Exploring Linear Programs with Graphs and AI

First Author
Affiliation / Address line 1
email@domain

Second Author
Affiliation / Address line 1
email@domain

Third Author
Affiliation / Address line 1
email@domain

Abstract

Linear programming problems have been a cornerstone of the optimization process, finding applications across diverse industries such as resource allocation, management, networks, power systems, finance, and logistics. Traditional LP solvers like the Simplex method have demonstrably achieved optimal solutions but scale unfavorably with large problem sizes. This survey paper discusses the recent enhancement and impact of integrating graphs into traditional LP solvers mainly the Simplex, Interior-Point, Primal-Dual Hybrid Gradient, and Branch and Bound methods. Our small further experimentation shows some potential for foundational models in the LP space. Lastly, the paper also discusses new emerging concepts of graphs on Operational Research techniques outside of LP.

1 Introduction

Linear Program (LP) has been a fundamental tool in optimization theory where the goal is to maximize or minimize a linear objective function, subject to a set of linear constraints. Consider an LP problem with n variables (including slack) and m constraints. let $c \in \mathbb{R}^n$ be objective coefficient vector, $x \in \mathbb{R}^n$ be variable vector, $A \in \mathbb{R}^{m \times n}$ be constraint coefficient matrix, $b \in \mathbb{R}^m$ be constraint RHS vector, and $B \in \mathbb{R}^m$ be the current basis (ie current solution).

$Objective$

$$\min_{x \in \mathbb{R}^n} \quad c^T x \quad (1)$$

$Such\ that$

$$Ax = b$$
$$x \geq 0$$

When considering all m constraints, the feasible region is shaped in a polyhedron containing all feasible solutions. Developed in 1947, the Dantzig Simplex Method (Nash, 2000) iteratively pivots from one vertex of the polyhedron to an adjacent vertex, improving the objective value at each step. Specifically, in each pivot, a non-basic variable is selected as the entering variable, and a basic variable is selected as the leaving variable. Since the introduction, many different rule-based strategies for initial basis selection and pivots have been proposed. More recently, (Fan et al., 2023) represented LP problems as bipartite graphs and leveraged GNN to make smart initial basis selections. Alternatively, (Liu et al., 2024b) developed new pivoting experts that use local and global information to pivot, then through imitation learning trained a GNN to pivot.

Interior-Point Method (IPM) is another LP algorithm famous for its polynomial timed complexity and was advanced by (Karmarkar, 1984) in the LP space. In this approach, the original LP problem (1) was perturbed by introducing a barrier penalty $\mu > 0$.

$$\min_{x \in R^n} \quad c^T x - \mu[1^T log(b - Ax) + 1^T log(x)] \quad (2)$$

Then, by taking the first-order optimality condition of (2), a system of equations is produced and can be solved algorithmically. (Qian et al., 2023) shows the algorithm for solving this system of equations can be represented as message-passing steps in a tripartite graph and solved faster.

The lesser known Primal-Dual Hybrid Gradient (PDHG) has also gathered attention and has been used to solve large-scale LP problems. The PDHG method considers the Lagrangian form of LP problem (1), whereby the Lagrangian multiplier becomes our dual variable.

$$\mathcal{L}(x, y) = \quad c^T x - y^T Ax + b^T y \quad (3)$$

It then takes the partial derivative of (3) and iteratively improves the solution. (Li et al., 2024) realises the process looks similar to a bipartite graph and took inspiration to develop their own PDHG-network which also shows great improvement in speed.

On another note, Mixed Integer Linear Programming (MILP) uses LP relaxation to recursively solve an integer problem and is strongly related to LP. A commonly used MILP solver is the Branch and Bound Method (B&B) which partitions the original problem into sub-problems and recursively solve them until an integer solution is reached. To speed up the process, (Liu et al., 2024a) attempted to generate useful parameters for B&B's pre-solver while (Gasse et al., 2019) shows a smart learner can make better partitions to terminate B&B method earlier.

We organized this paper with Section 2 - Theoretical Capability of LP on Graphs, Section 3 - Enhancements in the Simplex Method, Section 4 - Advancements in IPMs and PDHG Method, Section 5 - Improvements in MILP, Section 6 - Discussion on Dynamic and Quadratic Programming, Section 7 - Conclusion.

2 Theoretical Capability

With the recent success of LP on graphs, it is important to understand the theory to make logical improvements. The theorem produced by (Chen et al., 2023) concludes that GNNs can reliably predict the feasibility, boundedness, objective value, and optimal solution on a broad class of LP problems. More specifically, GNNs are assessed on two main criteria:

- Separation Power (Section 2.2): GNNs ability to distinguish different LP problems.
- Representation Power (Section 2.3): GNNs ability to represent broad classes of LP problems.

2.1 Preliminaries

An LP problem can be generalized as a bipartite graph with two types of nodes: Variable and Constraint nodes. Edges connecting the nodes are variable coefficients from matrix A. See Figure 1 and 2 for an illustration of node features from (Chen et al., 2023).

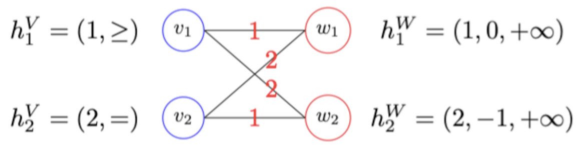
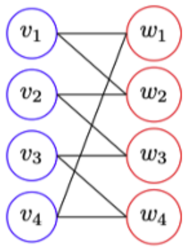
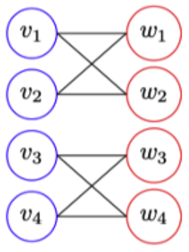


Figure 1: Illustration of the bipartite graph representation for the previous LP problem.



$$\begin{aligned} \min\ & x_1 + x_2 + x_3 + x_4, \\ \text{s.t. } & x_1 + x_2 = 1, \\ & x_2 + x_3 = 1, \\ & x_3 + x_4 = 1, \\ & x_4 + x_1 = 1, \\ & x_j \geq 1,\ 1 \leq j \leq 4. \end{aligned}$$



$$\begin{aligned} \min\ & x_1 + x_2 + x_3 + x_4, \\ \text{s.t. } & x_1 + x_2 = 1, \\ & x_2 + x_1 = 1, \\ & x_3 + x_4 = 1, \\ & x_4 + x_3 = 1, \\ & x_j \geq 1,\ 1 \leq j \leq 4. \end{aligned}$$

Figure 2: Two LP Problems on the right with their bipartite graph representation. The two problems are both infeasible.

With this representation, theoretical mappings ϕ_{feas}, ϕ_{obj}, ϕ_{solu}, should take the graph G, with features H, and outputs the feasibility, objective value, and solution of the LP.

$$\begin{aligned} \phi_{feas}: &\quad (G,H) \to \{0,1\} \\ \phi_{obj}: &\quad (G,H) \to \mathbb{R} \cup \{-\infty, \infty\} \\ \phi_{solu}: &\quad \phi_{solu}^{-1}(\mathbb{R}) \to \mathbb{R}^n \end{aligned}$$

Similarly, two types of GNN can be constructed that maps (G,H) to a predicted objective value or an objective solution.

$$\begin{aligned} F_{GNN}: &\quad (G,H) \to \mathbb{R} \cup \{-\infty, \infty\} \\ F_{GNN}^{W}: &\quad (G,H) \to \mathbb{R}^n \end{aligned}$$

Where $\{-\infty, \infty\}$ illustrates the infeasibility and unboundedness of the LP, respectively.

2.2 Separation Power

Consider the LPs in Figure 2 where both LP are feasible but are different problems. When applying the WL test, regardless of the hash function, the two graphs cannot be distinguished. This means for each iteration $l \in \{0, 1, 2, ...L\}$, the hash result on each nodes are equal in both graphs. Expanding on proposition 2 and 3 of (Berkholz et al., 2015), (Chen et al., 2023) algebraically proves that:

Lemma 1: *If the WL test is unable to distinguish two weighted bipartite graphs that were generated from different LP problems, then both problems are feasible or both are infeasible.* See lemma B.2 of (Chen et al., 2023).

Now consider $C_j^{l,W}$ as the resulting hash function on variable node j at the l^{th} iteration. Consider $h_j^{l,W}$ as the resulting features on variable node j at the l^{th} message-passing step of $F \in F_{GNN}$. Similarly for constraint counterpart $C_i^{l,V}$, $h_i^{l,V}$. Let the symbol $\sim$ be an equivalence relationship, defined as $(G, H) \sim (G', H')$ iff two LP cannot be distinguished by the WL test.

Inductively, if $h_j^{l,W} = h_{j'}^{l,W}$ implies $C_j^{l,W} = C_{j'}^{l,W}$ for all variable node j, j' and if $h_i^{l,V} = h_{i'}^{l,V}$ implies $C_i^{l,V} = C_{i'}^{l,V}$ for all constraint node i, i' then it is sufficient that:

lemma 2: *Given two LP Graphs (G, H) and (G', H'). If $F(G, H) = F(G', H')$ holds for any $F \in F_{GNN}$, then $(G, H) \sim (G', H')$.* See lemma C.3 of (Chen et al., 2023).

The situation in Lemma 1 was also proven true if both LP have the same optimal objective value including $\{-\infty, \infty\}$ for boundedness of the LP. Furthermore, if both LPs have a solution (ie LP is feasible and bounded) then they have the same solution up to some permutation. These together form Theorem 3. Inductively like Lemma 2, the corresponding Theorem 3 can be extended to Theorem 4.

Theorem 3: *If two LP (G, H) and (G', H') are not distinguishable by the WL test, then*

$$\phi_{feas}(G, H) = \phi_{feas}(G', H') \text{ and}$$

$$\phi_{obj}(G, H) = \phi_{obj}(G', H')$$

both holds. Furthmore, if $(G, H), (G', H') \in \phi_{solu}^{-1}(R)$, then for some permutation on variables σ_W, it holds that

$$\phi_{solu}(G, H) = \sigma_W(\phi_{feas}(G', H'))$$

See Theorem 4.1 of (Chen et al., 2023)

Theorem 4: *Given two LP (G, H) and (G', H') if $F(G, H) = F(G', H')$ for all $F \in F_{GNN}$, then both LP problems are either:*

(i) *feasible or infeasible*
(ii) *have the same optimal objective value*
(iii) *if both are feasible and bounded, then they have the same optimal solution with the smallest l_2 normalization up to some permutation σ_W.*

See Theorem 3.1 of (Chen et al., 2023)

Theorem 3 and 4 show that GNN has sufficient separation power and is at least that of the WL test. Hence, this concludes that F_{GNN} and F_{GNN}^W are rich enough to distinguish characteristics of LP problem.

2.3 Representation Power

Representational power refers to the capability to recognize LP problems that are of the same class. Let X be a compact set containing LP graphs (ie class of LP that shares some commonality). Let $C(X, R)$ be collections of algebraic real-valued continuous function on X. Through theorem 22 from (Azizian and Lelarge, 2021), (Chen et al., 2023) shows:

Theorem 5: *For any function $\phi \in C(X, R)$, and for all pair of graphs $(G, H), (G', H') \in X$ that are not distinguishable by the WL test, if ϕ satisfies $\phi(G, H) = \phi(G', H')$, then there exist $F \in F_{GNN}$ such that:*

$$\sup_{(G,H)\in X} |\phi(G, H) - F(G, H)| < \epsilon$$

See Theorem 4.3 and D.1 from (Chen et al., 2023).

However, the mapping ϕ_{feas}, ϕ_{obj}, ϕ_{solu} are not continuous. (Chen et al., 2023) shows that these mapping are measurable through Lemma F.1, F.2, F.2 from their paper. (Chen et al., 2023) also shows Theorem 5 similarly holds for these measurable mapping functions, see Theorem F.4 for the proof from their paper.

This shows when LP graphs that come from a compact set X, there will exist an $F_1 \in F_{GNN}$ that can classify the feasibility and boundedness of the problem up to some difference ε. Similarly, there will exist a $F_2 \in F_{GNN}$, $F_3 \in F_{GNN}^W$ that can approximate the objective value and objective solution, respectively up to some difference ε. Hence, GNN has good classification/predictive capability in regards to representation power.

3 Enhancement on Simplex Method

As previously mentioned, the Simplex Method largely depends on the initial basis and the pivoting method. We organise this section as such:

- Section 3.1: We discuss on a newly developed smart initial basis selection through the use of graphs (Fan et al., 2023).

- Section 3.2: We discuss on a smart pivoting approach that considers both local and global conditions of the current basis through imitation learning on graphs. (Liu et al., 2024b)
- Section 3.3: We conduct further extension on (Fan et al., 2023) and (Liu et al., 2024b) and present an ensemble learning system.

3.1 Smart Initial Basis (SIB)

In practice, many LP problems can originate from the same category of problems and can share substantial similarities and distributions. For example, all maritime routing LP problems comes from the same abstract model which is the Ocean. (Fan et al., 2023) takes motivation on this and exploits these underlying patterns to construct a smart initial basis.

3.1.1 Notation For SIB

Refer to the first paragraph of Section 1 but instead of b, let $s \in \mathbb{R}^m$ be the RHS objective coefficient. Each variable x_i contains a lower bound l_i^x and upper bound u_i^x, which can be $-\infty$ and ∞. let B_x, B_s be the set of primal and slack variables in our basis, respectively. Finally, let v_i^l, w_i^l be the message passing result for variable i and constraint j in the $l^{th} \in \{0, 1, ...L\}$ message-passing of the GNN.

3.1.2 SIB Overview and Result

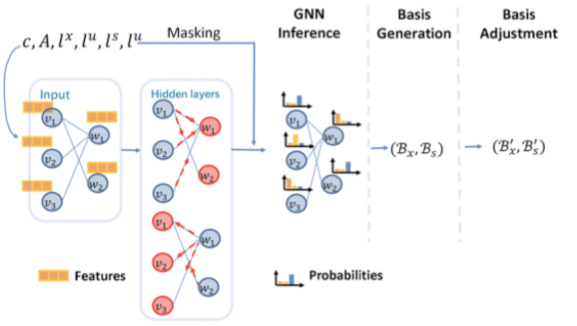


Figure 3: Process of the Smart Initial Basis.

Step 1 in Figure 3 is to represent the LP problems as a bipartite graph which is the same as in section 2.1 except with different features. In step 2, the bipartite graph is passed onto a standard GCN with two forward passing functions: one from variable to constraint, the other from constraint to variable.

In Step 3, to predict the optimal basis, for each variable, the probability of its state is required. More specifically, if a variable is at its lower or upper bound then the state is non-basic, and if the variable is between its lower and upper bound then the state is basic. Thus, for each variable x_i the output of the GCN should be:

$$p_{x_i} = \begin{bmatrix} \mathbb{P}(x_i = l_i^x) \\ \mathbb{P}(l_i^x < x_i < u_i^x) \\ \mathbb{P}(x_i = u_i^x) \end{bmatrix}$$

However, if l_i^x for x_i is $-\infty$, then $\mathbb{P}(x_i = l_i^x) = 0$. To ensure this happens, the GCN output is then masked to $p_{x_i} = softmax(v_i^L + h_{x,i})$ where:

$$h_{x,i} = \begin{cases} [-\infty \ \ 0 \ \ 0]^T & l_i^x = -\infty \\ [0 \ \ 0 - \infty]^T & u_i^x = \infty \\ [0 \ \ 0 \ \ 0]^T & otherwise \end{cases}$$

Step 4 takes the first m highest probability in $p_{x_i}[1]$ and set them to 1 (basic) while the rest are set to 0 (non-basic). However, this resulting $A_{(B_x, B_s)}$ may be singular, hence the basis adjustment in step 5 was introduced. This utilises LU factorisation technique from (Bixby, 1992) and (Bixby and Saltzman, 1994) whereby if a column in our basis, (B_x, B_s), does not have any entry bigger than a small number τ, then this column does not have a valid pivot and is removed. The next variable with the highest $p_{x_i}[1]$ enters and the adjustment process repeats until $A_{(B_x, B_s)}$ is non-singular.

However, the initialization is always close to the optimal solution but not the optimal. Hence, a Simplex solver like HiGHS or OptVerse was used at the end of step 5. As the initial basis is close to optimal, the Simplex Solver should terminate quickly.

(Fan et al., 2023) compares 4 baselines to their developed framework of SIB + (HiGHS, OptVerse) with 6 different datasets. Across the 6 dataset, the developed framework is always the fastest. When compared to the fastest baseline in each test, on average the framework is 56.4% faster in time (seconds). Furthermore, the accuracy, precision, and recall of SIB results compared to the optimal value were consistently 78% - 99%. However, when training and testing the framework in different dataset, the framework tends to yield results worse than the default baseline, up to 740% worse.

3.2 Learn to Pivot (LTP)

Many heuristic pivot rules only look at local information around the current basis such as Blands, Dantzig, Steepest-edge, and Greatest Improvement rules. This occurs because global information is difficult to incorporate in practice. (Liu et al., 2024b) creates a new pivoting expert that incorporates both global and local information. In practice, this expert is difficult to implement however, it can be used to guide an imitation learner which is more practical.

3.2.1 Pivoting Expert

One global information that is often missed is the optimal solution. One can see why this is impractical on its own, however (Liu et al., 2024b) uses this guide their expert. Specifically:

1. When selecting an entering variable, a smart pivot should let the basic variable in the optimal basis enter first.
2. When selecting a leaving variable, a smart pivot rule should let the non-basic variable in the optimal basis leave first.

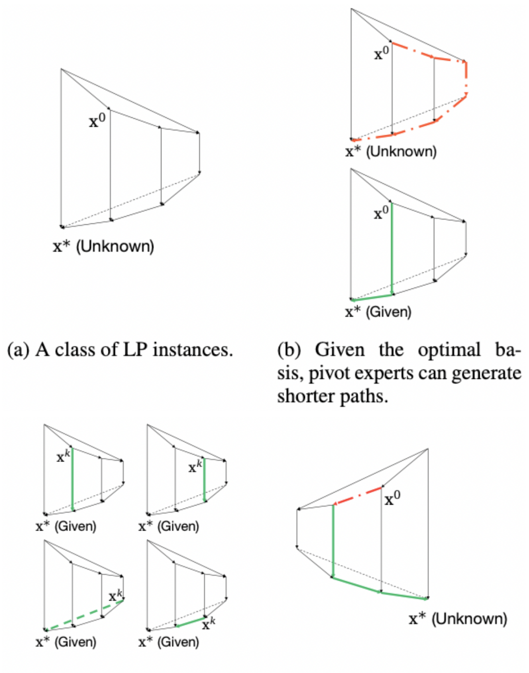
However, (Liu et al., 2024b) realized a good pivoting rule should also include local information. In fact, in their experimentation, they show an expert without local information performs worse than a rule with only local information. Two experts were developed by (Liu et al., 2024b) but this survey paper will only discuss one:

Expert-1:

- When selecting an entering variable amongst the optimal basis (Global information), the expert should assign the one with the steepest-edge score (Local information) from the optimal basis.
- After the ratio test, the expert should remove the non-basic variable in the optimal basis (Global information) with the smallest ratio (Local information).

3.2.2 LTP Overview and Results

Notice the expert's choice can be represented as a Markov decision process whereby in the l^{th} iteration, the state s_l contains the LP instance and current basis x^l. The action space $A(s_l)$ is a set of pairs, ie (entering variable, leaving variable) which the expert can choose. The chosen action is the label for the imitation learner.



(a) A class of LP instances. (b) Given the optimal basis, pivot experts can generate shorter paths.

(c) Collect expert choices for imitation. (d) Learned pivot rule gains improvement on new LP instances.

Figure 4: Illustration of the model imitating the expert's choice. The green lines are the expert's choice whereas the red lines are bad choices. Read in the order of (a) to (d).

(Liu et al., 2024b) feeds a bipartite graph into GCNN model and trains the model as described in Figure 4. A final softmax function is used to estimate the probability of a variable entering or leaving the basis.

(Liu et al., 2024b) used 4 datasets to compare the LTP model to 3 baselines that use local information. Compared to the baseline with the least pivot in each test, on average the LTP model uses 90% less pivots and consistently uses the least pivot across all test. However, for time (seconds) the LTP model performs slower due to GCNN's forward passes. (Liu et al., 2024b) states without experimentation that a meticulous design should further optimize this time.

3.3 Ensemble Model

[1] We make an initial attempt at an ensemble model for the Simplex Method. 1000 LP feasible instances with random coefficients and equality signs were generated as our dataset. 2 SIB were used as the weak learner, one for primal and another for slack variables. Unfortunately (Liu et al., 2024b) did not provide their code, we tried our best to replicate their paper. For LTP training, we

[1]https://github.com/boomer3boom/Exploring-Linear-Programs-with-Graphs-and-AI

infer the initial basis using the weak learners, collect the expert's choice, and train the LTP.
The result shows the SIB performed well with an average 70% accuracy across 200 instances. However, the LTP performed poorly and only improved the SIB 2 out of 200 times. The average accuracy after LTP decreased to 69.1%.

4 Other LP Algorithm

Aside from the Simplex Method, other popular LP method includes IPMs and PDHG, which have both gathered some attention in the Graph AI space. This section is organized into:

- Section 4.1: We discuss on a tri-partite graph approach by (Qian et al., 2023) to simulate IPMs.
- Section 4.2: We discuss a bipartite graph inspired neural network called PDHG-network from (Li et al., 2024) for large-scale LP.

4.1 Enhancement on IPMs

As previously explained, IPMs move the LP constraints to the objective with a barrier term, see equation 2. From equation 2 the first derivative of Lagrangian is:

$$\frac{\partial \mathcal{L}}{\partial x} = c - \mu \sum_{j \in [m]} \frac{A_{j,*}}{A_{j,*}x - b_j} - \mu \sum_{i \in [n]} \frac{1}{x_i}$$

Setting $s_i = \mu / x_i, r_j = A_{j,*}x - b_j$, and $w_j = mu/r_j$, a system of equations is produced, see below. Note that s_i uses the i^{th} variable, while w_j, r_j uses the j^{th} constraint.

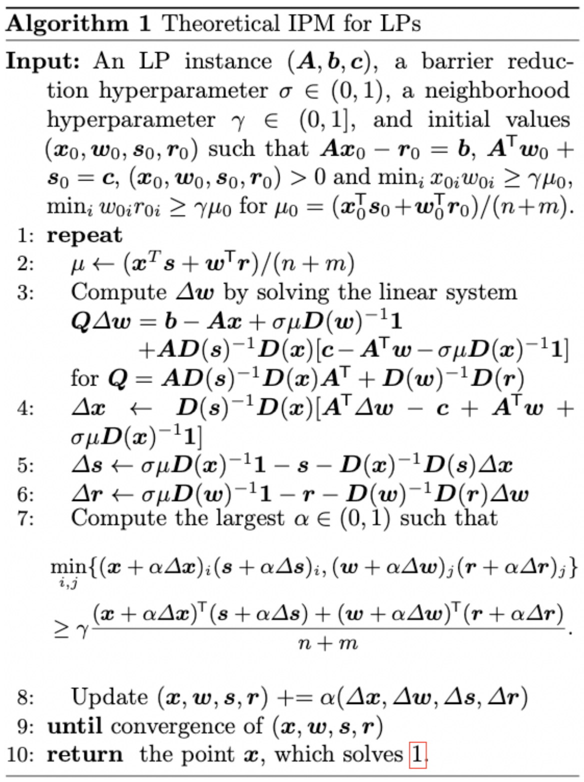
$$\begin{aligned} Ax - r =& \quad b \\ A^T w + s =& \quad c \\ x_i s_i =& \quad \mu \\ w_j r_j =& \quad \mu \end{aligned}$$

This system of equations can be solved iteratively with Newton Method. However, for each step the barrier is reduced by $\sigma \in (0, 1)$:

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

Where 1 is a vector of ones, and D is the diagonal matrix. Through expansion, the Newton Method can be expressed as four equations that can be solved sequentially (see line 3, 4, 5, 6) as demonstrated in Algorithm 1 by (Nocedal and Wright, 2006) and displayed by (Qian et al., 2023).

Consider a LP tripartite graph with three types of node: Variable, Constraint, and Objective nodes. Realize the tripartite has 4 actions: message-passing between Variable-Constraint, Constraint-Objective, Objective-Variable, and local operations. (Qian et al., 2023) shows that each operation in Algorithm 1 can be perfectly replicated as one of the 4 actions. For example in line 2, $x^T s \rightarrow h_1$ is a local operation on variable nodes then passed to the objective node, $w^T r \rightarrow h_2$ is a local operation on constraint node then passed to the objective node, finally $\frac{h_1+h_2}{n+m}$ is a local operation on the objective node.



Algorithm 1 Theoretical IPM for LPs

```
Input: An LP instance $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$, a barrier reduction hyperparameter $\sigma \in (0,1)$, a neighborhood
    hyperparameter $\gamma \in (0,1]$, and initial values $(\boldsymbol{x}_0, \boldsymbol{w}_0, \boldsymbol{s}_0, \boldsymbol{r}_0)$ such that $\boldsymbol{A}\boldsymbol{x}_0 - \boldsymbol{r}_0 = \boldsymbol{b}$, $\boldsymbol{A}^\top \boldsymbol{w}_0 + \boldsymbol{s}_0 = \boldsymbol{c}$,
    $(\boldsymbol{x}_0, \boldsymbol{w}_0, \boldsymbol{s}_0, \boldsymbol{r}_0) > 0$ and $\min_i x_{0i} w_{0i} \geq \gamma\mu_0$, $\min_i w_{0i} r_{0i} \geq \gamma\mu_0$ for $\mu_0 = (\boldsymbol{x}_0^\top \boldsymbol{s}_0 + \boldsymbol{w}_0^\top \boldsymbol{r}_0)/(n+m)$.
 1: repeat
 2:     $\mu \leftarrow (\boldsymbol{x}^T \boldsymbol{s} + \boldsymbol{w}^\top \boldsymbol{r})/(n+m)$
 3:     Compute $\Delta\boldsymbol{w}$ by solving the linear system
        $\boldsymbol{Q}\Delta\boldsymbol{w} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x} + \sigma\mu\boldsymbol{D}(\boldsymbol{w})^{-1}\boldsymbol{1}$
                $+\boldsymbol{A}\boldsymbol{D}(\boldsymbol{s})^{-1}\boldsymbol{D}(\boldsymbol{x})[\boldsymbol{c} - \boldsymbol{A}^\top \boldsymbol{w} - \sigma\mu\boldsymbol{D}(\boldsymbol{x})^{-1}\boldsymbol{1}]$
        for $\boldsymbol{Q} = \boldsymbol{A}\boldsymbol{D}(\boldsymbol{s})^{-1}\boldsymbol{D}(\boldsymbol{x})\boldsymbol{A}^\top + \boldsymbol{D}(\boldsymbol{w})^{-1}\boldsymbol{D}(\boldsymbol{r})$
 4:     $\Delta\boldsymbol{x} \leftarrow \boldsymbol{D}(\boldsymbol{s})^{-1}\boldsymbol{D}(\boldsymbol{x})[\boldsymbol{A}^\top \Delta\boldsymbol{w} - \boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{w} + \sigma\mu\boldsymbol{D}(\boldsymbol{x})^{-1}\boldsymbol{1}]$
 5:     $\Delta\boldsymbol{s} \leftarrow \sigma\mu\boldsymbol{D}(\boldsymbol{x})^{-1}\boldsymbol{1} - \boldsymbol{s} - \boldsymbol{D}(\boldsymbol{x})^{-1}\boldsymbol{D}(\boldsymbol{s})\Delta\boldsymbol{x}$
 6:     $\Delta\boldsymbol{r} \leftarrow \sigma\mu\boldsymbol{D}(\boldsymbol{w})^{-1}\boldsymbol{1} - \boldsymbol{r} - \boldsymbol{D}(\boldsymbol{w})^{-1}\boldsymbol{D}(\boldsymbol{r})\Delta\boldsymbol{w}$
 7:     Compute the largest $\alpha \in (0,1)$ such that
        $\min_{i,j}\{(\boldsymbol{x} + \alpha\Delta\boldsymbol{x})_i(\boldsymbol{s} + \alpha\Delta\boldsymbol{s})_i, (\boldsymbol{w} + \alpha\Delta\boldsymbol{w})_j(\boldsymbol{r} + \alpha\Delta\boldsymbol{r})_j\}$
        $\geq \gamma \frac{(\boldsymbol{x} + \alpha\Delta\boldsymbol{x})^\top(\boldsymbol{s} + \alpha\Delta\boldsymbol{s}) + (\boldsymbol{w} + \alpha\Delta\boldsymbol{w})^\top(\boldsymbol{r} + \alpha\Delta\boldsymbol{r})}{n+m}.$
 8:     Update $(\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{s}, \boldsymbol{r})$ += $\alpha(\Delta\boldsymbol{x}, \Delta\boldsymbol{w}, \Delta\boldsymbol{s}, \Delta\boldsymbol{r})$
 9: until convergence of $(\boldsymbol{x}, \boldsymbol{w}, \boldsymbol{s}, \boldsymbol{r})$
10: return the point $\boldsymbol{x}$, which solves 1.
```

Theorem 6*: There exist a Message-Passing Neural Network f with O(k) message-passing steps to replicate an iteration of Algorithm 1. Thus, given an LP instance* $L = (A, b, c)$ *where variable node carries* $[x_l, s_l]$ *features and constraint node carries* $[w_l, r_l]$ *features in the* l^{th} *iteration, GNN can map these variable and* *constraint features to* $[x_{l+1}, s_{l+1}]$ $[w_{l+1}, r_{l+1}]$. *See Theorem 4 of (Qian et al., 2023).*

(Qian et al., 2023) trains their GNN by updating the constraint nodes first, then objective, then variable, and then gets the result from variable nodes. See Figure 5.

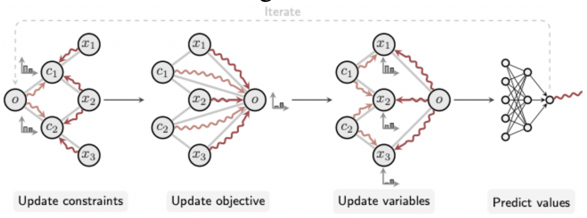


Figure 5: Illustration of a Message-Passing Neural Network in a tripartite graph for IPMs. (note that c_1 and c_2 is to display as constraint node, not the objective coefficient. In all other cases, please interpret c_i as objective coefficient.)

(Qian et al., 2023) uses 4 datasets that are known to be NP-hard and train their IPMs-MPNN. When compared to Scipy's solver on small instances, across the 4 datasets SciPy's average time was 0.011s while IPM-MPNNs performed 0.021s. However, on large instances, SciPy's average time was 0.41s while IPM-MPNNs was 0.0212s. IPM-MPNN time was more predictable showing around 1.001x slowdown from small to large instances while Scipy's experienced 37.27x slowdown.

The objective gap and constraint violation were consistently between $1 - 0.1\%$ and $0.9 - 0.03\%$, respectively. However, when training IPMs-MPNN with smaller LP problems and testing on larger LP problems, the result becomes sub-optimal with objective gaps lying between $0.9\% - 14\%$ and the constraint violation lies between $10\% - 59\%$.

We trained the GCN model from (Qian et al., 2023) on a dataset with all 4 problems. When testing, the mean objective gap and constraint violation were 1.55% and 0.29%, respectively. The test on each problem is recorded in Table 1.

Metric	Setcover	Cauction	Facilities	Cauctions
Objective Gap %	1.65	1.6	1.71	1.73
Constraint Violation %	0.06	0.14	0.07	0.19

Table 1: Result of training GCN on all a dataset and testing individually.

Although the objective gap is not comparable to individual GCNs trained on individual problem types, this result still shows some potential for foundational models to be trained across all datasets.

4.2 Enhancement on PDHG

In an LP problem, the original problem is referred to as the primal, and the dual originates from the primal. Specifically, the primal constraint becomes the dual variable, the primal variable becomes the dual constraint, and the Primal objective coefficient becomes the dual constraint RHS. Also if primal aim is to minimise then the dual aim is to maximise. More importantly, by the Duality Theorem, the optimal solution is equal $c^T x^* = b^T y^*$. Using this, the PDHG considers the Lagrangian of equation (1) and iteratively updates the primal and dual variables with small steps τ and σ until both are optimal.

$$\frac{\partial \mathcal{L}(x,y)}{\partial x} = c - y^T A \rightarrow x^{k+1} = x^k - \tau \frac{\partial \mathcal{L}(x,y)}{\partial x}$$

$$\frac{\partial \mathcal{L}(x,y)}{\partial y} = b^T - Ax \rightarrow y^{k+1} = y^k + \sigma \frac{\partial \mathcal{L}(x,y)}{\partial y}$$

In PDHG, the primal is updated first x^{k+1}, then when updating the dual y^{k+1} a combination of x^{k+1} and x^k is used. To ensure x^{k+1} and y^{k+1} stays within bound, a piece-wise linear function $proj$ is added:

Algorithm 1 PDHG Iteration

```
Require: An LP instance $(A, b, c)$, with an initial
    $x^0$ and $y^0$.
1: for $k = 0, 1, 2, \ldots, K-1$ do
2:     $x^{k+1} \leftarrow \text{proj}_x(x^k - \tau(c - y^T A))$
3:     $y^{k+1} \leftarrow \text{proj}_y(y^k - \sigma(b - 2Ax^{k+1} + Ax^k))$
4: end for
5: return $x^{K-1}$ and $y^{K-1}$
```

Realize that each iteration of PDHG can be viewed as a bipartite graph of primal and dual variables, see Figure 6. This is the same as section 2.1 since the dual variables are the constraints of the primal. Inspired by this, (Li et al., 2024) unroll this single iteration into a network, and stacks these sequentially as seen in figure 6 and 7. $\theta_p^k = [\tau_k, U_x^k, U_y^k], \theta_d^k = [\sigma_k, V_x^k, V_y^k, W_x^k]$ are learnable parameters, where $U_x^k, U_y^k, V_x^k, V_y^k, W_x^k$ allow size generalizability of primal and dual variable into the network. d_k is the number of channels in the k^{th} layer.

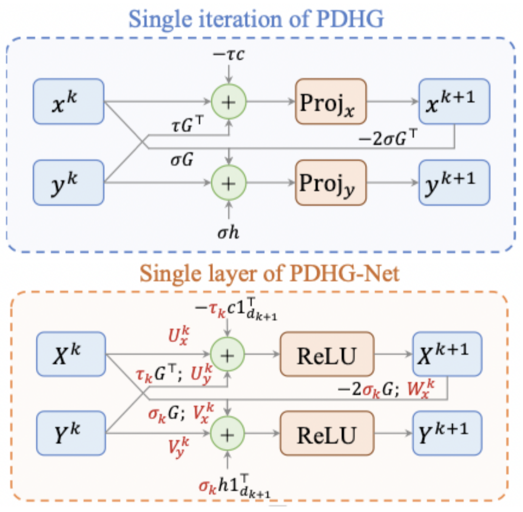
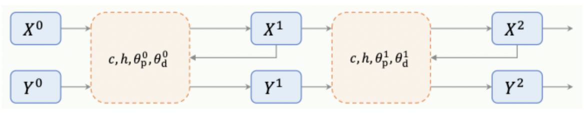


Figure 6: Graphical representation of PDHG iteration and a single layer of developed PDHG-network both retrieved from (Li et al., 2024).



X^0
Y^0
$c, h, \theta_p^0, \theta_d^0$
X^1
Y^1
$c, h, \theta_p^1, \theta_d^1$
X^2
Y^2

Figure 7: Example of stacked PDHG-network. Image retrieved from (Li et al., 2024).

(Li et al., 2024) shows that provided there are sufficient channels, d_k, in the k^{th} layer, the PDHG-network can re-iterate the entire PDHG process. This is difficult to achieve in practice as the required d_k is different for each problem. Instead, (Li et al., 2024) addes a PDLP solver at the end of their framework to ensure optimality. The PDLP solver terminates quickly as the provided solution is close to optimal.

(Li et al., 2024) experimentation shows that compared to standard PDLP solvers, the proposed framework has around 45% and 73% speed improvement for small and large datasets, respectively, on linearised Google pagerank problems. However, in large problems from Item-Placement and Work-Approxmiation dataset, the improvement is only 21.3% and 4.4%. This suggest the improvement fluctuates depending on different problems.

5 MILP

MILP is a crucial aspect of LP relating to many real-world problems. In this section, we will discuss the GNN enhancement on the Branch and Bound (B&B) method often used to solve MILP. At its core, B&B recursively splits the solution space into a decision-making tree and solves the relaxed LP in each node until an integer solution is computed. If a node's relaxed LP solution are not integers, B&B splits the feasible region by choosing a variable that is not an integer.

$$x_i \leq \lfloor x_i^* \rfloor \vee x_i \geq \lceil x_i^* \rceil, x*_i \notin \mathbb{Z}$$

The section is arranged as:

- Section 5.1: To efficiently solve MILP using B&B, the problem is often pre-solved to simplify and reduce the size of the problem before passing it onto a B&B solver. (Liu et al., 2024a) demonstrates initial success on their hybrid algorithmic neural network targeted for instance-specific presolving.
- Section 5.2: An important process in B&B is deciding how to partition the problem (ie which variable to select) also known as branching. Strong branching is a technique that produces good branching decisions but introduces further computational costs. (Gasse et al., 2019) reduces the computational cost of strong branching through imitation learning on GCNN.

5.1 Learning to Presolve (L2P)

To remove redundant constraints, strengthen the linear programming relaxation, and extract useful information from the problem, the presolving process is usually sequentially stacked with presolvers taking the output of the previous as input. To generate good presolving process, there are three key parameters: (1) The priority of execution for each pre-solver, (2) the max-round the pre-solver should participate in, and (3) the amount of time the pre-solver runs for. Existing solvers have these three parameters set by default no matter the LP instance. (Liu et al., 2024a) recognises that simulated annealing (SA) is a potential heuristic algorithm to generate good parameters but with heavy computation and time. Hence through imitation learning, they train a GCNN to infer the three parameters for any MILP instance.

As the author of this paper believes the priority parameter is more important, they train the first GCNN to minimise the loss on the priority label. Treating the first GCNN as a weak learner and using the priority knowledge, two other GCNNs fine-tune the parameters for max-round and timing. This is illustrated in figure 8.

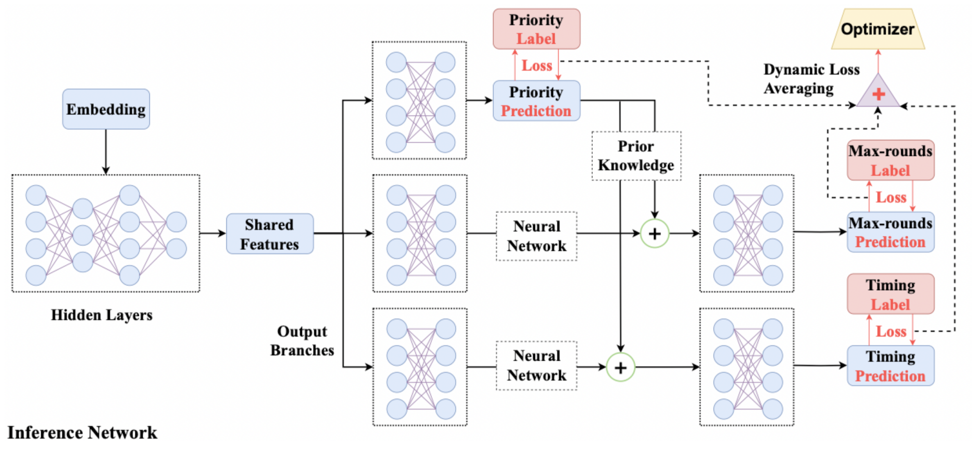


Figure 8: Inferenceing network for the GCNN that is to predict the parameters Priority, Max-round, and Timing for a pool of presolvers by (Liu et al., 2024a).

To evaluate the model, (Liu et al., 2024a) runs 5 other algorithms as baseline including SA and a default parameter that does not change. Using the generated parameters, a B&B solver attempts to solve the MILP instances. In the easy instances, the baseline and the model showed no differences, in the medium difficulty the model performed on average 18.88% faster across two datasets, beating most other baselines but SA who performed 26.24% faster on average. In the large instances, (Liu et al., 2024a) states the SA baseline took days to generate parameters while the model took a few seconds.

Further experiment from (Liu et al., 2024a) shows the developed model when trained on small instances can generalise to medium and large instances. However, larger instances will deteriorate the performance.

5.2 Enhancement on B&B

Strong Branching works by calculating a score s_i for all variables that are not integers at a relaxed LP node, and then chooses the variable with the highest score to split. Different MILP solvers may calculate the score differently, one common approach is to calculate the expected lower bound improvement for each variable candidate. This requires heavy computation as it needs the solution of the relaxed LP to calculate the expected improvement.

(Gasse et al., 2019) attempts to quicken this process through GCNN. Similar to section 2.1, (Gasse et al., 2019) expresses the LP as a bipartite graph and encodes the strong branching as a markov decision process similar to section 3.2.2.

The GCNN was structured through two interleaved half-convolution message-passing, see Figure 9.

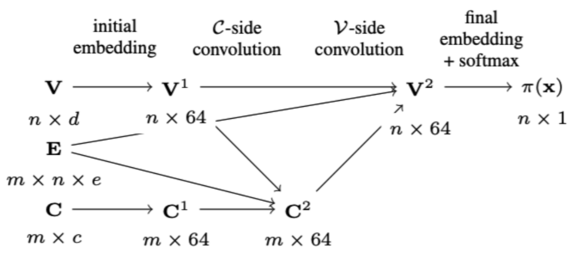


Figure 9: (Gasse et al., 2019) GCNN structure to imitate strong branching.

(Gasse et al., 2019) experiments the developed model on 4 datasets: Set Covering, Combinatorial Auction, Capacitated Facility Location, and Maximum Independent Set. The metrics used to evaluate the performance includes: the total solving time, the number of wins across 100 attempts, and the number of nodes required to solve the LP problem. For each attempt, the fastest baseline is awarded 1 win. Results from (Gasse et al., 2019) clearly show their developed GCNN model dominates in almost all metrics across each test. Furthermore, their model's result solution had the highest accuracy to the optimal solution.

6 Future Directions

One natural question moving on from LP is QP (Quadratic Programming). Indeed, theoretical progress has recently emerged from (Chen et al., 2024) for Linearly Constraint QP (LCQP) and Mixed-Integer QP (MIQP). They provide an affirmatively positive answer with GNN on LCQP. Unfortunately, (Chen et al., 2024) states that GNN cannot universally make predictions for MI-LCQP. However, they defined precise subclasses of MI-LCQP in which GNN can accurately make predictions for the problem's feasibility, boundedness, and optimal solutions.

Considering the success of LP, another natural question is the effect of graphs on DP (Dynamic Programming) which is a general problem-solving technique that can also solve MILP. (Dudzik and Veličković, 2022) derives a generic diagram that connects DP with GNN. They also demonstrated how this works for the Bell-Man Ford algorithm and provided experimentation on other popular algorithms.

7 Conclusion

In summary, this paper explores the recent success of LP in the Graph and AI domain. Theoretically, we see that GNN with their message-passing attribute makes them an excellent choice to speed up the process for LP methods. This theory is backed by many experiments for different solving methods, specifically the Simplex, IPMs, PDHG, and B&B methods which are discussed in this paper. One common flaw we see throughout these experiments is that in practice, GNN is unable to make 100% optimal basis predictions for all LP problems. However, including a solver in the framework can quickly bring the solution to optimality. Our experimentation mentioned at the end of section 4.1 shows small success on Graph Foundation Models for the LP space and would be a potential research direction to investigate. Two questions moving forward from LP include graphs on QP and DP which we see initial success occurring.

Acknowledgments

I would like to thank Ruihong Qiu for supervising and guiding me throughout the research project and for always being available in person and online for consultation. I would also like to thank Helen Huang for providing this amazing opportunity to conduct research with her team. Lastly, I would like to thank UQ staff on this winter research project and MATH3202 teaching team for sparking my initial interest in Operational Research.

References

Azizian, W. and Lelarge, M. (2021). Expressive power of invariant and equivariant graph neural networks.

Berkholz, C., Bonsma, P., and Grohe, M. (2015). Tight lower and upper bounds for the complexity of canonical colour refinement.

Bixby, R. E. (1992). Implementing the simplex method: The initial basis. *INFORMS J. Comput.*, 4:267–284.

Bixby, R. E. and Saltzman, M. J. (1994). Recovering an optimal lp basis from an interior point solution. *Oper. Res. Lett.*, 15:169–178.

Chen, Z., Chen, X., Liu, J., Wang, X., and Yin, W. (2024). Expressive power of graph neural networks for (mixed-integer) quadratic programs.

Chen, Z., Liu, J., Wang, X., Lu, J., and Yin, W. (2023). On representing linear programs by graph neural networks.

Dudzik, A. and Veličković, P. (2022). Graph neural networks are dynamic programmers.

Fan, Z., Wang, X., Yakovenko, O., Sivas, A. A., Ren, O., Zhang, Y., and Zhou, Z. (2023). Smart initial basis selection for linear programs. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J., editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 9650–9664. PMLR.

Gasse, M., Chételat, D., Ferroni, N., Charlin, L., and Lodi, A. (2019). Exact combinatorial optimization with graph convolutional neural networks.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395.

Li, B., Yang, L., Chen, Y., Wang, S., Chen, Q., Mao, H., Ma, Y., Wang, A., Ding, T., Tang, J., and Sun, R. (2024). Pdhg-unrolled learning-to-optimize method for large-scale linear programming.

Liu, C., Dong, Z., Ma, H., Luo, W., Li, X., Pang, B., Zeng, J., and Yan, J. (2024a). L2p-MIP: Learning to presolve for mixed integer programming. In *The Twelfth International Conference on Learning Representations*.

Liu, T., Pu, S., Ge, D., and Ye, Y. (2024b). Learning to pivot as a smart expert. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(8):8073–8081.

Nash, J. C. (2000). The (dantzig) simplex method for linear programming. *Computing in Science & Engineering*, 2(1):29–31.

Nocedal, J. and Wright, S. (2006). *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York.

Qian, C., Chételat, D., and Morris, C. (2023). Exploring the power of graph neural networks in solving linear optimization problems.