sigma\mu D(x)^{-1}\mathbf{1}],$$
$$Q\Delta w = b - Ax + \sigma\mu D(w)^{-1}\mathbf{1} + AD(s)^{-1}D(x)[c - A^\top w - \sigma\mu D(x)^{-1}\mathbf{1}]$$

Algorithmic Frameworks for IPMs

Algorithm Practical IPM for LPs

Require: An LP instance $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$, a barrier reduction hyperparameter $\sigma \in (0,1)$ and initial values $(\boldsymbol{x}_0, \boldsymbol{w}_0, \boldsymbol{s}_0, \boldsymbol{r}_0, \mu_0)$ such that $(\boldsymbol{x}_0, \boldsymbol{w}_0, \boldsymbol{s}_0, \boldsymbol{r}_0) > 0$ and $\mu_0 = (\boldsymbol{x}_0^\mathsf{T} \boldsymbol{s}_0 + \boldsymbol{w}_0^\mathsf{T} \boldsymbol{r}_0)/(n+m)$

1: **repeat**

2: Compute $\Delta \boldsymbol{w}$ by solving the linear system

$$\boldsymbol{Q}\Delta\boldsymbol{w} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x} + \sigma\mu\boldsymbol{D}(\boldsymbol{w})^{-1}\mathbf{1} + \boldsymbol{A}\boldsymbol{D}(\boldsymbol{s})^{-1}\boldsymbol{D}(\boldsymbol{x})[\boldsymbol{c} - \boldsymbol{A}^\mathsf{T}\boldsymbol{w} - \sigma\mu\boldsymbol{D}(\boldsymbol{x})^{-1}\mathbf{1}]$$

for $\boldsymbol{Q} = \boldsymbol{A}\boldsymbol{D}(\boldsymbol{s})^{-1}\boldsymbol{D}(\boldsymbol{x})\boldsymbol{A}^\mathsf{T} + \boldsymbol{D}(\boldsymbol{w})^{-1}\boldsymbol{D}(\boldsymbol{r})$.

3: $\Delta\boldsymbol{x} \leftarrow \boldsymbol{D}(\boldsymbol{s})^{-1}\boldsymbol{D}(\boldsymbol{x})[\boldsymbol{A}^\mathsf{T}\Delta\boldsymbol{w} - \boldsymbol{c} + \boldsymbol{A}^\mathsf{T}\boldsymbol{w} + \sigma\mu\boldsymbol{D}(\boldsymbol{x})^{-1}\mathbf{1}]$

4: $\Delta\boldsymbol{s} \leftarrow \sigma\mu\boldsymbol{D}(\boldsymbol{x})^{-1}\mathbf{1} - \boldsymbol{s} - \boldsymbol{D}(\boldsymbol{x})^{-1}\boldsymbol{D}(\boldsymbol{s})\Delta\boldsymbol{x}$

5: $\Delta\boldsymbol{r} \leftarrow \sigma\mu\boldsymbol{D}(\boldsymbol{w})^{-1}\mathbf{1} - \boldsymbol{r} - \boldsymbol{D}(\boldsymbol{w})^{-1}\boldsymbol{D}(\boldsymbol{r})\Delta\boldsymbol{w}$

6: Find the largest $\alpha > 0$: $\min_{i,j}\{(\boldsymbol{x}+\alpha\Delta\boldsymbol{x})_i(\boldsymbol{s}+\alpha\Delta\boldsymbol{s})_i, (\boldsymbol{w}+\alpha\Delta\boldsymbol{w})_j(\boldsymbol{r}+\alpha\Delta\boldsymbol{r})_j\} \geq 0$

7: Update $(\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{s}, \boldsymbol{r}) \mathrel{+}= 0.99\alpha(\Delta\boldsymbol{x}, \Delta\boldsymbol{w}, \Delta\boldsymbol{s}, \Delta\boldsymbol{r})$, $\mu \leftarrow \sigma\mu$

8: **until** convergence of $(\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{s}, \boldsymbol{r})$

9: **Return** the point $\boldsymbol{x}$, which solves LP.

Weighted Tripartite Graph

- An undirected weighted graph has three disjoint sets of the vertices U, V and W
- Denote $U = \{u_1, u_2, \ldots, u_m\}$, $V = \{v_1, v_2, \ldots, v_n\}$ and $W = \{w_1, w_2, \ldots, w_l\}$
- Each edge connects two vertices from the different sets between them
- If there is an edge between the node u_i and v_j, we denote this edge as (u_i, v_j)
- All the edges constitute the set E and we associate each edge with a weight

Representing LPs as Weighted Tripartite Graphs

- We represent an LP instance $I = (A, b, c)$ using an undirected weighted tripartite graph $\mathcal{G}(I) = \{V(I), C(I), \{o\}, E_{\mathrm{vc}}(I), E_{\mathrm{co}}(I), E_{\mathrm{ov}}(I)\}$
- $V(I)$ contains the vertices that represent the decision variables
- $C(I)$ contains the vertices that represent the constraints
- o is a vertex that represents the objective function. For each vertex in $V(I)$ and $C(I)$, there is an edge that connects o with it

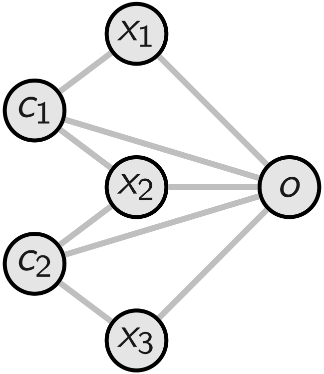


Figure: Representing an LP instance with two constraints and three variables as a tripartite graph.

Representing LPs as Weighted Tripartite Graphs

- $E_{\mathrm{vc}}(I)$ is the set that contains all edges between the vertices in $V(I)$ and $C(I)$. For the i-th vertex v_i in $V(I)$ and j-th vertex c_j in $C(I)$, if $A_{i,j}$ does not equal to 0, then there is an edge $(v_i, c_j) \in E_{\mathrm{vc}}(I)(I)$ connects them and the weight is assigned as $A_{i,j}$

- $E_{\mathrm{co}}(I)$ is the set that contains all edges between the vertices in $C(I)$ and o. The weight is set to b_j for edge (c_j, o)

- $E_{\mathrm{ov}}(I)$ is the set that contains all edges between the vertices in $V(I)$ and o. The weight is set to c_i for edge (v_i, o)

Theoretical Foundation: MPNNs can emulate IPMs

Theorem

There exists an MPNN $f_{\text{MPNN,IPM1}}$ composed of $\mathcal{O}(m)$ message-passing steps that reproduces an iteration of IPMs, in the sense that for any LP instance $I = (\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ and any iteration step $t \geq 0$, $f_{\text{MPNN,IPM1}}$ maps the graph $G(I)$ carrying $[\boldsymbol{x}_t, \boldsymbol{s}_t]$ on the variable nodes and $[\boldsymbol{w}_t, \boldsymbol{r}_t]$ on the constraint nodes to the same graph $G(I)$ carrying $[\boldsymbol{x}_{t+1}, \boldsymbol{s}_{t+1}]$ on the variable nodes and $[\boldsymbol{w}_{t+1}, \boldsymbol{r}_{t+1}]$ on the constraint nodes.

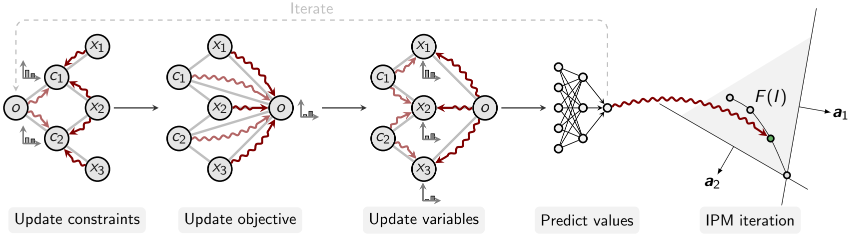


Figure: IPM-MPNNs emulate interior-point methods.

Asynchronous Updates of the IPM-MPNN

- Let $\boldsymbol{h}_c^{(t)} \in \mathbb{R}^d$, $d > 0$, be the node features of a constraint node $c \in C(I)$ at iteration $t > 0$, and let $\boldsymbol{h}_v^{(t)} \in \mathbb{R}^d$ and $\boldsymbol{h}_o^{(t)} \in \mathbb{R}^d$ be the node features of a variable node $v \in V(I)$ and the objective node o at iteration t, respectively
- Let $\boldsymbol{e}_{co}, \boldsymbol{e}_{vc}, \boldsymbol{e}_{vo}$ denote the edge weights
- The parameterized message function $\mathsf{MSG}^{(t)}_{\mathsf{v} \to \mathsf{c}}$ maps variable node features and corresponding edge features $\boldsymbol{e}_{vc}$, to a vector in $\mathbb{R}^d$
- The parameterized function $\mathsf{MSG}^{(t)}_{\mathsf{o} \to \mathsf{c}}$ maps the current node features of the objective node and edge features $\boldsymbol{e}_{oc}$ to a vector in $\mathbb{R}^d$

Asynchronous Updates of the IPM-MPNN

- The parameterized function $\mathsf{UPD}_{\mathsf{c}}^{(t)}$ maps the constraint node's previous features, the outputs of $\mathsf{MSG}_{\mathsf{o}\to\mathsf{c}}^{(t)}$ and $\mathsf{MSG}_{\mathsf{v}\to\mathsf{c}}^{(t)}$ to a vector in $\mathbb{R}^d$
- In the first pass, we update the embeddings of constraint nodes from the embeddings of the variable nodes and of the objective node. That is, let $c \in C(I)$ be a constraint node and let $t > 0$, then

$$\boldsymbol{h}_c^{(t)} := \mathsf{UPD}_{\mathsf{c}}^{(t)} \Big[\boldsymbol{h}_c^{(t-1)}, \mathsf{MSG}_{\mathsf{o}\to\mathsf{c}}^{(t)} \left(\boldsymbol{h}_o^{(t-1)}, \boldsymbol{e}_{oc} \right), \\ \mathsf{MSG}_{\mathsf{v}\to\mathsf{c}}^{(t)} \left(\{\!\{ (\boldsymbol{h}_v^{(t-1)}, \boldsymbol{e}_{vc}) \mid v \in N(c) \cap V(I) \}\!\} \right) \Big]$$

Asynchronous Updates of the IPM-MPNN

- Next, we update the objective node's features depending on variable and constraint node features,

$$\boldsymbol{h}_{\mathrm{o}}^{(t)} := \mathrm{UPD}_{\mathrm{o}}^{(t)}\Big[\boldsymbol{h}_{o}^{(t-1)}, \mathrm{MSG}_{\mathrm{c}\to\mathrm{o}}^{(t)}\left(\{\!\{\boldsymbol{h}_{c}^{(t)}, \boldsymbol{e}_{co} \mid c \in C(I)\}\!\}\right), \\ \mathrm{MSG}_{\mathrm{v}\to\mathrm{o}}^{(t)}\left(\{\!\{\boldsymbol{h}_{v}^{(t-1)}, \boldsymbol{e}_{vo} \mid v \in V(I)\}\!\}\right)\Big]$$

- Subsequently, we update the representation of a variable node $v \in V(I)$ from the constraints nodes and objective node,

$$\boldsymbol{h}_{v}^{(t)} := \mathrm{UPD}_{\mathrm{v}}^{(t)}\Big[\boldsymbol{h}_{v}^{(t-1)}, \mathrm{MSG}_{\mathrm{o}\to\mathrm{v}}^{(t)}\left(\boldsymbol{h}_{o}^{(t)}, \boldsymbol{e}_{ov}\right), \\ \mathrm{MSG}_{\mathrm{c}\to\mathrm{v}}^{(t)}\left(\{\!\{\boldsymbol{h}_{c}^{(t)}, \boldsymbol{e}_{cv} \mid c \in N(v) \cap V(C)\}\!\}\right)\Big]$$

- Finally, we map each variable node feature $\boldsymbol{h}_{v}^{(t)}$ to $\mathrm{MLP}(\boldsymbol{h}_{v}^{(t)}) \in \mathbb{R}$, and concatenate the outputs in the final prediction $\boldsymbol{z}^{(t)} \in \mathbb{R}^{n}$

Loss Function

- **Variable supervision** Let N denote the number of training samples. We set

$$\mathcal{L}_{\text{var}} := \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{\top}\alpha^{T-t}\|\boldsymbol{y}_i^{(t)} - \boldsymbol{z}_i^{(t)}\|_2^2$$

- **Objective supervision** We do not predict the objective directly but calculate it via $\boldsymbol{c}^{\mathsf{T}}\boldsymbol{z}^{(t)}$ instead. Suppose the ground-truth values are given by $\boldsymbol{c}^{\mathsf{T}}\boldsymbol{y}^{(t)}$, we have

$$\mathcal{L}_{\text{obj}} := \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{\top}\alpha^{T-t}\left[\boldsymbol{c}^{\mathsf{T}}\left(\boldsymbol{y}_i^{(t)} - \boldsymbol{z}_i^{(t)}\right)\right]^2$$

- **Constraint supervision** Finally, we penalize constraint violations:

$$\mathcal{L}_{\text{cons}} := \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{\top}\alpha^{T-t}\|\text{ReLU}(\boldsymbol{A}_i\boldsymbol{z}_i^{(t)} - \boldsymbol{b}_i)\|_2^2$$

Numerical Results

- We combine the above three loss terms into the loss function

$$\mathcal{L} := w_{\text{var}}\mathcal{L}_{\text{var}} + w_{\text{obj}}\mathcal{L}_{\text{obj}} + w_{\text{cons}}\mathcal{L}_{\text{cons}}$$

- **GCNs (Graph Convolutional Networks)**: GCNs aggregate information from neighboring nodes using a localized filter, effectively learning node representations based on their local graph structure
- **GINs (Graph Isomorphic Networks)**: GINs improve upon GCNs by using more expressive aggregation functions, enabling them to distinguish between non-isomorphic graphs that GCNs might find similar
- **GENs (Generalized Graph Neural Networks)**: GENs offer a broader framework for graph representation learning by allowing for flexible message-passing mechanisms and aggregation schemes beyond the limitations of GCNs and GINs

Numerical Results: Bipartite v.s. Tripartite

Table: Results of our proposed IPM-MPNNs (✓) versus bipartite representation ablations (✗). We report the relative objective gap and the constraint violation, averaged over all three runs. We print the best results per target in bold.

	Tri.	MPNN	Small instances Setcover	Small instances Indset	Small instances Cauc	Small instances Fac	Large instances Setcover	Large instances Indset	Large instances Cauc	Large instances Fac
Objective gap [%]	✓	GEN	**0.319±0.020**	0.119±0.003	**0.612±0.049**	**0.549±0.112**	0.629±0.086	0.158±0.035	**0.306±0.047**	**0.747±0.083**
	✓	GCN	0.418±0.008	**0.103±0.006**	0.682±0.029	0.578±0.015	**0.420±0.047**	**0.094±0.005**	0.407±0.038	0.914±0.141
	✓	GIN	0.478±0.038	0.146±0.011	0.632±0.036	0.810±0.221	0.711±0.115	0.126±0.021	0.378±0.052	0.911±0.132
	✗	GEN	8.310±1.269	0.735±0.032	1.417±0.009	2.976±0.013	15.170±6.844	0.320±0.008	0.851±0.122	2.531±0.025
	✗	GCN	5.523±0.133	0.639±0.009	1.394±0.081	3.031±0.059	6.092±0.456	0.298±0.009	0.766±0.093	2.535±0.034
	✗	GIN	5.592±0.179	0.634±0.021	1.202±0.016	2.996±0.031	5.835±1.917	0.290±0.005	0.810±0.140	2.660±0.062
Constraint violation	✓	GEN	**0.002±0.0002**	0.0006±0.00003	0.003±0.0007	0.002±0.001	0.009±0.001	0.0015±0.0003	**0.0004±0.0002**	0.002±0.001
	✓	GCN	0.002±0.001	**0.0003±0.0001**	0.002±0.00007	**0.002±0.0002**	0.009±0.001	**0.0005±0.00004**	0.001±0.0005	**0.001±0.0004**
	✓	GIN	0.004±0.001	0.0006±0.00008	**0.001±0.0001**	0.002±0.0005	**0.008±0.002**	0.0006±0.0001	0.002±0.0008	0.002±0.0007
	✗	GEN	0.181±0.023	0.006±0.0003	0.006±0.001	0.011±0.004	0.309±0.025	0.004±0.0002	0.006±0.001	0.003±0.001
	✗	GCN	0.207±0.006	0.004±0.001	0.002±0.001	0.006±0.0003	0.267±0.049	0.003±0.0004	0.004±0.001	0.002±0.0003
	✗	GIN	0.211±0.007	0.003±0.0002	0.003±0.001	0.008±0.002	0.236±0.014	0.003±0.0004	0.004±0.002	0.003±0.0002

Numerical Results: ODE Baseline

Table: Comparing between IPM-MPNNs and the ODE method on 1000 mini-sized instances. We report the average relative objective gap, constraint violation, training time over three runs, and maximal GPU memory allocated. We print the best results per target in bold.

	Method	MPNN	Setcover	Indset	Cauc	Fac
Obj. gap [%]	ODE	GEN	14.915±0.425	6.225±0.097	13.845±0.554	20.560±0.059
		GCN	14.545±0.055	6.148±0.071	12.945±0.385	20.690±0.037
		GIN	15.050±0.228	6.474±0.114	13.470±1.145	21.010±0.529
	Ours	GEN	2.555±0.122	1.580±0.095	**2.733±0.074**	1.449±0.255
		GCN	**2.375±0.062**	1.447±0.152	2.769±0.091	1.478±0.154
		GIN	2.740±0.3184	**1.404±0.153**	2.847±0.091	**1.328±0.201**
Constraint vio.	ODE	GEN	0.072±0.006	0.046±0.002	0.025±0.008	0.020±0.001
		GCN	0.049±0.012	0.048±0.008	0.025±0.0002	0.020±0.0005
		GIN	0.064±0.005	0.043±0.008	0.024±0.005	0.014±0.004
	Ours	GEN	**0.023±0.002**	0.005±0.0001	0.015±0.003	0.013±0.003
		GCN	0.030±0.003	0.005±0.0006	0.017±0.002	**0.005±0.0006**
		GIN	0.023±0.005	**0.005±0.0003**	**0.014±0.001**	0.006±0.0006
Time [s]	ODE	GEN	47.829	51.283	63.068	96.298
		GCN	57.196	80.133	79.606	34.297
		GIN	55.918	64.628	39.904	62.448
	Ours	GEN	10.177	9.617	9.946	11.124
		GCN	18.964	8.688	**7.368**	**8.834**
		GIN	**6.042**	**8.096**	8.881	10.771
Memory (GB)	ODE	GEN	16.455	25.931	23.354	44.520
		GCN	16.489	34.003	23.805	10.640
		GIN	18.238	30.101	13.482	24.713
	Ours	GEN	**0.091**	0.088	0.101	0.148
		GCN	0.201	0.134	**0.069**	**0.142**
		GIN	0.094	**0.073**	0.148	0.187

Numerical Results: Size generalization

Table: Size generalization. We report the relative objective gap and constraint violation on larger test instances. Numbers represent mean and standard deviation across multiple pretrained models.

	Train size		Inference size		GEN		GCN		GIN	
	Rows	Cols	Rows	Cols	Obj. (%)	Cons.	Obj. (%)	Cons.	Obj. (%)	Cons.
Setc.	[300, 500]	[500, 700]	500	700	0.717±0.158	0.516±0.010	0.511±0.047	0.509±0.004	1.034±0.237	0.486±0.023
			550	750	0.917±0.317	0.552±0.012	0.871±0.252	0.543±0.003	2.318±1.411	0.497±0.032
			600	700	0.993±0.211	0.573±0.015	0.705±0.125	0.565±0.012	1.491±0.512	0.521±0.045
			500	800	0.902±0.323	0.528±0.008	1.058±0.441	0.509±0.004	12.538±16.027	0.485±0.050
			600	800	1.004±0.407	0.589±0.014	1.556±0.588	0.568±0.005	12.217±14.715	0.486±0.071
Indset.	[584, 990]	[300, 500]	[978, 994]	500	0.128±0.027	0.299±0.001	0.099±0.008	0.303±0.001	0.129±0.031	0.304±0.001
			[1028, 1044]	525	0.157±0.063	0.300±0.001	0.101±0.013	0.304±0.001	0.111±0.017	0.305±0.001
			[1076, 1094]	550	0.300±0.186	0.301±0.002	0.096±0.022	0.303±0.001	0.177±0.097	0.304±0.001
			[1128, 1144]	575	1.402±1.036	0.305±0.006	0.146±0.044	0.304±0.001	0.380±0.367	0.304±0.002
			[1178, 1194]	600	4.552±3.153	0.317±0.015	0.408±0.317	0.304±0.001	0.647±0.725	0.304±0.002
Cauc.	[320, 562]	[300, 499]	[530, 564]	500	0.333±0.134	0.257±0.001	0.318±0.048	0.259±0.001	0.344±0.108	0.259±0.001
			[596, 646]	500	0.363±0.131	0.267±0.002	0.519±0.069	0.270±0.003	0.576±0.165	0.271±0.001
			[652, 720]	500	0.524±0.039	0.284±0.001	1.255±0.523	0.289±0.007	0.944±0.114	0.289±0.001
			[559, 596]	600	7.325±3.615	0.257±0.002	0.587±0.268	0.255±0.001	1.014±0.845	0.263±0.006
			[633, 677]	600	7.965±3.941	0.263±0.002	0.868±0.441	0.258±0.003	1.375±0.693	0.269±0.005
Fac.	[441, 900]	[420, 870]	961	930	0.912±0.251	0.178±0.006	1.154±0.206	0.173±0.007	1.452±0.528	0.178±0.003
			936	900	1.320±0.347	0.148±0.009	1.615±0.322	0.145±0.009	1.736±0.558	0.153±0.004
			936	910	0.964±0.063	0.209±0.005	1.538±0.526	0.211±0.007	1.538±0.422	0.215±0.006
			1116	1080	1.502±0.704	0.163±0.009	3.540±3.134	0.161±0.006	2.288±0.659	0.167±0.005
			1296	1260	1.808±0.566	0.173±0.009	7.629±7.577	0.179±0.008	13.522±8.027	0.163±0.021

Numerical Results: Inference Profiling

Table: Comparing IPM-MPNNs' inference time to SciPy's IPM implementation and our Python-based IPM solver. We report mean and standard deviation in seconds over three runs. We print the best results per target in bold.

Instances	SciPy Solver	Our Solver	GEN	GCN	GIN
Small setcover	**0.006**$\pm$**0.004**	0.071$\pm$0.015	0.033$\pm$0.001	0.029$\pm$0.001	0.017$\pm$0.001
Large setcover	0.390$\pm$0.098	3.696$\pm$2.141	0.033$\pm$0.001	0.030$\pm$0.001	**0.021**$\pm$**0.001**
Small indset	**0.008**$\pm$**0.067**	0.089$\pm$0.024	0.033$\pm$0.001	0.031$\pm$0.002	0.021$\pm$0.001
Large indset	0.226$\pm$0.087	1.053$\pm$0.281	0.033$\pm$0.002	0.030$\pm$0.001	**0.021**$\pm$**0.001**
Small cauc	**0.012**$\pm$**0.005**	0.151$\pm$0.035	0.033$\pm$0.001	0.028$\pm$0.001	0.021$\pm$0.001
Large cauc	0.282$\pm$0.065	3.148$\pm$0.880	0.033$\pm$0.001	0.029$\pm$0.001	**0.021**$\pm$**0.001**
Small fac	**0.017**$\pm$**0.011**	2.025$\pm$1.854	0.029$\pm$0.001	0.029$\pm$0.001	0.022$\pm$0.001
Large fac	0.732$\pm$0.324	6.229$\pm$2.672	0.030$\pm$0.001	0.031$\pm$0.001	**0.022**$\pm$**0.001**

Outline

PDHG for Linear Programming

Consider the LP problem $\mathcal{M} = (G; l, u, c; h)$ in standard form

$$\begin{aligned} \min\ & c^\top x \\ \text{s.t. } & Gx \geq h \\ & l \leq x \leq u \end{aligned}$$

where $G \in \mathbb{R}^{m \times n}, h \in \mathbb{R}^m, c \in \mathbb{R}^n, l \in (\mathbb{R} \cup \{-\infty\})^n, u \in (\mathbb{R} \cup \{+\infty\})^n$

Saddle point problem form: $\min_{l \leq x \leq u} \max_{y \geq 0} L(x, y; \mathcal{M}) = c^\top x - y^\top G x + h^\top y$

Primal-Dual Hybrid Gradient (PDHG)

– Initialize $x^0 \in \mathbb{R}^n,\ y^0 \in \mathbb{R}^m$

For $k = 0, 1, 2, ..., K-1$

$$\left\lfloor \begin{aligned} x^{k+1} &= \mathbf{Proj}_{l \leq x \leq u}(x^k - \tau(c - G^\top y^k)); \\ y^{k+1} &= \mathbf{Proj}_{y \geq 0}(y^k + \sigma(h - 2Gx^{k+1} + Gx^k)). \end{aligned} \right.$$

Unrolling PDHG into PDHG-Net

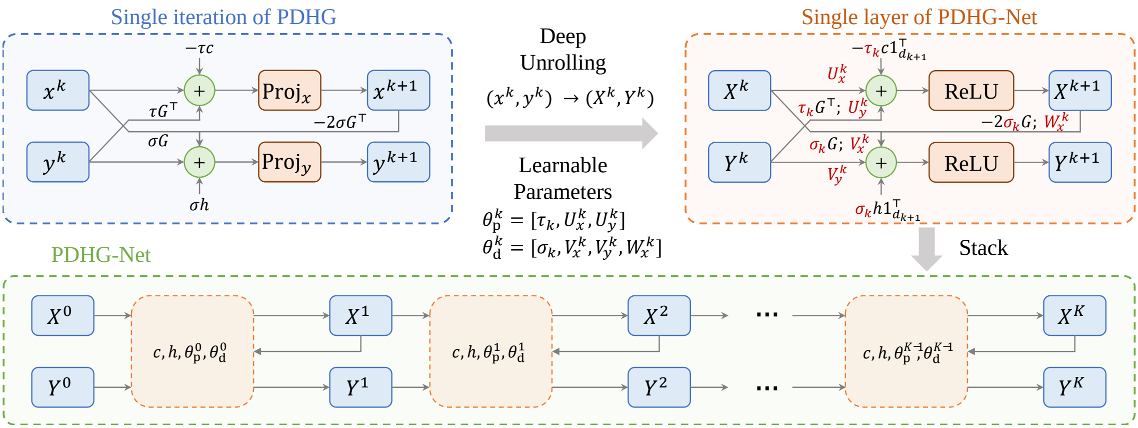


Figure: Overview of how each layer in PDHG-Net corresponds to each iteration of the traditional PDHG algorithm, along with the overall architecture of PDHG-Net.

Key Technique: Channel Expansion

- Expanding the n-dimensional vectors x^k, y^k into $(n \times d_k)$-dimensional matrices X^k, Y^k with d_k columns (or called channels following the convention of neural network)

- The linear combination $x^k - \tau(c - G^\top y^k)$ of primal-dual is replaced by

$$X^k U_x^k - \tau_k(c \cdot \mathbf{1}_{d_{k+1}}^\top - G^\top Y^k U_y^k)$$

  where $\Theta_{\mathrm{p}}^k = (\tau_k, U_x^k, U_y^k) \in \mathbb{R} \times \mathbb{R}^{d_k \times d_{k+1}} \times \mathbb{R}^{d_k \times d_{k+1}}$ is the trainable parameter of the k-th primal NN block

- **Generalizability to LP instances of different sizes**: Following the principle of classical unrolling, a natural idea would be to unroll $x^k - \tau(c - G^\top y^k)$ to

$$x^k - \tau(c - W^k y^k)$$

  where W^k is trainable matrix. This is **unsuitable** for applying to LP problems with different sizes

Architecture of PDHG-Net

$$
\begin{aligned}
&- \text{Initialize } X^0 = [x^0, l, u, c],\ \ Y^0 = [y^0, h] \\
&\text{For } \ k = 0, 1, 2, ..., K-1 \\
&\quad \left| \begin{aligned}
X^{k+1} &= \mathrm{ReLU}\left(X^k U_x^k - \tau_k (c \cdot \mathbf{1}_{d_{k+1}}^\top - G^\top Y^k U_y^k)\right), \\
Y^{k+1} &= \mathrm{ReLU}\left(Y^k V_y^k \right. \\
&\qquad \left. + \sigma_k (h \cdot \mathbf{1}_{d_{k+1}}^\top - 2 G X^{k+1} W_x^k + G X^k V_x^k)\right),
\end{aligned} \right. \\
&- \text{Output } X^K \in \mathbb{R}^n,\ \ Y^K \in \mathbb{R}^m
\end{aligned}
$$

The trainable parameter is $\Theta = \{\Theta_{\mathrm{p}}^k, \Theta_{\mathrm{d}}^k\}_{k=0}^{K-1}$, where

$$
\Theta_{\mathrm{p}}^k = (\tau_k, U_x^k, U_y^k) \in \mathbb{R} \times \mathbb{R}^{d_k \times d_{k+1}} \times \mathbb{R}^{d_k \times d_{k+1}}
$$

$$
\Theta_{\mathrm{d}}^k = (\sigma_k, V_x^k, V_y^k, W_x^k) \in \mathbb{R} \times \mathbb{R}^{d_k \times d_{k+1}} \times \mathbb{R}^{d_k \times d_{k+1}} \times \mathbb{R}^{d_{k+1} \times d_{k+1}}
$$

Convergence Property of PDHG

Theorem

Let $(x^k, y^k)_{k\geq 0}$ be the primal-dual variables generated by the PDHG algorithm for the LP problem $\mathcal{M} = (G; l, u, c; h)$. If the step sizes τ, σ satisfy $\tau\sigma\|G\|_2^2 < 1$, then for any $(x, y) \in \mathbb{R}^n \times \mathbb{R}^m_{\geq 0}$ satisfying $l \leq x \leq u$, the primal-dual gap satisfies

$$
\begin{aligned}
&L(\bar{x}^k, y; \mathcal{M}) - L(x, \bar{y}^k; \mathcal{M}) \\
&\leq \frac{1}{2k}\left(\frac{\|x - x^0\|^2}{\tau} + \frac{\|y - y^0\|^2}{\sigma} - (y - y^0)^\top G(x - x^0)\right),
\end{aligned}
$$

where $\bar{x}^k = (\sum_{j=1}^k x^j)/k$, $\bar{y}^k = (\sum_{j=1}^k y^j)/k$, and L is the Lagrangian defined by LP.

Alignment Theorem: PDHG is a Specific PDHG-Net

Theorem

Given any pre-determined network depth K and the widths $\{d_k\}_{k \leq K-1}$ with $d_k \geq 10$, there exists a K-layer PDHG-Net with its parameter assignment Θ_{PDHG} satisfying the following property: given any LP problem $\mathcal{M} = (G; l, u, c; h)$ and its corresponding primal-dual sequence $(x^k, y^k)_{k \leq K}$ generated by PDHG algorithm within K iterations, we have

1. *For any hidden layer k, both $\bar{x}^k$ and x^k can be represented by a linear combination of X^k's channels, both $\bar{y}^k$ and y^k can be represented by a linear combination of Y^k's channels. Importantly, these linear combinations do not rely on the LP problem $\mathcal{M}$.*
2. *PDHG-Net's output embeddings $X^K \in \mathbb{R}^{n \times 1}$ and $Y^K \in \mathbb{R}^{m \times 1}$ are equal to the outputs $\bar{x}^K$ and $\bar{y}^K$ of the PDHG algorithm, respectively.*

Estimate the Approximation Efficiency

Theorem

Given the approximation error bound ϵ, there exists a PDHG-Net with $\mathcal{O}(1/\epsilon)$ number of neurons and the parameter assignment Θ_{PDHG} fulfilling the following property. For any LP problem $\mathcal{M} = (G; l, u, c; h)$ and $(x, y) \in \mathbb{R}^n \times \mathbb{R}^m_{\geq 0}$ satisfying $l \leq x \leq u$, it holds that

$$L(X^K, y; \mathcal{M}) - L(x, Y^K; \mathcal{M}) < \epsilon.$$

- The proof is rather concise to compare with Bipartite representation theorem
- Give an explicit estimation of the number of neurons required to represent a solution

Numerical Result

- **Training dataset:** A set of instances denoted by $\mathcal{I} = \{(\mathcal{M}, z^*)\}$. The input of an instance is an LP problem $\mathcal{M} = (G; l, u, c; h)$; the label $z^* = (x^*, y^*)$ is the solution
- **Loss Function:** We train the PDHG-Net to minimize the ℓ_2 square loss

$$\min_{\Theta} \mathcal{L}_{\mathcal{I}}(\Theta) = \frac{1}{|\mathcal{I}|} \sum_{(\mathcal{M}, z^*) \in \mathcal{I}} \left\| Z^K(\mathcal{M}; \Theta) - z^* \right\|_2^2$$

- **Metric:** We calculate the improvement ratio over PDLP using the following equation:

$$\text{Improv.} = \frac{\text{PDLP} - \text{ours}}{\text{PDLP}},$$

where this metric is applicable to both the solving time and the number of iterations

Overview of the Datasets

# nodes	# vars.	# cons.	# nnz.
10^3	1,000	1,001	7,982
10^4	10,000	10,001	79,982
5×10^4	50,000	50,001	399,982
10^5	100,000	100,001	799,982
10^6	1,000,000	1,000,001	7,999,982

Table: Sizes of utilized PageRank instances.

dataset	# vars.	# cons.	# nnz.
`IP-S`	31,350	15,525	5,291,250
`IP-L`	266,450	91,575	94,826,250
`WA-S`	80,800	98,830	3,488,784
`WA-L`	442,000	541,058	45,898,828

Table: Sizes of utilized instances.

Two-stage Algorithm: PDHG-Net as Warm-start

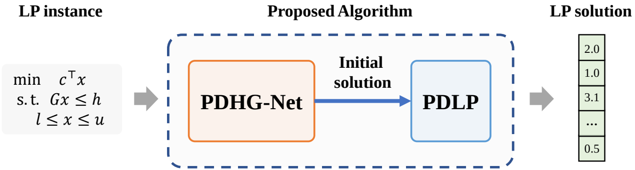


Figure: The proposed post-processing procedure warm-starts the PDLP solver using the prediction of PDHG-Net as initial solutions to ensure optimality.

Table: Solve time comparison between the proposed framework and vanilla PDLP on PageRank instances. The improvement ratio of the solving time is also reported.

# nodes	ours	PDLP	Improv.
10^3	0.01sec.	0.04sec.	↑ 45.7%
10^4	0.4 sec.	1.1sec.	↑ 67.6%
10^5	22.4sec.	71.3sec.	↑ 68.6%
10^6	4,508sec.	16,502sec.	↑ 72.7%

Efficiency and the Number of Restarts

Table: Comparison of the proposed framework against default PDLP in solving IP and WA instances.

dataset.	time (sec.)			# iters.		
	ours	PDLP	Improv.	ours	PDLP	Improv.
IP-S	**9.2**	11.4	↑ 19.5%	**422**	525	↑ 19.5%
IP-L	**7,866.3**	10,045.6	↑ 21.7%	**6,048**	8,380	↑ 27.8%
WA-S	**114.7**	137.8	↑ 16.7%	**8,262**	9,946	↑ 16.9%
WA-L	**4817.6**	6426.2	↑ 25.0%	**14,259**	17,280	↑ 17.5%

Table: The average number of restarts in the PDLP solving process with our framework (ours) and default settings (represented by PDLP).

# of Nodes		5×10^3	1×10^4	2×10^4	4×10^4
# restarts	Ours	2.2	4.15	2.0	2.0
	PDLP	5.9	11.7	20.25	11.3

Prediction Accuracy v.s. Epochs

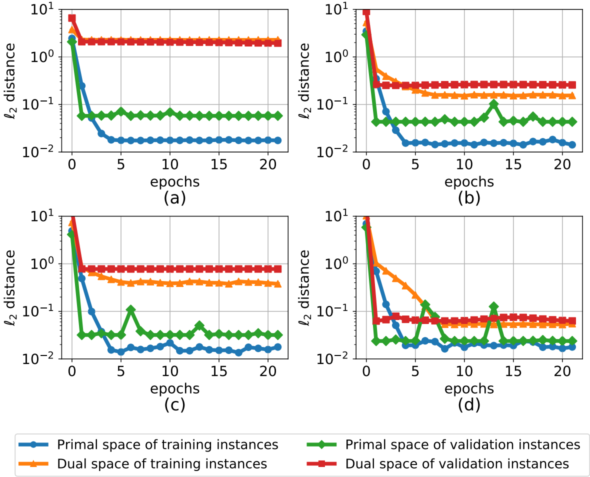
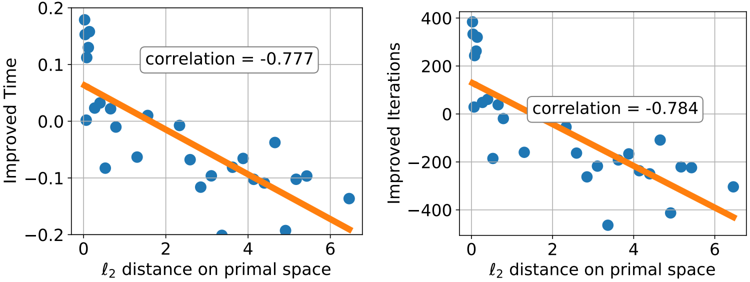


Figure: The distance between the predicted solution of PDHG-Net and optimal solution in PageRank training and validation instances with (a) 5×10^3, (b) 1×10^4, (c) 2×10^4, (d) 4×10^4 variable sizes.

Improvement v.s. Prediction Accuracy



(a) Solving time (b) Number of iterations

Figure: We present the improvement ratio in both solving time and the number of iterations for solutions extrapolated at varying distances from the optimal solution. Each blue dot symbolizes an extrapolated solution, while the yellow line represents the trend line fitted through these points. Results demonstrate a strong correlation.

Generalizability to Larger Sizes

Table: Solving time and number of iterations for PageRank, `IP` and `WA` instances larger than training set sizes. For clarity, we denote the size of the largest instance of `IP` and `WA` datasets as `Large`.

metric	Dataset	size	ours	PDLP	Improv.
time (sec.)	PageRank	5×10^4	**5.5**	11.2	↑ 50.9%
		1×10^5	**17.0**	32.5	↑ 47.8%
	`IP`	`Large`	**6796.7**	8631.4	↑ 21.3%
	`WA`	`Large`	**5599.1**	5859.4	↑ 4.4%
# iter.	PageRank	5×10^4	**1,605**	3,397	↑ 52.7%
		1×10^5	**1,958**	3,914	↑ 50.0%
	`IP`	`Large`	**7,291**	8,970	↑ 18.7%
	`WA`	`Large`	**16,166**	17,280	↑ 6.4%

Portion of GPU Time & Comparison with GNNs

Table: Comparison of total solving time and GPU time for initial solutions, including the ratio of GPU time to total solving time.

# nodes.	10^3	10^4	10^5	10^6
GPU time (sec.)	0.01	0.02	0.21	1.12
CPU time (sec.)	0.02	0.4	22.4	4,508.3
Ratio	52.6%	5.7%	0.9%	0.02%

Table: Comparison of improvement ratio and ℓ_2 distance between the proposed framework implemented with PDHG-Net and GNN.

# nodes.	Improv.		ℓ_2 distance	
	ours	GNN	ours	GNN
10^3	↑ 45.7%	↑ 1.4%	0.05	0.51
10^4	↑ 67.6%	↑ 19.3%	0.2	1.38
10^5	↑ 71.3%	↓ 4.0%	0.95	30.35

Outline

Dual Interior-Point Optimization Learning for Linear Programming

- Consider parametric optimization problems of the form

$$
\begin{aligned}
(\mathrm{P}_\beta) \quad \min_{x} \quad & c_\beta^\top x \\
\text{s.t.} \quad & A_\beta x = b_\beta \\
& l_\beta \leq x \leq u_\beta
\end{aligned}
$$

- *Dual Optimization Proxy*: a model that returns a *dual-feasible solution* to P_β

- The dual of the problem is given by

$$
\begin{aligned}
(\mathrm{D}_\beta) \quad \max_{y} \quad & b^\top y + l^\top zl - u^\top zu \\
\text{s.t.} \quad & A^\top y + z^l - z^u = c \\
& z^l, z^u \geq 0
\end{aligned}
$$

where $y \in \mathbb{R}^m$ and $z^l, z^u \in \mathbb{R}^n$

The Lagrangian Functions for Primal and Dual Problems

- The Lagrangian function for P_β is

$$\mathcal{L}_s(\mathbf{y}) = \min_{\mathbf{l} \le \mathbf{x} \le \mathbf{u}} \mathbf{c}^\top \mathbf{x} + \mathbf{y}^\top (\mathbf{b} - \mathbf{A}^\top \mathbf{x})$$

- Set $\varphi_\mu(\mathbf{x}) = -\mu \sum_j \ln(\mathbf{x}_j - \mathbf{l}_j) + \ln(\mathbf{u}_j - \mathbf{x}_l)$. The Lagrangian function for D_β is

$$\mathcal{L}_\mu(\mathbf{y}) = \min_{\mathbf{l} \le \mathbf{x} \le \mathbf{u}} \mathbf{c}^\top \mathbf{x} + \mathbf{y}^\top (\mathbf{b} - \mathbf{A}^\top \mathbf{x}) + \varphi_\mu(\mathbf{z})$$

Dual Interior Point Learning (DIPL) & Dual Supergradient Learning (DSL)

The training of the proxy model $\mathcal{M}_\theta$ for DSL and DIPL can be formalized as

$$\max_\theta \mathbb{E}_\beta \left[\mathcal{L}\left(\mathcal{M}_\theta\left(A_\beta, \mathbf{b}_\beta, \mathbf{c}_\beta, \mathbf{l}_\beta, \mathbf{u}_\beta\right)\right)\right]$$

where the expectation is taken over a distribution of problem data parameterized by β. The loss function $\mathcal{L}$ is set to $\mathcal{L}_s$ for DSL and $\mathcal{L}_\mu$ for DIPL.

Algorithm DSL and DIPL Training

Input: Dataset $\{A_i, b_i, c_i, l_i, u_i\}_{i=1}^N$, learning rate α, epoch count E, loss function $\mathcal{L}$
Output: Dual proxy model $\mathcal{M}_\theta$

1: Initialize θ randomly $e = 1, \ldots, E$ each i
2: $y_i \leftarrow \mathcal{M}_\theta(A_i, b_i, c_i, l_i, u_i)$
3: $\ell \leftarrow \frac{1}{N} \sum_i \mathcal{L}(y_i)$
4: $\theta \leftarrow \theta + \alpha \nabla_\theta \ell$
5: **return** $\mathcal{M}_\theta$

Experimental Results

- The implementation of DSL is denoted by $\mathcal{S}$ and the implementation of DIPL by $\mathcal{I}_\mu$
- $\mathcal{M}_\theta$: A 3-layer fully-connected neural network with ReLU activations
- 10,000 feasible samples are used for training and 2,500 are used for validation. The testing set is always the same set of 5,000 feasible samples

Table: DCOPF Benchmark Summary. For each benchmark, we report the number of constraints m, the number of variables n, the hidden layer size h, and the total number of parameters $|\theta|$ in the neural network $\mathcal{M}_\theta$

Benchmark	m	n	h	$\lvert\theta\rvert$
1354_pegase	1992	2251	2048	16.6M
2869_pegase	4583	5092	4096	71.1M
6470_rte	9006	9766	8192	281M

The parametric DCOPF problem is of the form

$$\begin{aligned} \mathrm{OPF}(\mathbf{p}^{\mathrm{d}}) \quad \min_{\mathbf{p}^{\mathrm{g}},\mathbf{p}^{\mathrm{f}}} \quad & \mathbf{c}^{\top}\mathbf{p}^{\mathrm{g}} \\ s.t. \quad & \mathbf{e}^{\top}\mathbf{p}^{\mathrm{g}} = \mathbf{e}^{\top}\mathbf{p}^{\mathrm{d}} \\ & \mathbf{p}^{\mathrm{f}} = \mathrm{PTDF}(\mathbf{p}^{\mathrm{g}} - \mathbf{p}^{\mathrm{d}}) \\ & \underline{\mathbf{p}}^{\mathrm{g}} \leq \mathbf{p}^{\mathrm{g}} \leq \overline{\mathbf{p}}^{\mathrm{g}} \\ & \underline{\mathbf{p}}^{\mathrm{f}} \leq \mathbf{p}^{\mathrm{f}} \leq \overline{\mathbf{p}}^{\mathrm{f}} \end{aligned}$$

The DCOPF instances are first converted to the normal form (only **b** varies)

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{c}^{\top}\mathbf{x} \\ s.t. \quad & \mathbf{A}\mathbf{x} = \mathbf{b}^{\beta} \\ & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \end{aligned}$$

Dual Gap Ratio v.s. Epochs

The dual gap ratio is reported as a percentage:

$$\text{Dual Gap Ratio} = \frac{\mathcal{L}(y^*) - \mathcal{L}(\dot{y})}{\mathcal{L}(y^*)} \times 100\%$$

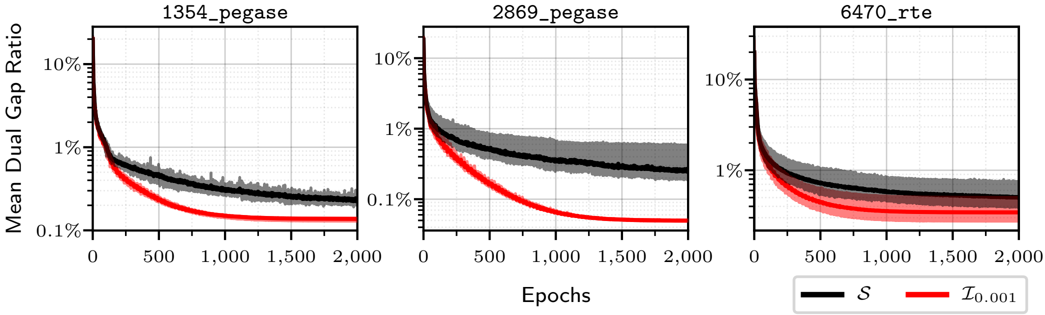
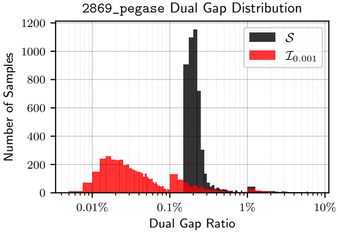
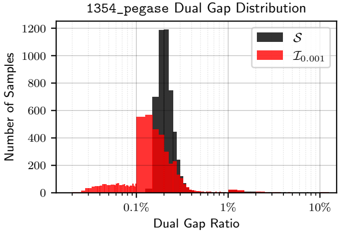


Figure: Convergence of the mean dual gap ratio on testing set samples during training.

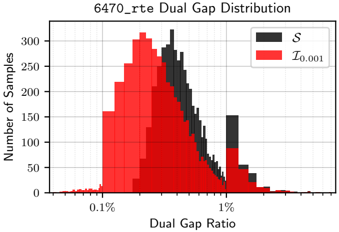
Distribution of the Dual Gap Ratio



(a) 2869_pegase



(b) 1354_pegase



(c) 6470_rte

Figure: Distribution of dual gap ratios over testing set samples for different benchmarks.

Many Thanks For Your Attention!