Exploring the Potential of Graph Neural Networks-based Methods for General Linear Programs

Yung-Cheng Chuang
s4745141@uq.edu.au
The University of Queensland
Brisbane, Queensland, Australia

Ruihong Qiu
r.qiu@uq.edu.au
The University of Queensland
Brisbane, Queensland, Australia

Abstract

Linear Programming (LP) and Integer Programming (IP) problems have been pivotal in optimization processes, finding applications across diverse industries like resource management, finance, and logistics. It is also used for internet network traffic optimization. Traditionally, LP solvers have consistently provided optimal solutions for problems formulated linearly. However, with more data and larger problems to solve, these LP solvers become computationally heavy and time-consuming. One solution is to provide better close-to-optimal initial solutions for these solvers, which previously many heuristics have attempted. Recently, researchers have successfully trained Graph Neural Networks to infer close-to-optimal initial solutions as a quick start for specific LP/IP problem types. Instead, this paper explores the potential of GNN models to predict close-to-optimal initial solutions for diverse problems. Specifically, the Smart Initial Basis (SIB), Learn To Pivot (LTP), and Interior-Point Message-passing GNN (IPMGNN) were investigated. Furthermore, an interactive website was developed releasing these models for readers to experiment with. A video demonstration can be accessed here, the repository for the website is open-sourced here, and the model learning code is open-sourced here.

CCS Concepts

• **Applied computing** → **Operations research**.

Keywords

Graph Neural Network; Linear Programs; Optimisation

ACM Reference Format:
Yung-Cheng Chuang and Ruihong Qiu. 2025. Exploring the Potential of Graph Neural Networks-based Methods for General Linear Programs. In *Companion Proceedings of the ACM Web Conference 2025 (WWW Companion '25), April 28-May 2, 2025, Sydney, NSW, Australia.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3701716.3715174

1 Introduction

Linear Program (LP) is a technique that provides efficient and optimized solutions for problems with linear objective functions and constraints. An LP solves a problem by exploring a polyhedron containing the feasible region, defined by the problem's constraint. The Simplex Method [10] iteratively selects an entry and exit variable to pivot from one vertex of the polyhedron to an adjacent vertex, until optimality is reached. Alternatively, the Interior-Point Method (IPM) [6], is known for its polynomial-time complexity as it perturbs the original problem into a barrier-penalized objective function, and the resulting system of equations is solved with the first-order optimality conditions. However, the Simplex Method and IPM exhibit performance limitations when scaling to larger problems; particularly when placed with an unfavorable basis, the Simplex Method requires an exponential time [7]. Furthermore, Mixed Integer Linear Programs (MILP) are linear problems with integer and linear variables that exist more commonly in the real world. However, MILP relies on the Branch and Bound (B&B) methodology which recursively partitions the problem by branching on non-integer variables and solving LP relaxations at each step. It relies on "good" branching decisions and the underlying LP solver to terminate early. To address the complexity of MILP problems, several advanced frameworks, including Bender's Decomposition [12], Column Generation [13], Branch and Price/Cut, and heuristics have been developed to accelerate the solving process. These methods significantly speed up specific problem types, but rely on the efficiency of underlying LP solvers and are not universally applicable across problems.

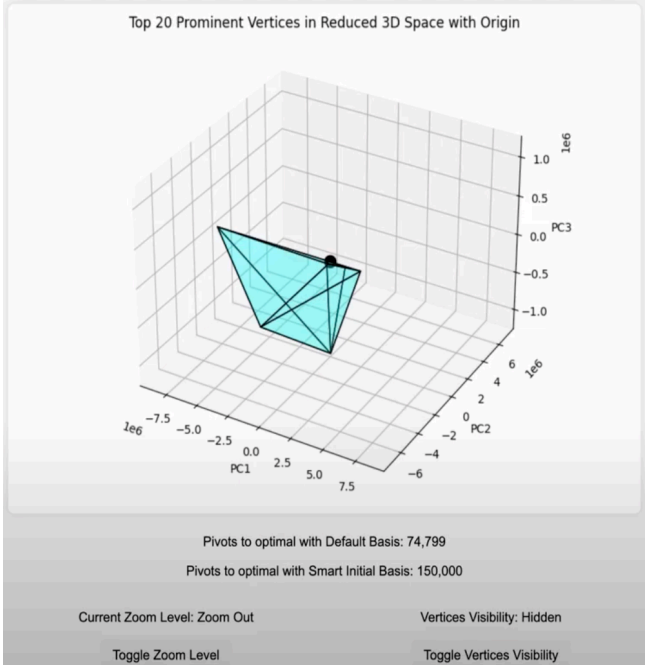

Figure 1: A visualisation of a solved programming problem.

Advancements in research have focused on employing Graph Neural Networks (GNNs) to replicate traditional linear program (LP) solvers as a quick-start approach. Theoretically, GNN have

WWW Companion '25, April 28-May 2, 2025, Sydney, NSW, Australia

ACM ISBN 979-8-4007-1331-6/2025/04
https://doi.org/10.1145/3701716.3715174

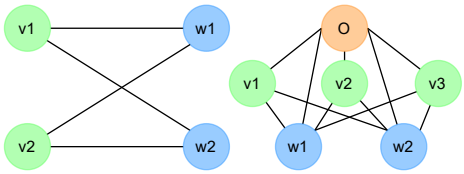

Figure 2: Illustration of the bi/tri-partite graph representation on an LP problem inspired SIB [4] and IPMGNN [11].

been proven to possess sufficient Separation (ability to distinguish problems) and Representational (ability to represent a broad class of problems) Power for LP [3]. To train the GNN, SIB [4] and LTP [9] transform the LP into a bipartite graph while IPMGNN [11] transform the LP into a tripartite graph as shown in Figure 2. Similarly, the L2P method [8] optimizes pre-solver parameters for B&B, while the Strong Branching rule can be imitated with GNN to make fast and effective decisions for the B&B [5].

However, existing approaches primarily focused on training models tailored to specific problem instances, which results in high performance only on the type of problems they were trained on. This limits their effectiveness in assisting general-purpose solvers like Gurobi and CPLEX, which are used to solve a range of problems. This challenge forms the motivation for this paper. Improving solver efficiency requires models that generalize across any problems and reliably provide close to optimal solutions.

This paper presents an initial attempt to replicate the SIB [4], LTP [9], and IPMGNN model [11], but aimed at providing near-optimal starting solutions for any problems. Specifically, random LP instances were generated to train the SIB model, which was then treated as a weak learner for LTP. The LTP model pivots the starting basis from SIB toward optimality. Meanwhile, the IPMGNN model was trained on four distinct IP problems—Set Cover, Maximal Independent Set, Combinatorial Auction, and Capacitated Facility Location—and tested on all problem types together. All models are available on the website for readers to explore and experiment with.

2 Preliminary

In the following, a lowercase letter represents a vector of scalars, a bold lowercase represents a vector of variables, a bold uppercase represents a matrix, and a scripted uppercase represents a set. For example $h, \boldsymbol{h}, \boldsymbol{X}, \mathcal{D}$. Consider an LP problem with n variables (including slack) and m constraints. let $c \in \mathbb{R}^n$ be objective coefficient vector, $\boldsymbol{x} \in \mathbb{R}^n$ be variable vector, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ be constraint coefficient matrix, $b \in \mathbb{R}^m$ be constraint RHS vector, and $\mathcal{B} \subseteq \{1, 2, ..., n\}$ be the current basis (i.e., current solution).

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

such that:

$$\begin{aligned} Ax &= b \\ x &\geq 0 \end{aligned}$$

This 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 $\boldsymbol{A}$. SIB [4] and LTP [9] used this bipartite representation to train their GNN model. In the bipartite graph, v represents the variable nodes and w represents the constraint nodes. The edges between each node corresponds to $\boldsymbol{A}$, specifically the scalar $\boldsymbol{A}_{w,v}$. Similarly, the problem can be transposed into a tripartite graph with three nodes: Variable, Constraint, and Objective. Variable to Constraint edges holds the coefficient scalar from $\boldsymbol{A}$, Variable to Objective edges represents the objective scalar c_v and Constraint to Objective edges represents the right-hand side scalar value b_w. IPMGNN [11] uses this structure to train their IPMGNN model. See Figure 2.

Table 1: Features in bipartite graph

	Variable Node	Constraint Node
1	Objective coefficient $= c_i$	Constraint RHS scaler $= b_j$
2	ratio of number of non-zero $= \text{nnz}(A_{*,i})/\text{num\_con}$	ratio of number of non-zero $= \text{nnx}(A_{j,*})/\text{num\_var}$

Table 2: Problem size for training IPMGNN

Problem	Setcover		Independent Set	
	Rows	Columns	Nodes	Affinity
Size	700	700	700	2
Problem	**Aution**		**Facility**	
	Items	Bids	Customer	Facility
Size	500	550	29	24

3 Methodology

For SIB models, LP problems were generated with 700 constraints and 500 variables. Three independent values were sampled from predefined ranges to facilitate the allocation of constraints. For each inequality in $\{<, >, \leq\}$, the three samples were utilized as upper bounds to randomly determine the specific number of constraints assigned to that inequality. The remaining constraints were allocated to $\geq$. The constraints were shuffled throughout $\boldsymbol{A}$, and slack variables were introduced to convert all inequalities into equalities. Note that this method of problem generation is not completely random and some patterns still exist. Notably, after generating 100,000 instances, the average count of inequalities per problem converges to 163.3 for $<$, 167.44 for $>$, 176.08 for $\leq$, and 193.18 for $\geq$. The coefficients in $\boldsymbol{A}$, and c were uniformly sampled between 0 and 1. However, this process resulted in no zero values in $\boldsymbol{A}$. To address this, coefficients below a certain tolerance were selected, and a coin flip with a random probability between 0.1 and 0.9 was used to determine if the coefficient was set to zero. Additionally, small problems (200 variables, 280 constraints) and large problems (800 variables, 1120 constraints) were generated to test how the model scales on different problem sizes. Unlike the training data, the number of inequality signs were truly random for the test dataset. This is to test the model for generalizability on unseen problems. Node features were inspired from SIB [4] and detailed in Table 1.

For LTP, Expert 1 was developed as described in LTP method [9]. LP problems were solved using Expert 1, with each pivot recorded as training data for the LTP model.

For IPMGNN, the problem generators provided from IPMGNN [11] were used, and their "large instances" were referenced to determine the problem size. Table 2 shows the problem size for each problem type. The features mentioned from the IPMGNN method [11] were

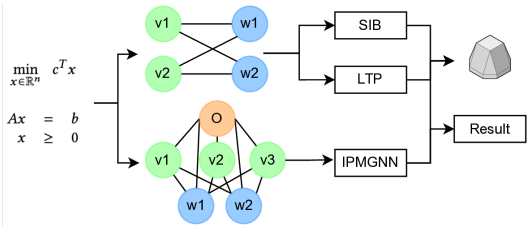

Figure 3: The process begins with inputting an LP problem and then converted into a Bi/Tri-partite graph with relevant features extracted. Features are passed into the corresponding model to generate an inference. Based on the output, the system samples the problem's feasible region and produces an image, which is then displayed on the website.

also utilized. To test, 1000 instances per problem were generated and tested with the trained model.

Formally, for each model, given a graph representation of the LP problem $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes, and $\mathcal{E}$ is the set of edges. The goal is to predict an initial basis $\mathcal{B}$ for the LP problem. Let $\mathbf{h}_v^{(0)}$ be the initial features for node v and $t = 1, \ldots, T$ be the message passing step. Let $\mathcal{N}(v)$ be the set of neighbors of node v, $\mathbf{e}_{uv}$ be the edge feature connecting node u and v, these models aim to learn parameters for the message passing function ϕ and ψ is the update function for the node feature. Once trained, the models were integrated into the website for user experimentation. See Figure 3 for the system overview.

$$\mathbf{h}_v^{(t)} = \psi\left(\mathbf{h}_v^{(t-1)}, \sum_{u \in \mathcal{N}(v)} \phi\left(\mathbf{h}_v^{(t-1)}, \mathbf{h}_u^{(t-1)}, \mathbf{e}_{uv}\right)\right). \tag{2}$$

4 Results and Discussions

4.1 SIB Analysis

700 LP instances were used for training, 100 for validation, and 200 for testing. On average, the SIB resulted in 70.65% similarity to the optimal basis with a standard deviation of 1.2%. The Revised Simplex Method was used to compare the number of pivots required from the default basis (all slack variables are active) and SIB's initial basis. Due to the time for this, the test was conducted on the first 20 test instances with a time limit of 30 minutes. The results are displayed on the websites. In 6 out of 20 instances, both basis exceeded the time limit. In 1 instance, the SIB terminated within the time limit while the default exceeded. In the remaining 13 instances, SIB outperformed the default basis 10 times with an average improvement of 46.9%. On the other hand, the default basis outperformed SIB 3 times with an average improvement of 29.93%. The SIB prediction performs better than the default, but there are cases where the default performed better for example, in problem 800 the default basis took 74,799 pivots while SIB took 150,000 pivots. In some cases, the predicted basis may be infeasible, but recovery processes such as LU factorizations [1, 2, 4] can transform the infeasible basis into a feasible one.

Figure 4 shows SIB's prediction on 200 small and large problems which were designed to be more unpredictable than the training data. Small problems scored an average of 67.07% with a standard

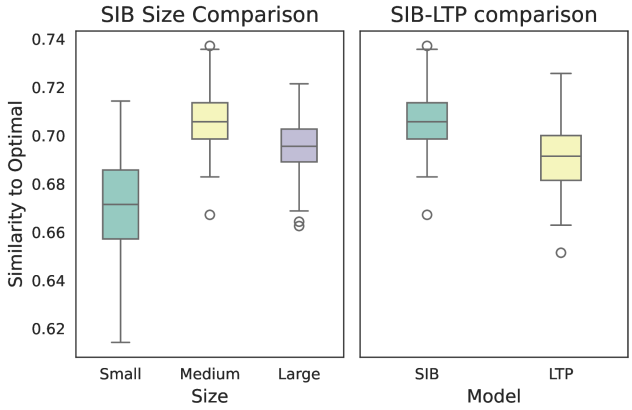

Figure 4: Box plot comparison of different problem sizes. The results indicate SIB model deteriorates a bit in smaller/larger instances. The medium instances were training size.

Table 3: IPMGNN Objective and Constraint Gap results

Problem	Setcover	Independent Set
Objective Gap (%)	19.167	40.847
Constraint Gap	0.2117	0.00069
Problem	Auction	Facility
Objective Gap (%)	27.843	1.622
Constraint Gap	0.0299	0.00061

deviation of 2.03%, while large problems scored 69.58% with a standard deviation of 1.08%. It was hypothesized the SIB's parameter expects more constraints interacting in the problem, however, this was incorrect for small problems which is possibly why it performs worse than large problems.

4.2 LTP Analysis

In the framework, the LTP model does not improve the SIB's predictions and often decreases accuracy by 1-2%, as seen on the right-hand side of Figure 4. The current LTP model also struggles to identify when to stop. In contrast, the revised Simplex method can mathematically stop at the optimal value. Furthermore, a sufficient initial basis from the SIB model can accelerate the Simplex Method by up to 77% [4].

4.3 IPMGNN Analysis

The IPMGNN model was trained on approximately 28,570 instances of Setcover, Independent Set, and Auction, along with 14,285 instances of Facility. This was done because the Facility problem performed significantly better than the others, as shown in Table 3. Test results show the average objective gap for all problems were below 41%. During experimentation, it was observed that when the model performed well on Setcover, the Independent Set and Auction would deteriorate, and vice versa. This is likely due to the difference in objectives—Setcover and Facility involve minimisation, while Independent Set and Auction involve maximisation—leading to a paradox between the problems. This occurred despite transforming all problems into minimisation objectives. On the other

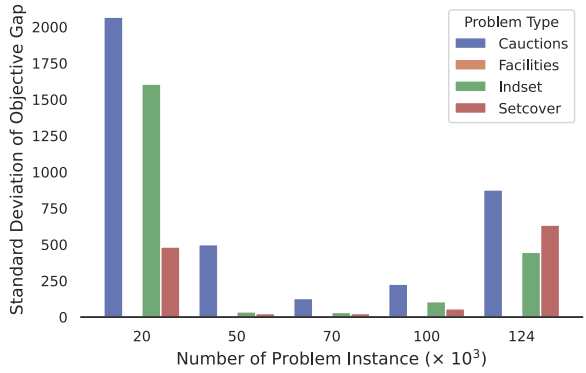

Figure 5: Chart displaying standard deviation (sd) against the number of training data. Result shows the sd decreases up until 70×10^3 but picks up again after. The Caution problem consistently has the highest sd while the Facility is the lowest.

hand, the constraint gap is considerably big, especially for the Setcover problem. For problems with fixed values, this is undesirable and investigation on repairments such as LU factorization is necessary [1, 2]. The current architecture is unstable, producing different results with each run. The IPMGNN algorithm ran 10 times using training datasets with 20, 50, 70, 100, and 124×10^3 total instances, and the results presented in Figure 5. As expected, when fewer data is available, the model becomes unstable, evident by a standard deviation of up to 2,000 on the Caution. However, after processing 70×10^3 files of LP instances, the standard deviation decreases to 100 for Caution. When the number of problems was increased up to 124×10^3, the standard deviation began to pick up again to ≈ 800. This may suggest limitations in the architecture's ability to effectively distinguish and represent diverse problems.

5 Conclusion and Future Works

Recently, many methods including SIB [4], LTP, [9], and IPMGNN, [11], have seen strong success in building GNN to speed up traditional LP and MILP solvers. However, in these papers, the GNN was trained and tested on one type of problem which are not beneficial for retail solvers such as Gurobi and CPlex, where various problems are solved. This paper explores the potential of GNN using SIB [4], LTP [9] and IPMGNN [11] models but for diverse LP problems. The result shows the SIB model is capable of predicting basis with 70.65% accuracy, and many of these indeed decrease the pivots required to optimality by 46.9% on average. However, instances where the default basis is faster also exist. When testing the SIB's prediction on smaller and larger problems with unseen problem structures, the SIB continues to perform well with a decline of 1-3% average. The LTP model struggles to pivot the SIB's initial basis to optimality and cannot identify when to stop. Lastly, the IPMGNN shows promising results on the objective gaps for all 4 IP problems, with the average gaps across all problem types to be under 41%.

There are still many future works to be conducted. Four main work ideas are discussed in this section. (1) The current algorithm struggles to identify the difference between minimisation and maximisation problems, resulting in paradoxical results. (2) Furthermore, the model is very unstable unless given the right amount of data, in this case around 70,000 instances. Interestingly, the model's standard deviation increases again once past this point. (3) Retail solvers like Gurobi are complex with different branches and cuts. Currently, the SIB and IPMGNN model is not integrated within these solvers making it a natural next step to work on. Understanding when the SIB's prediction will not work is also crucial when integrating these models into solvers. (4) Displayed in Figure 5, investigation on stabilising IPMGNN is also crucial to remove fluctuations as much as possible, especially as more problem types are fed into the model. (5) Lastly, this paper focuses on LP solvers, but MILP methods like B&B [5], along with Quadratic and Dynamic Programming, are also important.

6 Acknowledgments

This work is supported by Australian Research Council CE200100025 and DE250100919.

References

[1] Robert E. Bixby. 1992. Implementing the Simplex Method: The Initial Basis. *INFORMS J. Comput.* 4 (1992), 267–284. https://api.semanticscholar.org/CorpusID:36729926

[2] Robert E. Bixby and Matthew J. Saltzman. 1994. Recovering an optimal LP basis from an interior point solution. *Oper. Res. Lett.* 15 (1994), 169–178. https://api.semanticscholar.org/CorpusID:11090518

[3] Ziang Chen, Jialin Liu, Xinshang Wang, Jianfeng Lu, and Wotao Yin. 2023. On Representing Linear Programs by Graph Neural Networks. arXiv:2209.12288 [cs.LG] https://arxiv.org/abs/2209.12288

[4] Zhenan Fan, Xinglu Wang, Oleksandr Yakovenko, Abdullah Ali Sivas, Owen Ren, Yong Zhang, and Zirui Zhou. 2023. Smart Initial Basis Selection for Linear Programs. In *Proceedings of the 40th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 202)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 9650–9664. https://proceedings.mlr.press/v202/fan23d.html

[5] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. 2019. Exact Combinatorial Optimization with Graph Convolutional Neural Networks. arXiv:1906.01629 [cs.LG] https://arxiv.org/abs/1906.01629

[6] Narendra Karmarkar. 1984. A new polynomial-time algorithm for linear programming. *Combinatorica* 4 (1984), 373–395. https://api.semanticscholar.org/CorpusID:7257867

[7] Victor Klee and George J. Minty. 1970. HOW GOOD IS THE SIMPLEX ALGORITHM. https://api.semanticscholar.org/CorpusID:117965841

[8] Chang Liu, Zhichen Dong, Haobo Ma, Weilin Luo, Xijun Li, Bowen Pang, Jia Zeng, and Junchi Yan. 2024. L2P-MIP: Learning to Presolve for Mixed Integer Programming. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=McfYbKnpT8

[9] Tianhao Liu, Shanwen Pu, Dongdong Ge, and Yinyu Ye. 2024. Learning to Pivot as a Smart Expert. *Proceedings of the AAAI Conference on Artificial Intelligence* 38, 8 (Mar. 2024), 8073–8081. https://doi.org/10.1609/aaai.v38i8.28646

[10] John C Nash. 2000. The (Dantzig) simplex method for linear programming. *Computing in Science & Engineering* 2, 1 (2000), 29–31.

[11] Chendi Qian, Didier Chételat, and Christopher Morris. 2023. Exploring the Power of Graph Neural Networks in Solving Linear Optimization Problems. arXiv:2310.10603 [cs.LG] https://arxiv.org/abs/2310.10603

[12] Ragheb Rahmaniani, Teodor Gabriel Crainic, Michel Gendreau, and Walter Rei. 2017. The Benders decomposition algorithm: A literature review. *European Journal of Operational Research* 259, 3 (2017), 801–817. https://doi.org/10.1016/j.ejor.2016.12.005

[13] Helmut Simonis. 2006. Chapter 25 - Constraint Applications in Networks. In *Handbook of Constraint Programming*, Francesca Rossi, Peter van Beek, and Toby Walsh (Eds.). Foundations of Artificial Intelligence, Vol. 2. Elsevier, 875–903. https://doi.org/10.1016/S1574-6526(06)80029-5