TOWARDS EXPLAINING THE POWER OF CONSTANT DEPTH GRAPH NEURAL NETWORKS FOR STRUCTURED LINEAR PROGRAMMING

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph neural networks (GNNs) have recently emerged as powerful tools for solving complex optimization problems, often being employed to approximate solution mappings. Empirical evidence shows that even *shallow* GNNs (with fewer than ten layers) can achieve strong performance in predicting optimal solutions to linear programming (LP) problems. This finding is somewhat counter-intuitive, as LPs are *global* optimization problems, while shallow GNNs predict based on *local* information. Although previous theoretical results suggest that GNNs have the expressive power to solve LPs, they require deep architectures whose depth grows at least polynomially with the problem size, and thus leave the underlying principle of this empirical phenomenon still unclear. In this paper, we examine this phenomenon through the lens of distributed computing and average-case analysis. We establish that the expressive power of GNNs for LPs is closely related to well-studied distributed algorithms for LPs. Specifically, we show that any d-round distributed LP algorithm can be simulated by a d-depth GNN, and vice versa. In particular, by designing a new distributed LP algorithm and then unrolling it, we prove that constant-depth, constant-width GNNs suffice to solve sparse binary LPs effectively. Here, in contrast with previous analyses focusing on *worst-case* scenarios, in which we show that GNN depth must increase with problem size by leveraging an impossibility result about distributed LP algorithms, our analysis shifts the focus to the *average-case* performance, and shows that constant GNN depth then becomes sufficient no matter how large the problem size is. Our theory is validated by numerical results.

1 INTRODUCTION

Learning to Optimize (L2O) has recently gained considerable attention as a promising framework that leverages machine learning techniques to discover efficient optimization strategies. Compared to traditional rule-based methods, L2O methods adaptively learn optimization strategies tailored to specific problem classes, offering greater flexibility and efficiency. The L20 framework has shown its potential in handling both continuous optimization (Monga et al., 2021) and combinatorial optimization (Bengio et al., 2021; Mazyavkina et al., 2021), and opens up new possibilities for automating and improving the efficiency of various real-world applications.

Graph Neural Networks (GNNs) have emerged as a powerful tool within the L2O framework (Cappart et al., 2023; Peng et al., 2021). GNNs are dedicated to capturing dependencies and relational structures among variables, thus well-suited for representing and solving optimization problems with graph-structured data, e.g., max cut (Schuetz et al., 2022), traveling salesman problem (Hudson et al., 2021), and minimum vertex covering (Sato et al., 2019), where the relationships between variables are key to finding efficient solutions.

In particular, there is a line of research (Li et al., 2022; Chen et al., 2023; Kuang et al., 2023; Fan et al., 2023; Liu et al., 2024; Qian et al., 2024; Li et al., 2024a) exploring the power of GNNs to accelerate the solving process of linear programming (LP). These studies are based on the observation that a general LP instance $\max\{\boldsymbol{c}^T \cdot \boldsymbol{x} \mid \boldsymbol{A} \cdot \boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq 0\}$ can be naturally represented as a labeled bipartite graph. Specifically, a variable x_j corresponds to a left node u_j labeled with c_j, a constraint

$\boldsymbol{A}_{i,:} \cdot \boldsymbol{x} \leq b_i$ (or a dual variable y_i) corresponds to a right node v_i labeled with b_i, and a left node u_j and a right node v_i are connected by an edge labeled with A_{ij} if $A_{ij} \neq 0$. Empirical results show that even very *shallow* (with fewer than ten layers) GNNs often perform well in predicting optimal solutions to LPs. Remarkably, Li et al. (2024a) demonstrated that by employing a 4-layer GNN as a warm-start, the PDLP solver can achieve a $3\times$ speedup on the PageRank problem with millions of nodes compared to the vanilla version.

We note that this empirical phenomenon is somewhat counter-intuitive. In a shallow GNN, no matter how wide it is, each node has to base its output on *local* information from its d-hop neighborhood (d denotes the number of layers of GNN, a.k.a. the depth of GNN), making the GNN inherently limited in capturing long-range dependencies. In contrast, LP is a global optimization problem, where the optimal value of one variable usually depends on the global information and long-range dependencies between variables often exist (see (Trevisan & Xhafa, 1998) for an example). Thus besides the potential guidance to L2O practitioners, it also has its own theoretical interest to understand the principles behind the empirical phenomenon.

Previous theoretical works (Chen et al., 2023; Qian et al., 2024; Li et al., 2024a) have proven that *anonymous* GNNs possess enough expressive power to solve LPs. Here, we say a GNN anonymous if the nodes are not equipped with (unique) identifiers. In particular, Li et al. (2024a) showed that the popular Primal-Dual Hybrid Gradient (PDHG) algorithm can be naturally aligned with GNNs, where the right node v_i in the bipartite graph represents a dual variable instead of a constraint. This alignment explains why GNNs are often used as the basic network architecture in L2O from a primal-dual perspective. Unfortunately, all these theoretical results require deep GNNs whose depth grows at least polynomially with the instance size. The reason behind the empirical success of shallow GNNs for LPs remains poorly understood.

1.1 Our contributions

In this paper, we examine this phenomenon through the lens of distributed computing and average-case analysis; and prove that constant-depth, constant-width GNNs suffice to solve sparse binary LPs (Theorem 3), where the depth and the width of GNN remain the same as the problem size grows.

Sparse Binary LPs. An instance of binary LP has all the entries of the constraint matrix $\boldsymbol{A}$ as binary, and $\boldsymbol{b}$ and $\boldsymbol{c}$ both as all-ones vectors. A binary LP instance, therefore, takes the following form:

$$\begin{aligned} \max_{\boldsymbol{x}} \quad & \sum_{j=1}^{n} x_j \qquad \text{(Primal binary LP)} \\ \text{s.t.} \quad & \sum_{j=1}^{n} A_{ij} x_j \leq 1, \forall i \in [m] \\ & \boldsymbol{x} \geq 0 \end{aligned} \qquad\qquad \begin{aligned} \min_{\boldsymbol{y}} \quad & \sum_{i=1}^{m} y_i \qquad \text{(Dual binary LP)} \\ \text{s.t.} \quad & \sum_{i=1}^{m} A_{ij} y_i \geq 1, \forall j \in [n] \\ & \boldsymbol{y} \geq 0 \end{aligned}$$

where each $A_{ij} \in \{0, 1\}$. Here, $\boldsymbol{A}$ has dimensions $m \times n$, and we use $[n]$ to denote $\{1, 2, \ldots, n\}$. An LP instance is called *sparse* if $\boldsymbol{A}$ has at most $O(m+n)$ non-zero entries. This paper particularly focuses on LP instances where $m = \Theta(n)$. We note that such sparse binary LPs can model the fractional versions of many basic combinatorial optimization problems, such as minimum vertex cover, maximum matching, and minimum dominating set on sparse graphs.

Our Ideas. Our result builds on the following two observations.

- Observation 1: The use of GNNs for LPs is closely related to the well-studied *distributed algorithms* for LPs. Specifically, we find that any d-round distributed LP algorithm can be simulated by a d-depth GNN, and vice versa.
- Observation 2: In the aforementioned empirical phenomenon, the performance of GNNs is measured based on the *average* instance rather than the *worst* instance. In other words, GNNs are required to work for "most" instances (i.e., they are allowed to fail in a few exceptional cases), rather than for every possible instance.

Remark 1. *We note that Observation 2 is general in L2O scenarios, for the following reasons: (i) most L2O methods target NP-hard problems, such as mixed integer linear programs or traveling salesman problem, where it is unrealistic to expect the L2O method works for every possible instance; (ii) the general objection of L2O methods is solve instances that arise "in practice", where the worst instance may never be encountered, and people are usually satisfied if L2O methods work for "many" or "most" instances.*

Remark 2. *This paper will consider a chain of reduction in generality: from general LPs to packing/covering LPs (a.k.a. non-negative LP), then to sparse binary LPs, and finally to row-sparse column-sparse binary LPs. While the focus is primarily on structured LPs, the two observations mentioned above are applicable to general LPs as well.*

A central topic in distributed computing is to determine what global goals can or cannot be achieved based on local information. Based on Observation 1, we can leverage the known principles of distributed LP algorithms to study the power of GNNs. First, by unrolling a distributed LP algorithm proposed by Li et al. (2024b), we show that there exists a constant-depth, constant-width GNN that can approximately solve row-sparse, column-sparse binary LPs (Theorem 2). A row-sparse, column-sparse binary LP is a special kind of sparse binary LP where the number of non-zero entries in each row and each column of $\boldsymbol{A}$ is upper bounded by a constant. Moreover, by leveraging an impossibility result about distributed LP algorithms (Kuhn et al., 2016), we conclude that there exist no constant-depth GNNs that can solve every possible instance of sparse binary LP (Lemma 1). However, if we shift the focus to the average-case performance, by designing and unrolling a new distributed algorithm, we prove that constant GNN depth then becomes sufficient for *almost all* instances of sparse binary LP (Theorem 3), which is the main technical part of this paper. Finally, we conduct experiments that directly validate the theoretical result.

2 Distributed LP algorithms and their connections with GNNs

In this section, we first briefly review known results about distributed LP algorithms, where we will provide the description of a distributed algorithm that will be used. Then we demonstrate the connections with GNNs.

2.1 Distributed algorithms for LPs

The Distributed Computational Model. In the distributed computational model, there is a network $G = (V, E)$, where (i) each node represents a processor and (ii) each edge (u, v) represents a bidirectional communication channel connecting processors u and v. The computation proceeds in rounds. In one round, each processor first executes local computations and then sends messages to its neighbors. Each of the messages is allowed to be arbitrarily long (so we are talking about the so-called LOCAL model). The algorithm complexity is measured in the number of rounds it performs. Note that in a d-round distributed algorithm, each node has to base its output on local information from its d-hop neighborhood. A central topic in distributed computing is to determine what global goals can or cannot be achieved based on local information (Peleg, 2000).

For LP problems, most of the related works (see e.g. (Kuhn et al., 2006)) considered the following distributed setting. Given an LP instance $\max\{\boldsymbol{c}^T \cdot \boldsymbol{x} \mid \boldsymbol{A} \cdot \boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq 0\}$,

- The network is bound to a bipartite graph $G = (U, V, E)$: each primal variable x_j is associated with a left node u_j; each dual variable y_i is associated with a right node v_i; and an edge (u_j, v_i) exists if and only if $A_{ij} > 0$. Note that this bipartite graph is exactly the one used by GNNs to encode LPs. Besides, we will use N_i to denote the set of neighbors of u_i.
- At the beginning of an algorithm, each left node u_j only knows c_j and the j-th column $\boldsymbol{A}_{:,j} = (A_{1j}, A_{2j}, \cdots)^T$, and each right node v_i only knows b_i and the i-th row $\boldsymbol{A}_{i,:} = (A_{i1}, A_{i2}, \cdots)$.
- At the end of an algorithm, each left node u_j is required to output a $\hat{x}_j$ and each right node v_i to output a $\hat{y}_i$, which together are supposed to form an approximate solution to the primal LP $\max\{\boldsymbol{c}^T \cdot \boldsymbol{x} \mid \boldsymbol{A} \cdot \boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq 0\}$ and the dual LP $\min\{\boldsymbol{b}^T \cdot \boldsymbol{y} \mid \boldsymbol{A}^T \cdot \boldsymbol{y} \geq \boldsymbol{c}, \boldsymbol{y} \geq 0\}$ respectively. Here, we say $\boldsymbol{x}$ is a $(1+\epsilon)$-approximate solution if (i) it is a feasible solution and (ii) the ratio between its objective value and optimum lies in $[1/(1+\epsilon), 1+\epsilon]$.

Distributed Algorithms for LPs. An important class of distributed LP algorithms are based on first-order methods (FOM) (Lu, 2024), which only utilize gradient information to update their iterates and thus avoid matrix factorization. The main FOM-based LP algorithms include PDHG Chambolle & Pock (2016), PDLP (Applegate et al., 2021), ABIP (Lin et al., 2021; Deng et al., 2022), ECLIPSE (Basu et al., 2020), and SCS (O'donoghue et al., 2016; O'Donoghue, 2021). Applegate et al. (2023) showed that the Restarted PDHG can find an solution that is ε-close to the optimal solution with $O(\|A\|_2 \cdot \alpha^{-1} \log(1/\varepsilon))$ iterations, and matches with the complexity lower bound of span-respecting FOMs. Here, α is the Hoffman's constant of the KKT system.

There is another line of research (Papadimitriou & Yannakakis, 1993; Bartal et al., 1997; Kuhn & Wattenhofer, 2005; Kuhn et al., 2006; Awerbuch & Khandekar, 2009; Floréen et al., 2008; 2011; Kuhn et al., 2016; Ahmadi et al., 2018; Li et al., 2024b) studying distributed algorithms for solving packing LPs and the dual covering LPs. A packing LP and its dual covering LP are non-negative LPs of the canonical form: $\max\{\boldsymbol{c}^T \cdot \boldsymbol{x} \mid \boldsymbol{A} \cdot \boldsymbol{x} \le \boldsymbol{b}, \boldsymbol{x} \ge 0\}$ and $\min\{\boldsymbol{b}^T \cdot \boldsymbol{y} \mid \boldsymbol{A}^T \cdot \boldsymbol{y} \ge \boldsymbol{c}, \boldsymbol{y} \ge 0\}$ where $\boldsymbol{A}, \boldsymbol{b}$ and $\boldsymbol{c}$ have only non-negative entries. Packing/covering LPs can be normalized into the following forms (Awerbuch & Khandekar, 2009; Li et al., 2024b):

$$\begin{aligned} \max_{\boldsymbol{x}} \quad & \mathbf{1}^T \boldsymbol{x} && \text{(Normalized packing LP)} \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} \le \mathbf{1} \\ & \boldsymbol{x} \ge 0 \end{aligned} \qquad \begin{aligned} \min_{\boldsymbol{y}} \quad & \mathbf{1}^T \boldsymbol{y} && \text{(Normalized covering LP)} \\ \text{s.t.} \quad & \boldsymbol{A}^T \boldsymbol{y} \ge \mathbf{1} \\ & \boldsymbol{y} \ge 0 \end{aligned}$$

where A_{ij} is either ≥ 1 or 0.

Let n and m denote the number of primal variables and dual variables respectively, i.e., $\boldsymbol{A}$ has dimensions $m \times n$, and let $A_{\max}$ denote $\max_{i,j} A_{ij}$. Define

$$\Gamma_p := \max_j \sum_{i=1}^{m} A_{ij}, \text{ and } \Gamma_d := \max_i \sum_{j=1}^{n} A_{ij}.$$

Bartal et al. (1997) proposed the first constant-factor approximation algorithm running in $\mathrm{polylog}(m+n)$ rounds for packing/covering LPs. This algorithm was further improved by Kuhn et al. (2006): they developed an $(1+\epsilon)$-approximation algorithm running in $O\left(\log \Gamma_p \cdot \log \Gamma_d / \epsilon^4\right)$ rounds. In particular, for row-sparse, column-sparse instances, their algorithm runs in $O\left(\log^2 A_{\max}/\epsilon^4\right)$ rounds. Later, Awerbuch & Khandekar (2009) proposed another $(1+\epsilon)$-approximation algorithm for packing/covering LPs running in $\tilde{O}\left(\log^2(mA_{\max}) \log^2(nmA_{\max})/\epsilon^5\right)$ rounds, which has slightly worse bound than Kuhn et al. (2006) but enjoys the features of simplicity and statelessness. Recently, Li et al. (2024b) proposed a distributed algorithm (Algorithm 1) that returns a $(1+\epsilon)$-approximate solution for row-sparse, column-sparse instances in $O\left(A_{\max} \cdot \log A_{\max}/\epsilon^2\right)$ rounds. We remark that all the aforementioned distributed algorithms are anonymous, where the nodes in the network are not equipped with identifiers.

On the lower bound side, Kuhn et al. (2016) proved that: every constant-factor approximation distributed algorithm for the fractional maximum matching problem, a special kind of sparse binary LP, requires at least $\Omega(\sqrt{\log(m+n)/\log\log(m+n)})$ rounds.

Description of a Distributed Algorithm. The algorithm proposed by Li et al. (2024b), which will be used later, is depicted in Algorithm 1. The three parameters α, f, and L are defined as

$$\alpha := 1 + \frac{\epsilon}{c \cdot \Gamma_d}, \text{ and } f := \frac{2}{\epsilon \cdot \ln \alpha} \cdot \ln \Gamma_p, \text{ and } L := \lceil \log_\alpha \Gamma_p + f \rceil. \tag{1}$$

where c is a sufficiently large constant, say $c = 1000$. Each left node u_j maintains two variables x_j and r_j, and each right node v_i maintains a variable y_i. Every x_j and y_i is initially 0 and the value can only increase throughout the algorithm; every r_j is initially 1 and the value can only decrease. Besides, each right node v_i also maintains a variable $\rho_i := \sum_j A_{ij} r_j$. Recall that the set of neighbors of u_i is denoted by N_i.

Theorem 1 (Li et al. (2024b)). *Given any $\epsilon > 0$, Algorithm 1 computes $(1+\epsilon)$-approximate solutions to the normalized packing LP and the normalized covering LP at the same time, running in $\le 10 \cdot \Gamma_d \cdot \log \Gamma_p / \epsilon^2$ rounds. Particularly, for row-sparse, column-sparse binary LP instances, Algorithm 1 runs in constant rounds.*

Algorithm 1: An $(1+\epsilon)$-approximation distributed algorithm for packing/covering LPs (Li et al., 2024b)

Input: A $m \times n$ matrix $\boldsymbol{A}$ where each A_{ij} is either 0 or ≥ 1, and a real number $\epsilon > 0$;
Parameters: $\alpha, f \in \mathbb{R}_{\geq 0}$ and $L \in \mathbb{N}$ defined as in equation 1;
Initialize $x_j := 0$ and $r_j := 1$ for any $j \in [n]$, and $y_i := 0$ for any $i \in [m]$;
for $\ell = 1$ *to* L **do**
  for *each right node* v_i *in parallel* **do**
    if $\rho_i \geq \frac{1}{\alpha} \cdot \max_{i':N_{i'} \cap N_i \neq \emptyset} \rho_{i'}$ **then** $\Delta y_i := 1$;
    else $\Delta y_i := 0$;
    $y_i := y_i + \Delta y_i$;
    send Δy_i and ρ_i to all of its neighbors;
  for *each left node* u_j *in parallel* **do**
    $x_j := x_j + r_j \cdot \sum_i A_{ij} \cdot \Delta y_i / \rho_i$ and $r_j := r_j / \alpha^{\sum_i A_{ij} \Delta y_i}$;
    if $r_j \leq \alpha^{-f}$ **then** $r_j := 0$;
    send r_j to all of its neighbors;
Return: $\boldsymbol{x}/(f \cdot (1+\epsilon))$ and $\boldsymbol{y}/f$ as a $(1+\epsilon)$-approximate solution to the normalized packing LP and the normalized covering LP respectively.

For the intuition behind Algorithm 1 and the analysis, we refer interested readers to (Li et al., 2024b).

2.2 Connection with Graph Neuron Networks for LPs

The connection builds on the observation that distributed LP algorithms and GNNs employ the same bipartite graph to represent an LP instance. On one hand, it is obvious that a d-depth GNN can be computed by a d-round distributed algorithm; so by the lower bound obtained by Kuhn et al. (2016), we have the following lemma:

Lemma 1. *There exists no $o(\sqrt{\log(m+n)/\log\log(m+n)})$-depth GNN that can output a constant-factor approximate solution for every sparse binary LP instance.*

On the other hand, by utilizing the universal approximation property of MLPs, any d-round (anonymous) distributed LP algorithm can be simulated by a d-depth (anonymous) GNN. In particular, by unrolling Algorithm 1, we have the following theorem. The proof can be found in the appendix.

Theorem 2. *Given any $\epsilon > 0$, there exists a constant-depth, constant-width GNN such that: for any row-sparse, column-sparse binary LP instance, it outputs $(1+\epsilon)$-approximate solutions to the primal LP and the dual LP at the same time.*

3 Main Result

By Lemma 1, it is impossible that a constant-depth GNN can approximately solve sparse binary LPs in the worst case. Despite this, if we care about the average-case performance, then such a GNN exists:

Theorem 3. *Given any $\epsilon > 0$ and $0 < \eta < 1$, there exists a constant-depth, constant-width GNN such that: for $(1-\eta)$ fraction of all possible sparse binary LP instances, it outputs a $(1+\epsilon)$-approximate solution to the primal LP and the dual LP at the same time.*

Remark 3. *To be more specific, in Theorem 3, the GNN depth is upper bounded by $10\gamma\beta \cdot \log(\gamma\alpha\beta)/\epsilon^2$, where $\alpha = m/n$, $\beta = nnz(A)/m$, and γ is defined as in equation 2. For example, if the GNN is required to output a 2-approximate solution for 99% fraction of LP instances where $m = n$ and $nnz(A) = 20m$, then the upper bound is 54079.*

While the theoretical constants may be large, we will empirically demonstrate that a 5-layer GNN designed by unrolling Algorithm 2 can solve sparse binary LPs. See Section 4 for details.

In the rest of this section, we will prove Theorem 3. We will propose a constant-round distributed algorithm that works for almost all sparse binary LPs, and can be naturally converted to a constant-depth, constant-width GNN.

3.1 Notations and Properties about average instances

Recall that the constraint matrix $\boldsymbol{A}$ has dimensions $m \times n$. Let α denote the ratio m/n. Since we have assumed $m = \Theta(n)$, α is bounded away from 0 and also upper bounded by a constant. Let $\mathcal{A}_{m,n,\beta}$ denote the set consisting of all binary matrix $\boldsymbol{A} \in \{0,1\}^{m\times n}$ having exactly $\beta \cdot m$ non-zero entries. For $A \in \mathcal{A}_{m,n,\beta}$, each row contains β non-zero entries in average, and we say a row γ-*dense* if it contains at least $\gamma\beta$ non-zero entries; similarly, each column contains $\alpha\beta$ non-zero entries in average, and we say a column γ-*dense* if it contains at least $\gamma\alpha\beta$ non-zero entries. Let $R_\gamma \subset [m]$ (and $C_\gamma \subset [n]$ respectively) denote the set of γ-dense rows (and columns respectively). Let $C_\gamma^{\text{int}} := \{j \mid \exists i \in R_\gamma \text{ s.t. } A_{ij} \neq 0\} \subset [n]$ denote the set consisting of all columns which intersect with some γ-dense row. We define $\text{opt}(\boldsymbol{A})$ to be the optimal objective value of the binary LP instance with constraint matrix $\boldsymbol{A}$, i.e.,

$$\text{opt}(A) := \max\left\{\sum_j x_j \mid \boldsymbol{A}\cdot\boldsymbol{x} \le \boldsymbol{1}, \boldsymbol{x} \ge 0\right\} = \min\left\{\sum_i y_i \mid \boldsymbol{A}^T\cdot\boldsymbol{y} \ge \boldsymbol{1}, \boldsymbol{y} \ge 0\right\}.$$

The following property about an "average" $\boldsymbol{A}$ will be used.

Lemma 2. *Let $\gamma \ge 8$. For at least $1-\eta$ fraction of $\boldsymbol{A}$ in $\mathcal{A}_{m,n,\beta}$, we have $|R_\gamma| \le \frac{6m}{\eta\cdot 2^{(\gamma-2)\beta}}$, $|C_\gamma| \le \frac{6n}{\eta\cdot 2^{(\gamma-2)\alpha\beta}}$, and $|C_\gamma^{\text{int}}| \le \frac{6\alpha(\gamma\beta+1)n}{\eta\cdot 2^{(\gamma-2)\beta}}$. In particular, by setting γ a sufficiently large constant, say*

$$\gamma = 2 + \frac{1}{\beta}\cdot\log^2\left(\frac{48\alpha\beta}{\eta\epsilon}\right) + \frac{1}{\alpha\beta}\cdot\log^2\left(\frac{48\beta}{\eta\epsilon}\right), \tag{2}$$

we have $|R_\gamma|, |C_\gamma| \le \frac{\epsilon n}{8\gamma\beta}$ and $|C_\gamma^{\text{int}}| + |C_\gamma| \le \frac{n}{2}$ for at least $1-\eta$ fraction of $\boldsymbol{A}$ in $\mathcal{A}_{m,n,\beta}$.

Proof. Let $\boldsymbol{A}$ denote a uniformly sampled matrix from $\mathcal{A}_{m,n,\alpha}$ at random. We use Z_k^i to denote the event that the i-th row of $\boldsymbol{A}$ contains exactly k 1-entries. For any $i \in [m]$ and $k \ge 8\beta$, we have

$$\begin{aligned}
\Pr_{\boldsymbol{A}}[Z_{k+1}]/\Pr_{\boldsymbol{A}}[Z_k] &= \binom{n}{k+1}\binom{(m-1)n}{\beta m-(k+1)}\Big/\binom{n}{k}\binom{(m-1)n}{\beta m-k} \\
&= \frac{n-k}{k+1}\cdot\frac{\beta m-k}{(m-1)n-\beta m+k+1} \le \frac{n-4\beta}{8\beta+1}\cdot\frac{\beta}{\frac{m-1}{m}\cdot n-\beta} \\
&\le \frac{1}{8}\cdot\frac{n-4\beta}{n/2-\beta} \le \frac{1}{2},
\end{aligned}$$

which implies that

$$\Pr[Z_k^i] \le 2^{-(k-2\beta)}\cdot\Pr[Z_{2\beta}^i] \le 2^{-(k-2\beta)}. \tag{3}$$

Furthermore, we have

$$\Pr[\text{the } i\text{-th row is } \gamma\text{-dense}] = \sum_{k=\gamma\beta}^{n}\Pr[Z_k^i] \le \sum_{k=\gamma\beta}^{\infty} 2^{-(k-2\beta)} = 2^{-(\gamma-2)\beta+1}.$$

Then, by linearity of expectation,

$$\mathbb{E}_{\boldsymbol{A}}\left[|R_\gamma|\right] = \sum_{i=1}^{m}\Pr_{\boldsymbol{A}}[\text{the } i\text{-th row is } \gamma\text{-dense}] \le 2^{-(\gamma-2)\beta+1}\cdot m.$$

By applying Markov's inequality,

$$\Pr_{\boldsymbol{A}}\left[|R_\gamma| \ge \frac{3}{\eta}\cdot 2^{-(\gamma-2)\beta+1}\cdot m\right] \le \frac{\eta}{3}. \tag{4}$$

Algorithm 2: A distributed algorithm that works for almost all instances of sparse binary LP

Input: A matrix $\boldsymbol{A} \in \{0,1\}^{m \times n}$ with βm 1-entries, and two real numbers $\epsilon > 0, 0 < \eta < 1$;
Parameters: γ defined as in equation 2;
for *each right node* v_i *in parallel* **do**
  if $\boldsymbol{A}_{i,:}$ *is* γ*-dense* **then**
    $y'_i := 1$;
    send "sleep" to all of its neighbors;
    set itself to the sleep mode;
for *each left node* u_j *in parallel* **do**
  if *recieves "sleep", or* $\boldsymbol{A}_{:,j}$ *is* γ*-dense* **then**
    send ("sleep",$1/\sum_i A_{ij}$) to all of its neighbors;
    set itself to the sleep mode;
Run Algorithm 1 with parameter $\epsilon/4$ on the awake nodes, and obtain $\hat{\boldsymbol{x}}'$ and $\hat{\boldsymbol{y}}'$;
for *each left node* u_j *in parallel* **do**
  if u_j *is in sleep mode* **then** return $\hat{x}_i = 0$;
  else return $\hat{x}_i = \hat{x}'_i$;
for *each right node* v_i *in parallel* **do**
  return $\hat{y}_i = \hat{y}'_i + \sum_{\text{sleep } j \in N_i} \frac{1}{\sum_{i'} A_{i'j}}$;

Similarly, we can also show that

$$\Pr_{\boldsymbol{A}}\left[|C_\gamma| \geq \frac{3}{\eta} \cdot 2^{-(\gamma-2)\alpha\beta+1} \cdot n\right] \leq \frac{\eta}{3}. \tag{5}$$

In the rest of the proof, we will show that

$$\Pr_{\boldsymbol{A}}\left[|C_\gamma^{\text{int}}| \geq \frac{3}{\eta} \cdot (\gamma\beta+1) \cdot 2^{-(\gamma-2)\beta+1} \cdot m\right] \leq \frac{\eta}{3}, \tag{6}$$

which, together with equation 4 and equation 5, imply the conclusion by the union bound.

Note that $|C_\gamma^{\text{int}}|$ is upper bounded by the total number of 1-entries in γ-dense rows. So by linearity of expectation, we have

$$\begin{aligned}
\mathbb{E}\left[|C_\gamma^{\text{int}}|\right] \leq & \mathbb{E}\left[\#\text{1-entries contained in the } \gamma\text{-dense rows}\right] \\
= & \sum_{i=1}^{m} \mathbb{E}\left[(\mathbf{1}_{i\text{-th row is } \gamma\text{-dense}}) \cdot (\#\text{1-entries contained in } i\text{-th row})\right] \\
= & \sum_{i=1}^{m} \sum_{k=\gamma\beta}^{n} \Pr[Z_k^i] \cdot k \leq \sum_{i=1}^{m} \sum_{k=\gamma\beta}^{+\infty} 2^{-(k-2\beta)} \cdot k \\
= & m(\gamma\beta+1) \cdot 2^{-(\gamma-2)\beta+1}
\end{aligned}$$

By applying Markov's inequality, we obtain equation 6. □

3.2 Construction of the constant-depth, constant-width GNN

In the following, we will set γ as in equation 2, which is a constant. To prove Theorem 3, we will propose a constant-round distributed algorithm that works for almost all sparse binary LP instances. Then, as we will see, it can be naturally converted to a constant-depth, constant-width GNN. The distributed algorithm builds on Algorithm 1. The idea is depicted as follows: given $\boldsymbol{A} \in \mathcal{A}_{m,n,\beta}$, we first obtain a reduced matrix $\boldsymbol{A}'$ from $\boldsymbol{A}$ by deleting all rows in R_γ and all columns in $C_\gamma \cup C_\gamma^{\text{int}}$. Note that $\boldsymbol{A}'$ is a row-sparse, column-sparse binary matrix. Then we run Algorithm 1 with $\epsilon/4$ on

the reduced LP instance with constraint matrix $\boldsymbol{A}'$, where the reduced primal LP is

$$\max\left\{\sum_{j\in[n]\setminus(C_\gamma\cup C_\gamma^{\text{int}})} x_j \mid \boldsymbol{A}'\boldsymbol{x}\le\mathbf{1},\boldsymbol{x}\ge 0\right\}$$

and the reduced dual LP is

$$\min\left\{\sum_{i\in[m]\setminus R_\gamma} y_i \mid \boldsymbol{A}'^T\boldsymbol{y}\ge\mathbf{1},\boldsymbol{x}\ge 0\right\},$$

to obtain $(1+\epsilon/4)$-approximate solutions $\hat{\boldsymbol{x}}'\in\mathbb{R}^{[n]\setminus(C_\gamma\cup C_\gamma^{\text{int}})}$ and $\hat{\boldsymbol{y}}'\in\mathbb{R}^{[m]\setminus R_\gamma}$ of the reduced instance. Finally,

- Let $\hat{\boldsymbol{x}}\in\mathbb{R}^n$ be obtained from $\hat{\boldsymbol{x}}'$ by setting each x_j in $C_\gamma\cup C_\gamma^{\text{int}}$ to 0. Return $\hat{\boldsymbol{x}}$ as an $(1+\epsilon)$-approximate solution to the original primal LP.
- Let $\hat{\boldsymbol{y}}$ be obtained from $\hat{\boldsymbol{y}}'$ by first setting each y_i in R_γ to 1, and then adding the vector $\sum_{j\in C_\gamma}\frac{1}{\sum_i A_{ij}}\cdot\boldsymbol{A}_{:,j}$. Return $\hat{\boldsymbol{y}}$ as an $(1+\epsilon)$-approximate solution to the original dual LP.

The formal description of the distributed algorithm is presented in Algorithm 2. It is easy to see that this algorithm runs in constant rounds and can be naturally converted to a constant-depth, constant-width GNN. What remains is to show the correctness.

3.3 Correctness

By Lemma 2, it suffices to show that for every $\boldsymbol{A}\in\mathcal{A}_{m,n,\beta}$ where $|R_\gamma|,|C_\gamma|\le\frac{\epsilon n}{8\gamma\beta}$, and $|C_\gamma^{\text{int}}|+|C_\gamma|\le n/2$, $\hat{\boldsymbol{x}}$ and $\hat{\boldsymbol{y}}$ returned by Algorithm 2 are $(1+\epsilon)$-approximate solutions to the original primal LP $\max\{\sum_{j=1}^n x_j\mid\boldsymbol{A}\boldsymbol{x}\le\mathbf{1},\boldsymbol{x}\ge 0\}$ and the dual LP $\min\{\sum_{i=}^m y_i\mid\boldsymbol{A}^T\boldsymbol{y}\ge\mathbf{1},\boldsymbol{y}\ge 0\}$ respectively. We first present some helpful observations.

Claim 1. $\text{opt}(\boldsymbol{A})-\frac{\epsilon n}{4\gamma\beta}\le\text{opt}(\boldsymbol{A})-|R_\gamma|-|C_\gamma|\le\text{opt}(\boldsymbol{A}')\le\text{opt}(\boldsymbol{A})$.

Proof. We first show that $\text{opt}(\boldsymbol{A}')\le\text{opt}(\boldsymbol{A})$. Suppose that $\boldsymbol{x}'^*\in\mathbb{R}^{[n]\setminus(C_\gamma\cup C_\gamma^{\text{int}})}$ is an optimal solution to the reduced primal LP. Let $\boldsymbol{x}\in\mathbb{R}^{[n]}$ be obtained from $\boldsymbol{x}'^*$ by setting each x_j in $C_\gamma\cup C_\gamma^{\text{int}}$ to 0. It is easy to check that $\boldsymbol{x}$ is a feasible solution to the original primal LP. So

$$\text{opt}(\boldsymbol{A}')=\sum_i x_i'^*=\sum_i x_i\le\text{opt}(\boldsymbol{A}).$$

Now, we show that $\text{opt}(\boldsymbol{A})\le\text{opt}(\boldsymbol{A}')+|R_\gamma|+|C_\gamma|$. Suppose that $\boldsymbol{y}'^*\in\mathbb{R}^{[m]\setminus R_\gamma}$ is an optimal solution to the reduced dual LP. Let $\boldsymbol{y}$ be obtained from $\boldsymbol{y}'^*$ by (i) first setting each y_i in R_γ to 1, and (ii) then adding $\sum_{j\in C_\gamma}\frac{1}{\sum_i A_{ij}}\cdot\boldsymbol{A}_{:,j}$. We claim that $\boldsymbol{y}$ is a feasible solution to the original dual LP, i.e., $\boldsymbol{A}_j^T\boldsymbol{y}=\sum_{i=1}^m A_{ij}y_i\ge 1$ for any $j\in[n]$. The feasibility can be checked case by case:

- If $j\in[n]\setminus\left(C_\gamma\cup C_\gamma^{\text{int}}\right)$, then we have $\boldsymbol{A}_j^T\boldsymbol{y}\ge\boldsymbol{A}_j'^T\boldsymbol{y}'^*\ge 1$.
- If $j\in C_\gamma$, then $\boldsymbol{A}_j^T\boldsymbol{y}\ge\boldsymbol{A}_j^T\cdot\frac{1}{\sum_i A_{ij}}\cdot\boldsymbol{A}_{:,j}=1$.
- If $j\in C_\gamma^{\text{int}}$, then there is a $i\in R_\gamma$ such that $A_{ij}=1$. Note that $y_i\ge 1$ for any $i\in R_\gamma$. We have $\boldsymbol{A}_j^T\boldsymbol{y}\ge A_{ij}\cdot y_i\ge 1$.

Finally, since $\boldsymbol{y}$ is a feasible solution to the original dual LP, it has

$$\text{opt}(\boldsymbol{A})=\sum_j y_j\le\sum_j y_j'^*+|R_\gamma|+\sum_i\sum_{j\in C_\gamma}\frac{1}{\sum_i A_{ij}}\cdot A_{i,j}=\text{opt}(\boldsymbol{A}')+|R_\gamma|+|C_\gamma|.\quad\square$$

Claim 2. $\text{opt}(\boldsymbol{A}')\ge\frac{n}{2\gamma\beta}$.

Proof. Note that $\boldsymbol{A}'$ contains no γ-dense rows, i.e., each row of $\boldsymbol{A}'$ contains no more than $\gamma\beta$ 1-entries. Thus $\left(\frac{1}{\gamma\beta}, \frac{1}{\gamma\beta}, \cdots, \frac{1}{\gamma\beta}\right)$ is a feasible solution to the reduced primal LP, and the objective value $\frac{1}{\gamma\beta} \cdot \left(n - |C_\gamma^{\text{int}} \cup C_\gamma|\right) \geq \frac{n}{2\gamma\beta}$ provides a lower bound on $\text{opt}(\boldsymbol{A}')$. □

In the following, we show that $\hat{\boldsymbol{x}}$ is a $(1+\epsilon)$-approximate solution to the original primal LP. The feasibility is obvious, and the optimality can be verified as follows:

$$\begin{aligned}
(1+\epsilon)\sum_{j\in[n]}\hat{x}_j =& (1+\epsilon)\sum_{j\in[n]\setminus R_\gamma}\hat{x}'_j \geq \frac{1+\epsilon}{1+\epsilon/4}\cdot\text{opt}(\boldsymbol{A}') \\
&\geq \left(1+\frac{\epsilon}{2}\right)\text{opt}(\boldsymbol{A}') = \text{opt}(\boldsymbol{A}') + \frac{\epsilon}{2}\cdot\text{opt}(\boldsymbol{A}') \\
&\geq \text{opt}(\boldsymbol{A}) - \frac{\epsilon n}{4\gamma\beta} + \frac{\epsilon}{2}\cdot\text{opt}(\boldsymbol{A}') && \text{(Claim 1)} \\
&\geq \text{opt}(\boldsymbol{A}) - \frac{\epsilon n}{4\gamma\beta} + \frac{\epsilon}{2}\cdot\frac{n}{2\gamma\beta} && \text{(Claim 2)} \\
&\geq \text{opt}(\boldsymbol{A}).
\end{aligned}$$

What remains is to show that $\hat{\boldsymbol{y}}$ is a $(1+\epsilon)$-approximate solution to the original dual LP. The feasibility can be checked similarly as in the proof of Claim 1. The optimality can be verified as follows:

$$\begin{aligned}
\sum_j \hat{y}_j \leq& \sum_j \hat{y}'_j + |R_\gamma| + \sum_i \sum_{j\in C_\gamma} \frac{1}{\sum_i A_{ij}}\cdot A_{ij} = \sum_j \hat{y}'_j + |R_\gamma| + |C_\gamma| \\
&\leq \left(1+\frac{\epsilon}{4}\right)\cdot\text{opt}(\boldsymbol{A}') + \frac{\epsilon n}{4\gamma\beta} \\
&\leq \left(1+\frac{\epsilon}{4}\right)\cdot\text{opt}(\boldsymbol{A}') + \frac{\epsilon}{2}\cdot\text{opt}(\boldsymbol{A}') && \text{(Claim 2)} \\
&\leq \left(1+\frac{3\epsilon}{4}\right)\cdot\text{opt}(\boldsymbol{A}) && \text{(Claim 1)}
\end{aligned}$$

Now, we finish the proof of the correctness.

4 Numerical Experiments

We conduct numerical experiments to validate our main theoretical results, namely Theorem 3. By unrolling Algorithm 2, we propose a GNN architecture, depicted in Appendix B.

Experimental Setup. To validate the performance of our method across different sizes of Normalized packing LP problems and ensure consistency with theoretical results, we randomly generated LP problems of varying sizes using Ortools. Specifically, the LP instances generated were of the form $M = N$, with each instance containing $10 * (M + N)$ non-zero entries. Four different problem sizes were considered: 100, 500, 1,000, and 1,500. After generation, we employed Ortools to solve these problems, collecting both the primal and dual solutions for training and testing.

For each problem size, we used 100 independent, randomly generated training samples and 100 independent, randomly generated testing samples. Our model was implemented using the PyTorch framework, and the training process was conducted on GPUs. The Adam optimizer was used with a learning rate of 1e-3 and a batch size of 1. Fixed parameters included $L = 5$, and $K = 16$, along with a learnable parameter f initialized to 1. In the comparison experiments, we use GCNs with the same L-layer cut each with 64-dimensional feature dimensions.

Evaluation Metrics. To assess the representational power of our method, we report the training loss after the model has converged. Additionally, we evaluate the relative gap. Using feasibility restoration (see Appendix C), the final values x^{final} and y^{final} returned by our GNN are used to compute x' and y'. The relative gaps are then calculated as follows: the relative gap of the primal is denoted as $RP = \frac{\text{obj}(x') - \text{opt}}{\text{opt}}$ and the relative gap of the dual is denoted as $RD = \frac{\text{obj}(y') - \text{opt}}{\text{opt}}$.

In order to explore the difference in the representational power of the two methods, we report the converged training loss, which is the training loss in the table. The lower the RP and RD the better the performance.

Experimental Results. The results of the numerical experiments directly illustrate the effectiveness of our method. Table 1 demonstrates that our GNN achieves much better performance while using significantly fewer parameters, where our model uses at most 2K parameters, which is almost 0.8% of the number of parameters in GCN (≈250K).

Table 1: Performance comparison of our method with GCN for training loss , Training RP, Training RD, Test RP and Test RD.

LP size	Training Loss		Training RP		Training RD		Test RP		Test RD	
	GCN	Ours	GCN	Ours	GCN	Ours	GCN	Ours	GCN	Ours
100	0.0087	**0.0086**	1.0	**0.20**	18.64	**0.43**	1.0	**0.20**	18.62	**0.45**
500	0.0086	**0.0084**	0.96	**0.20**	18.37	**0.48**	0.94	**0.20**	18.35	**0.47**
1000	**0.0079**	0.0085	1.0	**0.20**	18.33	**0.47**	1.0	**0.20**	18.35	**0.46**
1500	**0.0083**	0.0085	1.0	**0.20**	18.33	**0.45**	1.0	**0.20**	18.38	**0.46**

We also conduct small-scale numerical experiments to illustrate that our method TRAINING LOSS can converge to 0. In the experiments, the size of the LP problem is 1000 and the training samples are 10. We report the relationship (see Fig. 1) of converged training loss with the number of layers L. Even for very small L, the training loss for convergence can already be close to 0.

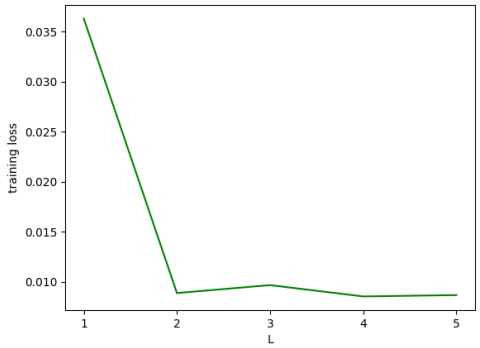

Figure 1: Convergence trends due to changes in the number of layers during training.

5 Conclusion

Towards understanding the empirical success achieved by shallow GNNs for solving LPs, we prove that constant-depth, constant-width GNNs suffice to solve sparse binary LPs effectively, by leveraging the principles of distributed LP algorithms. Besides, our analysis shifts the focus from worst-case performance to average-case performance, as empirical studies usually measure GNNs based on average performance across instances. We believe our ideas can help in further explorations of the power of GNNs in L2O. For future directions, it is interesting to investigate whether our result can be extended to more general LP classes, such as non-negative LPs.

References

Mohamad Ahmadi, Fabian Kuhn, and Rotem Oshman. Distributed approximate maximum matching in the CONGEST model. In Ulrich Schmid and Josef Widder (eds.), *32nd International Sym-*

posium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018, volume 121 of *LIPIcs*, pp. 6:1–6:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

David Applegate, Mateo Díaz, Oliver Hinder, Haihao Lu, Miles Lubin, Brendan O'Donoghue, and Warren Schudy. Practical large-scale linear programming using primal-dual hybrid gradient. *Advances in Neural Information Processing Systems*, 34:20243–20257, 2021.

David Applegate, Oliver Hinder, Haihao Lu, and Miles Lubin. Faster first-order primal-dual methods for linear programming using restarts and sharpness. *Mathematical Programming*, 201(1):133–184, 2023.

Baruch Awerbuch and Rohit Khandekar. Stateless distributed gradient descent for positive linear programs. *SIAM J. Comput.*, 38(6):2468–2486, 2009.

Yair Bartal, John W. Byers, and Danny Raz. Global optimization using local information with applications to flow control. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pp. 303–312. IEEE Computer Society, 1997.

Kinjal Basu, Amol Ghoting, Rahul Mazumder, and Yao Pan. Eclipse: An extreme-scale linear program solver for web-applications. In *International Conference on Machine Learning*, pp. 704–714. PMLR, 2020.

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

Quentin Cappart, Didier Chételat, Elias B. Khalil, Andrea Lodi, Christopher Morris, and Petar Velickovic. Combinatorial optimization and reasoning with graph neural networks. *J. Mach. Learn. Res.*, 24:130:1–130:61, 2023.

Antonin Chambolle and Thomas Pock. On the ergodic convergence rates of a first-order primal–dual algorithm. *Mathematical Programming*, 159(1):253–287, 2016.

Ziang Chen, Jialin Liu, Xinshang Wang, and Wotao Yin. On representing linear programs by graph neural networks. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

Qi Deng, Qing Feng, Wenzhi Gao, Dongdong Ge, Bo Jiang, Yuntian Jiang, Jingsong Liu, Tianhao Liu, Chenyu Xue, Yinyu Ye, et al. New developments of admm-based interior point methods for linear programming and conic programming. *arXiv preprint arXiv:2209.01793*, 2022.

Zhenan Fan, Xinglu Wang, Oleksandr Yakovenko, Abdullah Ali Sivas, Owen Ren, Yong Zhang, and Zirui Zhou. Smart initial basis selection for linear programs. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 9650–9664. PMLR, 2023.

Patrik Floréen, Marja Hassinen, Petteri Kaski, and Jukka Suomela. Tight local approximation results for max-min linear programs. In Sándor P. Fekete (ed.), *Algorithmic Aspects of Wireless Sensor Networks, Fourth International Workshop, ALGOSENSORS 2008, Reykjavik, Iceland, July 2008. Revised Selected Papers*, volume 5389 of *Lecture Notes in Computer Science*, pp. 2–17. Springer, 2008. doi: 10.1007/978-3-540-92862-1_2. URL https://doi.org/10.1007/978-3-540-92862-1_2.

Patrik Floréen, Marja Hassinen, Joel Kaasinen, Petteri Kaski, Topi Musto, and Jukka Suomela. Local approximability of max-min and min-max linear programs. *Theory Comput. Syst.*, 49(4): 672–697, 2011. doi: 10.1007/S00224-010-9303-6. URL https://doi.org/10.1007/s00224-010-9303-6.

Benjamin Hudson, Qingbiao Li, Matthew Malencia, and Amanda Prorok. Graph neural network guided local search for the traveling salesperson problem. In *International Conference on Learning Representations*, 2021.

Yufei Kuang, Xijun Li, Jie Wang, Fangzhou Zhu, Meng Lu, Zhihai Wang, Jia Zeng, Houqiang Li, Yongdong Zhang, and Feng Wu. Accelerate presolve in large-scale linear programming via reinforcement learning. *CoRR*, abs/2310.11845, 2023.

Fabian Kuhn and Roger Wattenhofer. Constant-time distributed dominating set approximation. *Distributed Comput.*, 17(4):303–310, 2005. doi: 10.1007/S00446-004-0112-5. URL https://doi.org/10.1007/s00446-004-0112-5.

Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The price of being near-sighted. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pp. 980–989. ACM Press, 2006.

Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local computation: Lower and upper bounds. *J. ACM*, 63(2):17:1–17:44, 2016.

Bingheng Li, Linxin Yang, Yupeng Chen, Senmiao Wang, Haitao Mao, Qian Chen, Yao Ma, Akang Wang, Tian Ding, Jiliang Tang, and Ruoyu Sun. Pdhg-unrolled learning-to-optimize method for large-scale linear programming. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024a. URL https://openreview.net/forum?id=2cXzNDe614.

Qian Li, Minghui Ouyang, and Yuyi Wang. A simple distributed algorithm for sparse fractional covering and packing problems. *arXiv preprint arXiv:2409.16168*, 2024b.

Xijun Li, Qingyu Qu, Fangzhou Zhu, Jia Zeng, Mingxuan Yuan, Kun Mao, and Jie Wang. Learning to reformulate for linear programming. *arXiv preprint arXiv:2201.06216*, 2022.

Tianyi Lin, Shiqian Ma, Yinyu Ye, and Shuzhong Zhang. An admm-based interior-point method for large-scale linear programming. *Optimization Methods and Software*, 36(2-3):389–424, 2021.

Tianhao Liu, Shanwen Pu, Dongdong Ge, and Yinyu Ye. Learning to pivot as a smart expert. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan (eds.), *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pp. 8073–8081. AAAI Press, 2024.

Haihao Lu. First-order methods for linear programming. *arXiv preprint arXiv:2403.14535*, 2024.

Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.

Vishal Monga, Yuelong Li, and Yonina C Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Processing Magazine*, 38(2):18–44, 2021.

Brendan O'Donoghue. Operator splitting for a homogeneous embedding of the linear complementarity problem. *SIAM Journal on Optimization*, 31(3):1999–2023, 2021.

Brendan O'donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169:1042–1068, 2016.

Christos H. Papadimitriou and Mihalis Yannakakis. Linear programming without the matrix. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal (eds.), *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pp. 121–129. ACM, 1993.

David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.

Yun Peng, Byron Choi, and Jianliang Xu. Graph learning for combinatorial optimization: a survey of state-of-the-art. *Data Science and Engineering*, 6(2):119–141, 2021.

Chendi Qian, Didier Chételat, and Christopher Morris. Exploring the power of graph neural networks in solving linear optimization problems. In *International Conference on Artificial Intelligence and Statistics*, pp. 1432–1440. PMLR, 2024.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Approximation ratios of graph neural networks for combinatorial problems. *Advances in Neural Information Processing Systems*, 32, 2019.

Martin JA Schuetz, J Kyle Brubaker, and Helmut G Katzgraber. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4(4):367–377, 2022.

Luca Trevisan and Fatos Xhafa. The parallel complexity of positive linear programming. *Parallel Process. Lett.*, 8(4):527–533, 1998. doi: 10.1142/S0129626498000511. URL https://doi.org/10.1142/S0129626498000511.

A Proof of Theorem 2

We will explicitly describe such a GNN. The idea is to simulate Algorithm 1 on binary LP instances. Given a matrix $\boldsymbol{M} \in \mathbb{R}^{m\times n}$ and a vector $\boldsymbol{p} \in \mathbb{R}^n$, we define the "max-product" $\boldsymbol{q} = \boldsymbol{M} \times_{\max} \boldsymbol{p}$ to be the m-dimensional column vector such that $q_i = \max_{j=1}^n M_{ij}p_j$

We design a GNN as follows: each left node in the bipartite graph maintains a 4-dimension feature vector $(x_j, \tilde{r}_j, b_j, \tilde{\rho}_j)$, and each right node also maintains a 4-dimension feature vector $(y_i, y_{\Delta,i}, \rho_i, \rho_{\max,i})$. Here,

- $\bar{r}_j$ represents $\log_\alpha r_j + f$. In other words, $r_j = \alpha^{\bar{r}_j - f}$.
- b_j is defined to be r_j (or equivalently $\alpha^{\bar{r}_j - f}$) if $r_j > \alpha^{-f}$, and 0 otherwise.
- ρ_i represents $\boldsymbol{A}_{i,:} \cdot \boldsymbol{b} = \sum_j A_{ij}b_j$.
- $\tilde{\rho}_j$ represents $\max_{i:N_i \ni j} \rho_i$, or equivalently $\boldsymbol{A}^T_{:,j} \times_{\max} \boldsymbol{\rho}$.
- $\rho_{\max,i}$ represents $\max_{i':N_{i'}\cap N_i \neq \emptyset} \rho_{i'} = \max_{j\in N_i} \tilde{\rho}_j$, or equivalently $\boldsymbol{A}_{i,:} \times_{\max} \tilde{\rho}$.
- $y_{\Delta,i}$ represents the increment of y_i.

The architecture of the GNN is depicted below.

- Initialize $\bar{\boldsymbol{r}}^0 := f \cdot \boldsymbol{1}_{n\times 1}$, $\boldsymbol{b}^0 = \boldsymbol{1}_{n\times 1}$; and $\tilde{\boldsymbol{\rho}}^0, \boldsymbol{x}^0$ both to be $\boldsymbol{0}_{n\times 1}$.

- Initialize all of $\boldsymbol{y}^0, \boldsymbol{y}^0_\Delta, \boldsymbol{\rho}^0, \boldsymbol{\rho}^0_{\max}$ to be $\boldsymbol{0}_{m\times 1}$.

- For $\ell = 0, 1, 2 \cdots, L-1$

- $\boldsymbol{\rho}^{\ell+1} := \boldsymbol{A} \cdot \boldsymbol{b}^\ell$;
- $\tilde{\boldsymbol{\rho}}^{\ell+1} := \boldsymbol{A}^T \times_{\max} \boldsymbol{\rho}^{\ell+1}$;
- $\boldsymbol{\rho}^{\ell+1}_{\max} := \boldsymbol{A} \times_{\max} \tilde{\boldsymbol{\rho}}^{\ell+1}$, $\boldsymbol{y}^{\ell+1}_\Delta := g_{\boldsymbol{\theta}}(\boldsymbol{\rho}^{\ell+1} - \boldsymbol{\rho}^{\ell+1}_{\max}/\alpha)$, and $\boldsymbol{y}^{\ell+1} := \boldsymbol{y}^\ell + \boldsymbol{y}^{\ell+1}_\Delta$;
- $\bar{\boldsymbol{r}}^{\ell+1} := \bar{\boldsymbol{r}}^\ell - \boldsymbol{A}^T \cdot \boldsymbol{y}^{\ell+1}_\Delta$, $\boldsymbol{x}^{\ell+1} := \boldsymbol{x}^\ell + \boldsymbol{b}^\ell \circ \left[\boldsymbol{A}^T \cdot \left(\boldsymbol{y}^{\ell+1}_\Delta \circ \frac{1}{\boldsymbol{\rho}^{\ell+1}}\right)\right]$, and $\boldsymbol{b}^{\ell+1} = b_{\boldsymbol{\theta}}(\bar{\boldsymbol{r}}^{\ell+1})$.

Here, $\circ$ denotes entry-wise multiplication (a.k.a. Hadamard product), and the parameterized functions $g_{\boldsymbol{\theta}}, b_{\boldsymbol{\theta}} : \mathbb{R} \to \mathbb{R}$ are applied entry-wise on to the vectors $\boldsymbol{\rho}^{\ell+1} - \boldsymbol{\rho}^{\ell+1}_{\max}/\alpha$ and $\bar{\boldsymbol{r}}^{\ell+1}$ respectively. If we set $g_{\boldsymbol{\theta}}(z) := \boldsymbol{1}_{z\geq 0}$ and $b_{\boldsymbol{\theta}}(z) := \boldsymbol{1}_{z\geq 0} \cdot \alpha^{z-f}$, then it is straightforward to check that this GNN exactly simulates Algorithm 1. By Theorem 1, we finish the proof.

B Graph neural network design by unrolling Algortihm 2

Each left node in the bipartite graph maintains a 5-dimension feature vector $(x_j, \tilde{r}_j, b_j, \tilde{\rho}_j, \text{mask}^{\leftarrow}_j) \in \mathbb{R}^5$, and each right node also maintains a 5-dimension feature vector $(y_i, y_{\Delta,i}, \rho_i, \rho_{\max,i}, \text{mask}^{\rightarrow}_j) \in \mathbb{R}^5$. Here,

- The feature mask indicates whether the node is in the sleep mode. For example, $\text{mask}^{\leftarrow}_j$ is supposed to be 1 if the left node u_j is in sleep mode, and 0 if not.
  - Recall that a right node v_i is set to be sleep if it has $\geq \gamma\beta$ edges. In the neural network, we will let $\textbf{mask}^{\rightarrow} := q^{\rightarrow}_{\boldsymbol{\theta}}(\boldsymbol{A} \cdot \boldsymbol{1}_{n\times 1})$, where $q^{\leftarrow}_{\boldsymbol{\theta}}(z) = \frac{1}{4}\sum_{k=1}^4 \sigma(\theta_{k,8} \cdot z + \theta_{k,9})$ is a learnable function and applied entry-wise. Here, $\sigma(z) = 1/(1+\exp(-z))$ is the sigmoid function.
  - Recall that a left node u_j is set to be sleep if it has $\geq \gamma\alpha\beta$ edges or connects to a sleep right node. In the neural network, we will let $\textbf{mask}^{\leftarrow} = \max\{q^{\leftarrow}_{\boldsymbol{\theta}}(\boldsymbol{A}^T \cdot \boldsymbol{1}_{m\times 1}), \boldsymbol{A}^T \times_{\max} \textbf{mask}^{\rightarrow}\}$, where $q^{\rightarrow}_{\boldsymbol{\theta}} = \frac{1}{4}\sum_{k=1}^4 \sigma(\theta_{k,10} \cdot z + \theta_{k,11})$ is a learnable function and applied entry-wise.
- The intuition behind the other features is the same as in Theorem 2.

The architecture of the GNN is depicted below.

- Initialize $\bar{\boldsymbol{r}}^0 := f \cdot \boldsymbol{1}_{n\times 1}$, $\boldsymbol{b}^0 = \boldsymbol{1}_{n\times 1}$; and $\tilde{\boldsymbol{\rho}}^0 = \boldsymbol{x}^0 = \textbf{mask}^{\leftarrow} = \boldsymbol{0}_{n\times 1}$.

- Initialize $\boldsymbol{y}^0 = \boldsymbol{y}^0_\Delta = \boldsymbol{\rho}^0 = \boldsymbol{\rho}^0_{\max} = \mathbf{mask}^{\rightarrow} = \mathbf{0}_{m\times 1}$.

- $\mathbf{mask}^{\rightarrow} = q^{\rightarrow}_{\boldsymbol{\theta}}(\boldsymbol{A}\cdot \mathbf{1}_{n\times 1})$.

- $\mathbf{mask}^{\leftarrow} = \max\{q^{\leftarrow}_{\boldsymbol{\theta}}(\boldsymbol{A}^T\cdot \mathbf{1}_{m\times 1}), \boldsymbol{A}^T \times_{\max} \mathbf{mask}^{\rightarrow}\}$.

- For $\ell = 0, 1, 2\cdots, L-1$

  - $\boldsymbol{\rho}^{\ell+1} := \left(\boldsymbol{A}\cdot \boldsymbol{b}^\ell\right)\circ(\mathbf{1}-\mathbf{mask}^{\rightarrow})$;
  - $\tilde{\boldsymbol{\rho}}^{\ell+1} := \left(\boldsymbol{A}^T \times_{\max} \boldsymbol{\rho}^{\ell+1}\right)\circ(\mathbf{1}-\mathbf{mask}^{\leftarrow})$;
  - $\boldsymbol{\rho}^{\ell+1}_{\max} := \left(\boldsymbol{A}\times_{\max}\tilde{\boldsymbol{\rho}}^{\ell+1}\right)\circ(\mathbf{1}-\mathbf{mask}^{\rightarrow})$, $\boldsymbol{y}^{\ell+1}_\Delta := g_{\boldsymbol{\theta}}(\boldsymbol{\rho}^{\ell+1}-\boldsymbol{\rho}^{\ell+1}_{\max}/\alpha)\circ(\mathbf{1}-\mathbf{mask}^{\rightarrow})$, and $\boldsymbol{y}^{\ell+1} := \left(\boldsymbol{y}^\ell + \boldsymbol{y}^{\ell+1}_\Delta\right)\circ(\mathbf{1}-\mathbf{mask}^{\rightarrow})$;
  - $\bar{\boldsymbol{r}}^{\ell+1} := \left(\bar{\boldsymbol{r}}^\ell - \boldsymbol{A}^T\cdot\boldsymbol{y}^{\ell+1}_\Delta\right)\circ(\mathbf{1}-\mathbf{mask}^{\leftarrow})$, $\boldsymbol{x}^{\ell+1} := \left(\boldsymbol{x}^\ell + \boldsymbol{b}^\ell\circ\left[\boldsymbol{A}^T\cdot\left(\boldsymbol{y}^{\ell+1}_\Delta\circ\frac{1}{\boldsymbol{\rho}^{\ell+1}}\right)\right]\right)\circ(\mathbf{1}-\mathbf{mask}^{\leftarrow})$, and $\boldsymbol{b}^{\ell+1} = b_{\boldsymbol{\theta}}(\bar{\boldsymbol{r}}^{\ell+1})\circ(\mathbf{1}-\mathbf{mask}^{\leftarrow})$.

- return $\boldsymbol{x}^{final} = \boldsymbol{x}^L$ and $\boldsymbol{y}^{final} = \boldsymbol{y}^L + \mathbf{mask}^{\rightarrow} + \overline{\boldsymbol{A}}\cdot\left(\boldsymbol{h}^T\circ\mathbf{mask}^{\leftarrow}\right)$.

Here, $\circ$ denotes entry-wise multiplication (a.k.a. Hadamard product), the $m\times n$-dimensional matrix $\overline{\boldsymbol{A}}$ is defined as $\overline{A}_{ij} = A_{ij}/\sum_{i=1}^m A_{ij}$, and the parameterized functions $g^\ell_{\boldsymbol{\theta}}, b^\ell_{\boldsymbol{\theta}}: \mathbb{R}\to\mathbb{R}$ are applied entry-wise on to the vectors $\boldsymbol{\rho}^{\ell+1} - \boldsymbol{\rho}^{\ell+1}_{\max}/\alpha$ and $\bar{\boldsymbol{r}}^{\ell+1}$ respectively where

$$g^\ell_{\boldsymbol{\theta}}(z) = \sum_{k=1}^K \theta^\ell_{k,1}\cdot\sigma(\theta^\ell_{k,2}\cdot z + \theta^\ell_{k,3}), \text{ and } \quad b^\ell_{\boldsymbol{\theta}}(z) := \sum_{k=1}^K \theta^\ell_{k,4}\cdot\sigma(\theta^\ell_{k,5}\cdot z+\theta^\ell_{k,6})\cdot 2^{-\mathrm{ReLU}(\theta^\ell_{k,7}(z-f))}$$

The learnable parameter is $\boldsymbol{\Theta} := \{\boldsymbol{h}\in\mathbb{R}^n, f\}\cup\{\theta^\ell_{k,q}\}_{\ell\in[0,L-1],k\in[1,K],q\in[1,11]}$. The number of parameter is $11LK+n+1$.

One can check that the above GNN can exactly simulate Algorithm 2.

Network training. The training data set is a set $\mathcal{I} = \{(\boldsymbol{A}, \boldsymbol{x}^*, \boldsymbol{y}^*)\}$ of binary LP instances. More specifically, the input of an instance is identified by the constraint matrix $\boldsymbol{A}$; the label $\boldsymbol{x}^*$ and $\boldsymbol{y}^*$ represents the corresponding optimal solutions to primal LP and dual LP respectively. Let $\boldsymbol{x}^{final}(\boldsymbol{\Theta}, \boldsymbol{A})$ and $\boldsymbol{y}^{final}(\boldsymbol{\Theta}, \boldsymbol{A})$ denote the output of our GNN parameterized by $\boldsymbol{\Theta}$ running on the input $\boldsymbol{A}$. The goal of the training process is to find a parameter $\boldsymbol{\Theta}^*$ minimizing MSE loss defined as:

$$\mathcal{L}(\mathcal{I};\boldsymbol{\Theta}) = \frac{1}{|\mathcal{I}|}\sum_{(A,\boldsymbol{x}^*)\in\mathcal{I}}\left[\frac{1}{N}\sum_{j\in[N]}|\boldsymbol{x}^{final}_j(\boldsymbol{\Theta},\boldsymbol{A}) - \boldsymbol{x}^*_j|^2 + \frac{1}{M}\sum_{i\in[M]}|\boldsymbol{y}^{final}_i(\boldsymbol{\Theta},\boldsymbol{A}) - \boldsymbol{y}^*_i|^2\right].$$

C Other detailed information about the experiments

Evaluation Configurations. All experiments were performed in the same hardware environment. The evaluation machine is equipped with two Intel(R) Xeon(R) Gold 5117 CPUs @ 2.00GHz, 256GB of RAM, and a single Nvidia V100 GPU. Ortools version 9.11 and PyTorch version 1.10.2 were utilized in our experiments. The maximum number of training epochs was set to 1,000.

Feasibility restoration for the primal LP. Note that the $\boldsymbol{x}^{final}$ returned by our GNN may be infeasible. To restore feasibility, we implement the following post-processing procedure:

– First, for each $j\in[n]$, update $x_j := \max(0, \min(1, x_j))$;

– Then, for $i = 1$ to m do

- If $\boldsymbol{A}_i\boldsymbol{x} \geq 1$, then update $x_j := \frac{x_j}{\boldsymbol{A}_i\boldsymbol{x}}$ for each j with $A_{ij}\neq 0$.

Feasibility restoration for the dual LP. Since the $\boldsymbol{y}^{final}$ returned by our GNN may be infeasible, we implement the following post-processing procedure to restore feasibility:

– First, for each $i\in[m]$, update $y_i := \max(10^{-5}, \min(1, y_i))$;

– Then, for $j = 1$ to n do

- If $\boldsymbol{A}_j^T \boldsymbol{y} \leq 1$, then update $y_i := \frac{y_i}{\boldsymbol{A}_j^T \boldsymbol{y}}$ for each i with $A_{ij} \neq 0$.