Smart Initial Basis Selection for Linear Programs

Zhenan Fan [*1] **Xinglu Wang** [*1 2] **Oleksandr Yakovenko** [*1] **Abdullah Ali Sivas** [1] **Owen Ren** [1] **Yong Zhang** [1] **Zirui Zhou** [1]

Abstract

The simplex method, introduced by Dantzig more than half a century ago, is still to date one of the most efficient methods for solving large-scale linear programming (LP) problems. While the simplex method is known to have the finite termination property under mild assumptions, the number of iterations until optimality largely depends on the choice of initial basis. Existing strategies for selecting an advanced initial basis are mostly rule-based. These rules usually require extensive expert knowledge and empirical study to develop. Yet, many of them fail to exhibit consistent improvement, even for LP problems that arise in a single application scenario. In this paper, we propose a learning-based approach for initial basis selection. We employ graph neural networks as a building block and develop a model that attempts to capture the relationship between LP problems and their optimal bases. In addition, during the inference phase, we supplement the learning-based prediction with linear algebra tricks to ensure the validity of the generated initial basis. We validate the effectiveness of our proposed strategy by extensively testing it with state-of-the-art simplex solvers, including the open-source solver HiGHS and the commercial solver OptVerse. Through these rigorous experiments, we demonstrate that our strategy achieves substantial speedup and consistently outperforms existing rule-based methods. Furthermore, we extend the proposed approach to generating restricted master problems for column generation methods and present encouraging numerical results.

[*]Equal contribution [1]Huawei Technologies Canada, Burnaby, Canada [2]Simon Fraser University, Burnaby, Canada. Correspondence to: Zhenan Fan <zhenan.fan1@huawei.com>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

1. Introduction

Linear programming (LP) has been a fundamental aspect of various industrial domains for decades. A pioneering method for solving LP problems is the simplex method, which searches for an optimal solution by traversing the vertices of the polyhedron defined by the linear constraints. Ever since its introduction by Gass & Vinjamuri (2004) more than half a century ago, much progress has been made in improving the practical efficiency and stability of the simplex method, including the revised simplex method (Dantzig & Orchard-Hays, 1954), the Harris two-pass ratio test (Harris, 1973), anti-degeneracy techniques (Ryan & Osborne, 1988; Gill et al., 1989), crash methods (Bixby, 1992), pivoting rules (Forrest & Goldfarb, 1992), and parallel dual simplex (Huangfu & Hall, 2018), just to name a few. To date, the simplex method is still considered to be one of the most efficient methods for computing a highly accurate LP solution and is the workhorse of most commercial LP solvers.

It is known that the simplex method has finite termination as long as proper anti-degeneracy technique is adopted (Gass & Vinjamuri, 2004). Thus, the efficiency of the simplex method is mostly determined by the number of iterations it takes until termination. Although in the worst case, the number of iterations needed can be exponential in the input size (Klee & Minty, 1972), it is usually observed to be polynomial in practice (Spielman & Teng, 2004). Moreover, the initial basis of the simplex method has a considerable impact on the number of iterations until termination. In particular, starting with a basis that is much closer to an optimal one can often result in less number of iterations. Given this, many works have proposed strategies for selecting advanced initial bases for the simplex method; see, e.g., Bixby (1992), Gould & Reid (1989), Junior & Lins (2005), Ploskas et al. (2021), Galabova & Hall (2020). Most existing strategies are rule-based. These rules are usually developed by carefully analyzing the structure of the problem at hand and extensive empirical experience. Yet, most of them still fail to exhibit consistent improvement over the canonical initial basis.

In practice, it is very common that we need to solve a set of LP problems which share substantial similarities. Indeed, in many applications, an abstract LP model is usually first developed, which descriptively specifies variables, constraints, and objectives. Then, each LP problem is instantiated by feeding data to the abstract model, where the data can be obtained on the fly or sampled from an underlying distribution. A manufacturer's daily production planning and an airport's hourly flight scheduling are typical examples of this scenario. As these LP problems are generated by the same abstract model and correlated data, it is expected that the optimal bases of the solved LP problems can be exploited for constructing advanced initial bases for the ones to be solved. However, existing rule-based approaches treat each LP problem separately and thus cannot take advantage of the similarities across the LP problems. Therefore, it is natural to ask if a learning-based approach can be developed for selecting advanced initial bases so that the efficiency of the simplex method can be enhanced when applied to a set of correlated LP problems.

Several challenges are present towards a learning-based initial basis selection. First, while being correlated, the LP problems are often of varying sizes. Thus the proposed model must be capable of handling LP problems of different dimensions. Second, for each LP problem, the number of all potential bases is exponential in the problem size, resulting in an exponentially large output space. Third, the selected initial basis has to be valid in the sense that the corresponding basis matrix is non-singular and the status of non-basic variables are consistent with their bounds (Definition 4.1), both of which further complicate the prediction task.

In this paper, we propose the first learning-based strategy for initial basis selection in the simplex method. Based on the fact that every LP problem can be equivalently represented by a bipartite graph (Gasse et al., 2019), we employ a graph neural network (GNN) as a building block in our model, which allows us to handle problems of varying sizes. Our model takes LP instances as input and outputs the status of each variable, i.e., basic, non-basic at the upper bound, or non-basic at the lower bound. We further equip our model with a knowledge-based masking technique (Fan et al., 2022) so that every variable's predicted status is consistent with its bound in the LP formulation. These allow us to transform the original challenging task into a more manageable classification task. Finally, in the inference phase, we exploit the structure of the computational form of LP problems and adopt a basis repair technique (Bixby & Saltzman, 1994) to ensure that a non-singular basis is selected. The selected basis is then used as the starting basis of the simplex method.

We conduct extensive experiments on various LP data sets with state-of-the-art LP solvers, including the open-source solver HiGHS (Huangfu & Hall, 2018) and the commercial solver OptVerse (Huawei, 2021). The computational results show that starting the simplex method with the proposed learning-based initial basis strategy can achieve substantial improvement in both the number of iterations and end-to-end computational time. It consistently outperforms existing rule-based initial basis selection methods. Moreover, the time for generating an initial basis using our strategy is negligible as compared to the simplex solution time, making it a practically valid approach for dealing with industrial-level applications.

Furthermore, we extend the proposed learning framework to generating an advanced restricted master problem for the column generation (CG) method (Dantzig & Wolfe, 1960; Gasse et al., 2019), which is an extension of the simplex method for solving LP problems with a large ratio between the number of variables and the number of constraints. The computational results show that our approach can lead to much fewer CG iterations and shorter running time than the canonical selection of the restricted master problem.

2. Related Work

Classical initial-basis strategies In the literature, various initial-basis strategies have been proposed for solving LP problems faster. Logical basis (Chvatal, 1983; Bertsimas & Tsitsiklis, 1997) is a simple and commonly used strategy, which involves all slack variables as the initial basis. One advantage of the logical basis is that the corresponding matrix is the identity matrix, enabling efficient computation of its inverse. This advantage makes it the default option for many state-of-the-art solvers such as HiGHS (Huangfu & Hall, 2018) and CPLEX (Gearhart et al., 2013). However, this approach only works for problems without equality constraints and has to create artificial variables for the equality constraints. In order to select as few artificial variables as possible, the CPLEX crash algorithm is proposed by Bixby (1992), which uses heuristics to quickly guess an initial basis, prioritizing structural variables and creating a sparse, well-conditioned basis with fewer artificial variables than the logical basis. However, both approaches may produce an initial basis far from optimal. Another approach is the Idiot Crash algorithm (Galabova & Hall, 2020), which utilizes the augmented Lagrangian method to find an initial basis closer to an optimal one, though it generally has a slow convergence speed. In contrast, our proposed approach benefits from previous experience by learning a lightweight model through historical data, providing a suitable initial basis without additional solving costs, and achieving a near-optimal basis with a negligible inference cost.

Graph neural network for optimization The use of graph neural networks (GNNs) to assist optimization solvers has received significant attention in recent years. Gasse et al. (2019) first proposed a constraint-variable bipartite graph representation for mixed-integer LPs and a GNN model for learning a strong branching policy to accelerate MIP solvers. Following this framework, many researchers have proposed various approaches to improve MIP or LP solvers with GNNs (Ding et al., 2020; Gupta et al., 2022; Qu et al., 2022; Li et al., 2022). Gupta et al. (2022) discovered a "look-back" property missing in the trained GNN and proposed incorporating it into the training process, which resulted in further speedup for MIP solvers. Qu et al. (2022) proposed an improved reinforcement learning algorithm that builds upon imitation learning (Gasse et al., 2019). Considering that high-end GPUs may not be accessible to many practitioners, Gupta et al. (2020) proposed a hybrid model for branching in MIP inferencing on CPU and maintained competitive speedup. Ding et al. (2020) proposed a tripartite graph representation for MIP and used GNNs to predict solution values for binary variables. Li et al. (2022) reformulated the LP and reordered the variables and constraints using a GNN and a Pointer Network. More recently, Chen et al. (2022) built a theoretical foundation for the representation power of GNNs for LP, proving that given any LP, a GNN can be constructed that maps from the LP to its feasibility, boundedness, and an optimal solution. Despite the advancements made in using GNNs to assist optimization solvers, there is currently no work on initial basis selection for LP, pushing it towards practicality. Our proposed approach aims to fill this gap by providing a machine-learning-based initial basis selection strategy for LP.

3. Notations

We use $\underline{\mathbb{R}} = \mathbb{R} \cup \{-\infty\}$ and $\overline{\mathbb{R}} = \mathbb{R} \cup \{+\infty\}$ to denote extended real domains. We use I^m to denote the m-by-m identity matrix. Given a vector $x \in \mathbb{R}^n$, a matrix $A \in \mathbb{R}^{m\times n}$ and a subset $\mathcal{B} \subseteq [n]$, we use $x_{\mathcal{B}} \in \mathbb{R}^{|\mathcal{B}|}$ to denote the subvector of x that contains the entries of x in $\mathcal{B}$ and use $A_{\mathcal{B}} \in \mathbb{R}^{m\times|\mathcal{B}|}$ to denote the submatrix of A that contains the columns of A in $\mathcal{B}$. We use Δ^d to denote the d-dimensional simplex, *i.e.*, $\Delta^d = \{p \in \mathbb{R}^d_+ \mid \sum_{i=1}^d p_i = 1\}$. We denote by $\mathsf{OneHot}(d)$ the set of d-dimensional binary vectors with exactly one entry being non-zero. The softmax mapping is defined by $\mathsf{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^d \exp(z_j)}$ $,\forall i \in [d]$. The crossentropy loss is defined by $\ell_{\mathsf{CE}}(p,q) = -\sum_{i=1}^d q_i \log(p_i)$.

4. Preliminaries on Linear Programs

We consider the following standard format of LP

$$\begin{aligned} \min_{x\in\mathbb{R}^n, s\in\mathbb{R}^m} \quad & c^T x \\ \text{s.t.} \quad & Ax = s \\ & \ell^x \le x \le u^x \\ & \ell^s \le s \le u^s, \end{aligned} \tag{P}$$

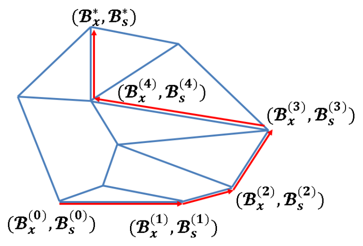
$(\mathcal{B}_x^*, \mathcal{B}_s^*)$
$(\mathcal{B}_x^{(4)}, \mathcal{B}_s^{(4)})$
$(\mathcal{B}_x^{(3)}, \mathcal{B}_s^{(3)})$
$(\mathcal{B}_x^{(2)}, \mathcal{B}_s^{(2)})$
$(\mathcal{B}_x^{(0)}, \mathcal{B}_s^{(0)})$
$(\mathcal{B}_x^{(1)}, \mathcal{B}_s^{(1)})$

Figure 1. Illustration of Simplex algorithm. It starts with initial basis $(\mathcal{B}_x^{(0)}, \mathcal{B}_s^{(0)})$, and routinely pivots to a neighbouring basis with improvement till it reaches an optimal basis $(\mathcal{B}_x^*, \mathcal{B}_s^*)$.

where $A \in \mathbb{R}^{m\times n}$ is the constraint matrix with $m \le n$, $c \in \mathbb{R}^n$ is the cost vector, $x \in \mathbb{R}^n$ is known as the decision variable, $s \in \mathbb{R}^m$ is known as the constraint variable, $\ell^x \in \underline{\mathbb{R}}^n$ and $u^x \in \overline{\mathbb{R}}^n$ are lower and upper bounds for the decision variable x, and $\ell^s \in \underline{\mathbb{R}}^m$ and $u^s \in \overline{\mathbb{R}}^m$ are lower and upper bounds for the constraint variable s. This formulation agrees with the input formulation for many LP solvers such as HiGHS (Huangfu & Hall, 2018), OptVerse (Huawei, 2021), CPLEX (Cplex, 2009) and Gurobi (Gurobi Optimization, LLC, 2022).

Without loss of generality, we assume the following assumption holds throughout the paper.

Assumption 4.1. *The LP problem* (P) *is feasible and bounded.*

4.1. Basis and Basic Solution

Next, we formally introduce the concept of basis and basic feasible solutions for LP problems.

Definition 4.1. *Given two index sets $\mathcal{B}_x \subset [n]$ and $\mathcal{B}_s \subseteq [m]$, the tuple $(\mathcal{B}_x, \mathcal{B}_s)$ is called a **basis** for the LP problem* (P) *if $|\mathcal{B}_x| + |\mathcal{B}_s| = m$ and the matrix $[A_{\mathcal{B}_x} \;\; -I^m_{\mathcal{B}_s}]$ is non-singular. Let $(\mathcal{N}_x, \mathcal{N}_s) = ([n] \setminus \mathcal{B}_x,\ [m] \setminus \mathcal{B}_s)$. A tuple $(x^{\mathcal{B}}, s^{\mathcal{B}}) \in \mathbb{R}^n \times \mathbb{R}^m$ is called a **basic solution** for* (P) *with respect to the basis $(\mathcal{B}_x, \mathcal{B}_s)$ if $Ax^{\mathcal{B}} = s^{\mathcal{B}}$ and for any $i \in \mathcal{N}_x$ and $j \in \mathcal{N}_s$, we have*

$$x_i^{\mathcal{B}} = \begin{cases} \ell_i^x & \text{if } u_i^x = +\infty \\ u_i^x & \text{if } \ell_i^x = -\infty \\ \ell_i^x \text{ or } u_i^x & \text{otherwise} \end{cases} \quad \text{and}$$

$$s_j^{\mathcal{B}} = \begin{cases} \ell_j^s & \text{if } u_j^s = +\infty \\ u_j^s & \text{if } \ell_j^s = -\infty \\ \ell_j^s \text{ or } u_j^s & \text{otherwise.} \end{cases}$$

*Moreover, if $\ell^x \le x^{\mathcal{B}} \le u^x$ and $\ell^s \le s^{\mathcal{B}} \le u^x$, then $(x^{\mathcal{B}}, s^{\mathcal{B}})$ is called a **basic feasible solution** for* (P).

4.2. Simplex Method

In this section, we briefly introduce the primal simplex method for solving problem (P). Note that given a basis

$(\mathcal{B}_x, \mathcal{B}_s)$ and the values for $(x^{\mathcal{B}}_{\mathcal{N}_x}, s^{\mathcal{B}}_{\mathcal{N}_s})$, the remaining values in the basic primal solution $(x^{\mathcal{B}}, s^{\mathcal{B}})$ are uniquely determined, *i.e.*,

$$\begin{bmatrix} x^{\mathcal{B}}_{\mathcal{B}_x} \\ \\ s^{\mathcal{B}}_{\mathcal{B}_s} \end{bmatrix} = [A_{\mathcal{B}_x} \quad -I^m_{\mathcal{B}_s}]^{-1} \left(I^m_{\mathcal{N}_s} s^{\mathcal{B}}_{\mathcal{N}_s} - A_{\mathcal{N}_x} x^{\mathcal{B}}_{\mathcal{N}_x} \right).$$

The primal simplex method is initialized with a primal feasible basis. If the basis is not dual feasible, then the primal simplex method will pick a dual infeasible index p to enter the basis and a leaving index $q \in \mathcal{B}_x \cup \mathcal{B}_s$ such that $(\mathcal{B}_x, \mathcal{B}_s) \setminus \{q\} \cup \{p\}$ is still primal feasible. Note that adding p into the basis will perturb the values $(x^{\mathcal{B}}_{\mathcal{B}_x}, s^{\mathcal{B}}_{\mathcal{B}_s})$, and q is defined as the first index violating the primal constraint.

Similarly, the dual simplex method is initialized with a dual feasible basis. In each iteration, it will find a leaving index q that is not primal feasible and find an entering index p such that the resulting basis is still dual feasible.

In summary, the simplex method is an algorithm that starts with initial basis $(\mathcal{B}^{(0)}_x, \mathcal{B}^{(0)}_s)$, from which we can obtain a basic solution $(x^{(0)}, s^{(0)})$. It then routinely visits candidate bases and corresponding basic solutions till it encounters an optimal basic feasible solution.

$$(\mathcal{B}^{(0)}_x, \mathcal{B}^{(0)}_s) \to (x^{(0)}, s^{(0)}) \to (\mathcal{B}^{(1)}_x, \mathcal{B}^{(1)}_s) \to (x^{(1)}, s^{(1)}) \ldots$$

This process is illustrated in Figure 1. We refer interested readers to (Maros, 2002) for a more detailed description of the simplex method.

Although the simplex method is one of the most widely adopted algorithms for solving LP problems, the theoretical guarantee of its performance is actually weak. Klee & Minty (1972) showed that the simplex method has exponential time complexity in the worst case. Moreover, many empirical results demonstrate that the performance of the simplex method depends largely on the selection of the initial basis (Bixby, 1992; Huang et al., 2021; Sangngern & Boonperm, 2020).

4.3. Column Generation Method

When the LP problem (P) has a huge number of columns, *i.e.*, $n \gg m$, searching for entering and leaving variables might be computationally intractable. Dantzig & Wolfe (1960) and Gilmore & Gomory (1961) proposed the column generation (CG) method as an extension of the simplex method. The CG method starts with a subset $\mathcal{F} \subseteq [n]$ of variables and then deals with the following restricted master problem (RMP)

$$\begin{aligned} \min_{x \in \mathbb{R}^{|\mathcal{F}|}, s \in \mathbb{R}^m} \quad & c^T_{\mathcal{F}} x \\ \text{s.t.} \quad & A_{\mathcal{F}} x = s \\ & \ell^x_{\mathcal{F}} \le x \le u^x_{\mathcal{F}} \\ & \ell^s \le s \le u^s. \end{aligned} \tag{RMP}$$

Note that the initial subset $\mathcal{F}$ need to be carefully chosen so that the (RMP) is feasible. Then the CG algorithm will solve the (RMP) and use the corresponding optimal dual solution to add columns from $[n] \setminus \mathcal{F}$. Then the (RMP) will be updated and solved again. The process is repeated until an optimal solution is found.

5. GNN-based model for Initial Basis Selection

In this section, we step-by-step show how we are going to learn a mapping from an LP instance to a valid basis. Generally speaking, our methodology can be divided into two major steps: training and inference. More specifically, during the training step (Section 5.1), we want to learn a mapping $f(\theta; \cdot)$ such that given an LP instance (P) with n variables and m constraints, $f(\theta; (P)) = \{p_{x,i} \in \Delta^3, p_{s,j} \in \Delta^3 \mid i \in [n],\ j \in [m]\}$, where θ is the learnable parameters in f, and $p_{x,i}, p_{s,j}$ indicate the probability of the label of the corresponding variable, *i.e.*,

$$\begin{aligned} p_{x,i} &= [\mathbb{P}(x_i = \ell^x_i)\ \mathbb{P}(\ell^x_i < x_i < u^x_i)\ \mathbb{P}(x_i = u^x_i)]^T \quad \text{and} \\ p_{s,j} &= \left[\mathbb{P}(s_j = \ell^s_j)\ \mathbb{P}(\ell^s_j < s_j < u^s_j)\ \mathbb{P}(s_j = u^s_j)\right]^T. \end{aligned} \tag{1}$$

During the inference step (Section 5.2), we first generate a candidate basis $(\mathcal{B}_x, \mathcal{B}_s)$ according to the probabilities given by $f(\theta; (P))$, *i.e.*,

$$(\mathcal{B}_x, \mathcal{B}_s) \in \operatorname*{argmax}_{|\mathcal{B}_x| + |\mathcal{B}_s| = m} \left[\prod_{i \in \mathcal{B}_x} \mathbb{P}(\ell^x_i < x_i < u^x_i) \prod_{j \in \mathcal{B}_s} \mathbb{P}(\ell^s_j < s_j < u^s_j) \right].$$

Then we adjust the basis to make it valid, *i.e.*, the matrix $[A_{\mathcal{B}_x} \quad -I^m_{\mathcal{B}_s}]$ being non-singular (Definition 4.1).

5.1. Graph Representation and GNN Model formulation

The goal is to learn the mapping $f(\theta; \cdot)$. However, what we have is a training set with K samples $\mathcal{D} = \left\{[(P^k), (x^k, s^k)]\right\}^K_{k=1}$. In k-th sample, (P^k) is an LP problem with the same formulation as (P), and (x^k, s^k) is a optimal basic feasible solution to (P^k). In this section, we will represent (P^k) as a graph, construct the label from (x^k, s^k), and instantiate $f(\theta; \cdot)$ as a GNN model.

Graph representation. Gasse et al. (2019) suggested representing the LP problem as a weighted bipartite graph.

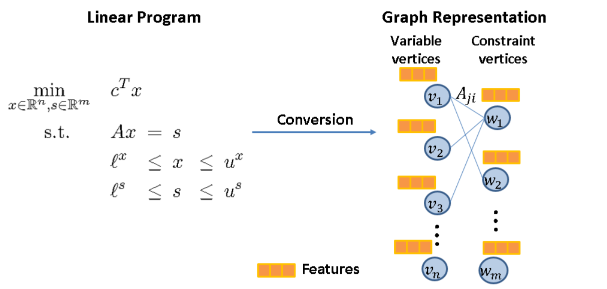

Figure 2. Represent the LP problem as a weighted bipartite graph. The graph topology is defined by the constraint matrix A. The information in c, A, l^x, l^u, l^s, and l^u will be extracted to build the node features (Appendix A).

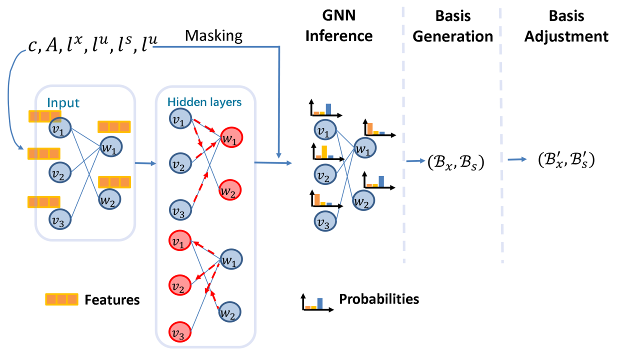
c, A, l^x, l^u, l^s, l^u
Masking
GNN Inference
Basis Generation
Basis Adjustment
Input
Hidden layers
$(\mathcal{B}_x, \mathcal{B}_s)$
$(\mathcal{B}'_x, \mathcal{B}'_s)$
Features
Probabilities

Figure 3. Overall procedure of the inference step.

As shown in Figure 2, a weighted bipartite graph $\mathcal{G} = (V, W, E)$ consists of two disjoint vertex sets V and W, and a collection E of weighted edges, where each edge connects exactly one vertex in V and one vertex in W. When representing an LP as a weighted bipartite graph, each variable vertex in V represents a decision variable x_i, and each constraint vertex in W represents a constraint variable s_j, and the graph topology is defined by the constraint matrix A. More specifically,

- The variable vertex set V contains n nodes $\{v_1, \dots, v_n\}$. Each node $v_i \in \mathbb{R}^p$ contains information about decision variable x_i. The composition of node information v_i is shown in Appendix A.
- The constraint vertex set W contains m nodes $\{w_1, \dots, w_m\}$, Each node $w_j \in \mathbb{R}^q$ contains information about the constraint variable s_j.
- The edge set is defined by $E_{(v_i, w_j)} = A_{ji}$ for all $(i, j) \in [n] \times [m]$.

Note that one big advantage of representing LP as a weighted bipartite graph is permutation equivalence. It refers to the equivalence of two LP problems when the decision variables, cost vector, bound vectors, and columns in the constraint matrix are permutated in the same order.

Following this bipartite graph representation, for any problem instance (P), we can uniquely transform it into a weighted bipartite graph, *i.e.*, $(P) \to \mathcal{G} = (V, W, E)$.

Label construction. As we introduced in Section 4.2, a valid initial basis for the simplex method requires the following information:

- A valid basis $(\mathcal{B}_x, \mathcal{B}_s)$.
- For any $i \in \mathcal{N}_x$, whether $x_i^{\mathcal{B}} = \ell_i^x$ or $x_i^{\mathcal{B}} = u_i^x$.
- For any $j \in \mathcal{N}_s$, whether $s_j^{\mathcal{B}} = \ell_j^s$ or $s_j^{\mathcal{B}} = u_j^s$.

For this purpose, given any optimal basic feasible solution (x, s), we transform it into $n + m$ one-hot labels $\{y_{x,i}, y_{s,j} \in \mathsf{OneHot}(3) \mid i \in [n], j \in [m]\}$. Specifically,

$$y_{x,i} = \begin{cases} [1\;\;0\;\;0]^T & \text{if } x_i = \ell_i^x \\ [0\;\;1\;\;0]^T & \text{if } \ell_i^x < x_i < u_i^x \\ [0\;\;0\;\;1]^T & \text{if } x_i = u_i^x \end{cases} \quad \text{and}$$
$$y_{s,j} = \begin{cases} [1\;\;0\;\;0]^T & \text{if } s_j = \ell_j^s \\ [0\;\;1\;\;0]^T & \text{if } \ell_j^s < s_j < u_j^s \\ [0\;\;0\;\;1]^T & \text{if } s_j = u_j^s. \end{cases} \tag{2}$$

Message-passing functions. We adopt the standard message-passing framework for the graph neural network (Scarselli et al., 2009). A graph neural network with L message-passing steps will be learned. The learnable message-passing functions are denoted as

$$f_l^V(\theta_l^V; \cdot) : \mathbb{R}^{d_{l-1}^V} \times \mathbb{R}^{d_{l-1}^W} \to \mathbb{R}^{d_l^V} \quad \text{and}$$
$$f_l^W(\theta_l^W; \cdot) : \mathbb{R}^{d_{l-1}^W} \times \mathbb{R}^{d_{l-1}^V} \to \mathbb{R}^{d_l^W} \quad \text{for } l \in [L],$$

where $\{\theta_l^V, \theta_l^W \mid l \in [L]\}$ are the learnable parameters, $d_0^V = p$, $d_0^W = q$ and $d_L^V = d_L^W = 3$. The node embeddings will be updated via these message-passing functions, *i.e.*, for any $i \in [n]$, $j \in [m]$ and $l \in [L]$,

$$v_i^l = f_l^V\left(\theta_l^V; v_i^{l-1}, \sum_{j=1}^m E_{(v_i,w_j)} w_j^{l-1}\right) \quad \text{and}$$
$$w_i^l = f_l^W\left(\theta_l^W; w_i^{l-1}, \sum_{i=1}^n E_{(v_i,w_j)} v_i^{l-1}\right), \tag{3}$$

where the 0-step embeddings v_i^0, w_j^0 are the input features v_i, w_j.

Knowledge-based masking. Then, after the L message passing steps, we want to predict the label for each node based on its current feature. Namely, we want to design mappings $v_i^L \to p_{x,i} \in \Delta^3$ and $w_j^L \to p_{s,j} \in \Delta^3$, $\forall i \in$

$[n], j \in [m]$, such that $p_{x,i}, p_{s,j}$ are the probabilities defined in (1).

It is an important requirement that these resulting probabilities satisfy the feasibility of non-basic entries. Formally, for any variable x_i, if $\ell_i^x = -\infty$ or $u_i^x = +\infty$, then we want the corresponding probability to be zero, *i.e.*, $\mathbb{P}(x_i = \ell_i^x) = 0$ or $\mathbb{P}(x_i = u_i^x) = 0$. A similar requirement exists for constraint variables. To satisfy the requirements, we adopt the knowledge-based masking technique proposed by Fan et al. (2022), where the key idea is to mask out the probability entries based on the knowledge of the problem. More specifically, we define $p_{x,i} = \mathsf{softmax}(v_i^L + h_{x,i})$ and $p_{s,j} = \mathsf{softmax}(w_j^L + h_{s,j})$, where

$$h_{x,i} = \begin{cases} [-\infty \;\; 0 \;\; 0]^T & \text{if } \ell_i^x = -\infty \\ [0 \;\; 0 \;\; -\infty]^T & \text{if } u_i^x = +\infty \\ [0 \;\; 0 \;\; 0]^T & \text{otherwise} \end{cases} \quad \text{and}$$

$$h_{s,j} = \begin{cases} [-\infty \;\; 0 \;\; 0]^T & \text{if } \ell_j^s = -\infty \\ [0 \;\; 0 \;\; -\infty]^T & \text{if } u_j^s = +\infty \\ [0 \;\; 0 \;\; 0]^T & \text{otherwise.} \end{cases}$$

Training Loss Finally, we define the loss function, where we use the cross-entropy function to measure the mismatch between the resulting probability and the ground truth label for every decision and constraint variable. Let $\theta = \{\theta_l^V, \theta_l^W \mid 1 \le l \le L\}$ denote the set of all learnable parameters. The loss function is defined as

$$\mathcal{L}_{\mathcal{D}}(\theta) = \frac{1}{K}\sum_{k=1}^{K} \ell\left(\theta; (P^k), (x^k, s^k)\right),$$

where

$$\ell\left(\theta; (P), (x,s)\right) = \Bigg[\sum_{i=1}^{n} \alpha(y_{x,i})\, \ell_{\mathsf{CE}}(p_{x,i}, y_{x,i}) + \sum_{j=1}^{m} \alpha(y_{s,j})\, \ell_{\mathsf{CE}}(p_{s,j}, y_{s,j})\Bigg].$$

The probabilities $\{p_{x,i}, p_{s,j} \mid 1 \le i \le n, 1 \le j \le m\}$ comes from $f(\theta; (P))$. Considering the basis label can be imbalanced, *e.g.*, only a few slacks are basic, weight terms $\alpha(y_{x,i})$ and $\alpha(y_{x,i})$ are introduced into the training loss. These terms are the inverse frequency of current label, *i.e.*, $\alpha(y_{x,i}) = 1/\sum_{i' \in [n]} \mathbf{1}\{y_{x,i'} = y_{x,i}\}$ and $\alpha(y_{s,j}) = 1/\sum_{j' \in [m]} \mathbf{1}\{y_{s,j'} = y_{s,j}\}$. Here, $\mathbf{1}\{\cdot\}$ is the indicator function.

5.2. Inference

In this section, we show how to infer a valid basis from the probabilities predicted by the learned mapping $f(\theta; \cdot)$. Given an LP instance (P), recall that the predicted probabilities from $f(\theta; (P))$ are $\{p_{x,i}, p_{s,j} \mid i \in [n], j \in [m]\}$.

Basis generation. First, we select basis as the indices corresponding to the top-m predicted values for $\mathbb{P}(\ell_i^x < x_i < u_i^x)$ and $\mathbb{P}(\ell_j^s < s_j < u_j^s)$, *i.e.*,

$$(\mathcal{B}_x, \mathcal{B}_s) \in \underset{|\mathcal{B}_x|+|\mathcal{B}_s|=m}{\operatorname{argmax}} \prod_{i \in \mathcal{B}_x} p_{x,i}[2] \prod_{j \in \mathcal{B}_s} p_{s,j}[2].$$

However, the chosen $(\mathcal{B}_x, \mathcal{B}_s)$ may not be a valid basis, as the matrix $[A_{\mathcal{B}_x} \;\; -I^m_{\mathcal{B}_s}]$ may be singular (Definition 4.1).

Basis adjustment. Next, we need to adjust $(\mathcal{B}_x, \mathcal{B}_s)$ to make it a valid basis. Our adjustment approach is inspired by a basis repair procedure described in (Bixby & Saltzman, 1994). We try to factor the matrix $[A_{\mathcal{B}_x} \;\; -I^m_{\mathcal{B}_s}]$ as described in Bixby (1992). Note that this factorization does not incur additional computational complexity, as it has to be done during the simplex method. During the factorization, if we encounter a column whose pivot value is smaller than some fixed tolerance, then we will remove this column by eliminating the corresponding index from $(\mathcal{B}_x, \mathcal{B}_s)$. After the factorization, if the basis is incomplete, *i.e.*, $|\mathcal{B}_x|+|\mathcal{B}_s| < m$, then we will fill it by adding the non-selected indices according to the predicted probabilities. The factorization is then attempted again until a complete and successfully factored basis is produced.

Finally, we determine statuses for the non-basic entries according to predicted probabilities of reaching the upper bound or lower bound, *i.e.*, for any $i \in \mathcal{N}_x$ and $j \in \mathcal{N}_s$

$$x_i^{\mathcal{B}} = \begin{cases} \ell_i^x & \text{if } p_{x,i}[1] \ge p_{x,i}[3] \\ u_i^x & \text{otherwise} \end{cases} \quad \text{and}$$

$$s_j^{\mathcal{B}} = \begin{cases} \ell_j^s & \text{if } p_{s,j}[1] \ge p_{s,j}[3] \\ u_j^s & \text{otherwise.} \end{cases}$$

The overall procedure of the inference step is shown in Figure 3.

6. Numerical Experiments

In this section, we want to evaluate the performance of our proposed methodology as a warm-start strategy for the simplex and column generation methods, as well as to investigate the impact of dataset diversity on the methodology and the potential for model transferability. To accomplish this, we conduct a series of experiments consisting of four main parts. First, in Section 6.1, we test the performance of the proposed methodology as a warm-start strategy for the simplex method. Second, in Section 6.2, we repeat this experiment but using the column generation method. Third, in Section 6.3, we sought to determine the impact of dataset diversity on the proposed methodology by training the model on generated datasets with varying diversity and comparing the results. Finally, in Section 6.4, we evaluate the potential for model transferability by training the model on one dataset and then testing it on another dataset that comes from a different source.

Datasets In this study, we evaluate the performance of our proposed method on a diverse set of datasets, including three publicly available datasets, two privately sourced datasets, and one synthetic dataset. The publicly available datasets include the Maritime Inventory Routing Problems (MIRP) (Papageorgiou et al., 2014), the One-norm Support Vector Machine Instances (LIBSVM) (Zhu et al., 2003; Applegate et al., 2021), and the 2-stage Stochastic Problems (STOCH) (Castro & de la Lama-Zubirán, 2020). The privately sourced datasets are derived from large-scale supply chain problems in two different industries, denoted as SC-1 and SC-2. The synthetic dataset (GEN) is generated using the LP generator developed by Bowly et al. (2020). The statistics of the datasets are presented in the appendix. To ensure a fair evaluation, each dataset is split into training and test sets in a 7:3 ratio.

Candidate Optimization Solvers We evaluate the performance of the proposed methodology using two optimization solvers: HiGHS (Huangfu & Hall, 2018), a state-of-the-art open-source solver that offers both primal and dual simplex methods, and an OptVerse solver (Huawei, 2021) that additionally incorporates the column generation algorithm. To eliminate any potential impact from solver configurations, the presolve option is turned off, and default settings are used.

Implementation We implement our approach with Python 3.7, PyTorch 1.8 and PyG framework (Fey & Lenssen, 2019). The GNN model is trained on NVIDIA V100. The evaluation is conducted on a system comprised of 8 cores CPU (Intel Xeon E5-2690 v4) and 64 G of memory, utilizing Ubuntu 18.04 Docker containers for solver execution. Our code is publicly available at Huawei AI Gallery [1].

6.1. Initial-Basis strategy for the simplex algorithm

In this section, we evaluate the performance of the proposed methodology as a warm-start strategy for the simplex method. The proposed method is compared to two commonly used initial-basis strategies: **1).** DEFAULT, in which the initial basis contains all the slack variables; **2).** CPLEX Crash (CA) (Bixby, 1992), which employs a heuristic approach to construct a basis with improved triangularity and a reduced number of artificial variables. Meanwhile, two more recent rule-based basis-selection strategies are compared based on OptVerse solver: **3).** CA-MPC (Ploskas et al., 2021), which constructs a sparse initial basis using triangulation and fill-reducing techniques; **4).** CA-ANG (Junior & Lins, 2005), which builds an initial basis *heuristically* closer to the optimal vertex.

Both the primal and dual simplex methods were employed in the evaluation. Due to the page limit, the results obtained from the primal simplex method are presented in the appendix. This is also because the dual simplex algorithm is more commonly employed in practical scenarios (Bertsimas & Tsitsiklis, 1997; Vanderbei et al., 2020).

Performance evaluations using the dual simplex method are presented in Table 1 and Table 2, which compares the number of simplex iterations and total running time. The running time includes both the time required to find the initial basis and the time required to execute the simplex method. The results are presented in the form of $mean_{\pm std}$ over the test set.

We would like to highlight the results in Table 1, the proposed initial-basis strategy consistently outperforms the DEFAULT and the rule-based strategies (CA, CA-MPC, and CA-ANG) in terms of both the number of simplex iterations and total running time across all datasets. It is because they are built upon heuristics and these heuristics have three drawbacks: **1).** CA and CA-MPC are designed to achieve better numerical properties for the initial basis matrix, instead of close to optimal. Consequently, only the several starting iterations can be accelerated and the number of iterations will not necessarily decrease. **2).** CA-ANG wants to find an initial basis that is close to the optimal basis, but it only works when angular conjectures hold. Similarly, the other rule-based methods are limited to specific problems and structures. CA-CPLEX only works for problems with equality constraints, CA-MPC prefers a constraint matrix with more singleton columns. **3).** They do not utilize past solved LPs, and thus limit the capacity and applicability.

The acceleration observed in the results is attributed to two aspects: **1).** the inference time is negligible; **2).** the predicted initial bases are close to optimal ones. To verify this point, the inference time and the prediction performance of the proposed approach are presented in Table 3. The "Inference time (s)" column represents the additional time required during the inference stage, including the time for GNN inference and basis adjustment. As we can see from the results, the inference cost of the proposed method is insignificant, accounting for less than 10% of the total running time. The performance of the GNN model with respect to optimal bases is also reported. We report the test accuracy, precision and recall. The test accuracy is defined as the number of correctly predicted variables over the total number of variables. The precision and recall are defined

[1] https://developer.huaweicloud.com/develop/aigallery/notebook/detail?id=ce45dd10-44ce-43bb-89c8-1f3277f1132d

	Iterations					Time (s)				
Dataset	DEFAULT	CA	CA-MPC	CA-ANG	GNN	DEFAULT	CA	CA-MPC	CA-ANG	GNN
LIBSVM	$14.9K_{\pm 9.5K}$	$14.9K_{\pm 9.5K}$	$21.0K_{\pm 4.8K}$	$15.2K_{\pm 1.1K}$	$\mathbf{9.1K}_{\pm \mathbf{3.1K}}$	$16.6_{\pm 10.0}$	$16.7_{\pm 10.0}$	$27.9_{\pm 12.4}$	$28.3_{\pm 2.2}$	$\mathbf{11.0}_{\pm \mathbf{3.7}}$
MIRP	$40.3K_{\pm 23.3K}$	$34.8K_{\pm 20.2K}$	$36.7K_{\pm 20.8K}$	$39.6K_{\pm 22.7K}$	$\mathbf{25.9K}_{\pm \mathbf{16.9K}}$	$22.1_{\pm 23.3}$	$21.4_{\pm 22.5}$	$18.6_{\pm 16.9}$	$21.6_{\pm 20.9}$	$\mathbf{15.4}_{\pm \mathbf{15.7}}$
STOCH	$75.3K_{\pm 4.3K}$	$52.5K_{\pm 4.8K}$	$48.7K_{\pm 5.2K}$	$53.3K_{\pm 1.7K}$	$\mathbf{31.8K}_{\pm \mathbf{14.3K}}$	$44.6_{\pm 11.8}$	$61.3_{\pm 12.3}$	$51.3_{\pm 12.4}$	$53.2_{\pm 8.5}$	$\mathbf{42.7}_{\pm \mathbf{30.0}}$
GEN	$2.4K_{\pm 225.0}$	$2.4K_{\pm 225.0}$	$2.4K_{\pm 225.0}$	$2.4K_{\pm 225.0}$	$\mathbf{552.8}_{\pm \mathbf{642.9}}$	$1.3_{\pm 0.2}$	$1.4_{\pm 0.2}$	$1.4_{\pm 0.3}$	$1.4_{\pm 0.3}$	$\mathbf{0.5}_{\pm \mathbf{0.5}}$
SC-1	$272.3K_{\pm 151.9K}$	$158.9K_{\pm 89.1K}$	$266.9K_{\pm 148.5K}$	$269.2K_{\pm 151.5K}$	$\mathbf{26.6K}_{\pm \mathbf{15.4K}}$	$77.9_{\pm 68.4}$	$85.8_{\pm 80.3}$	$86.1_{\pm 79.5}$	$100.1_{\pm 94.0}$	$\mathbf{22.8}_{\pm \mathbf{23.5}}$
SC-2	$1.2M_{\pm 170.7K}$	$1.1M_{\pm 172.2K}$	$1.2M_{\pm 163.5K}$	$431.9K_{\pm 99.0K}$	$\mathbf{169.1K}_{\pm \mathbf{34.3K}}$	$348.7_{\pm 101.0}$	$1.3K_{\pm 698.2}$	$382.8_{\pm 102.3}$	$338.7_{\pm 181.5}$	$\mathbf{87.3}_{\pm \mathbf{25.4}}$

Table 1. Performance comparison between the proposed and rule-based initial-basis strategy, with the dual simplex method and the OptVerse solver.

	Iterations			Time (s)		
Dataset	DEFAULT	CA	GNN	DEFAULT	CA	GNN
LIBSVM	$9.0K_{\pm 178.8}$	$9.0K_{\pm 64.2}$	$\mathbf{5.2K}_{\pm \mathbf{1.5K}}$	$7.4_{\pm 0.2}$	$7.5_{\pm 0.1}$	$\mathbf{4.6}_{\pm \mathbf{1.2}}$
MIRP	$29.9K_{\pm 17.0K}$	$25.9K_{\pm 14.5K}$	$\mathbf{18.2K}_{\pm \mathbf{12.3K}}$	$17.8_{\pm 17.2}$	$17.6_{\pm 16.7}$	$\mathbf{14.5}_{\pm \mathbf{14.2}}$
STOCH	$343.2K_{\pm 36.6K}$	$261.4K_{\pm 36.2K}$	$\mathbf{165.1K}_{\pm \mathbf{42.6K}}$	$718.7_{\pm 112.5}$	$553.9_{\pm 102.9}$	$\mathbf{251.6}_{\pm \mathbf{66.7}}$
GEN	$2.2K_{\pm 105.0}$	$2.2K_{\pm 103.4}$	$\mathbf{80.4}_{\pm \mathbf{186.1}}$	$1.3_{\pm 0.1}$	$1.3_{\pm 0.1}$	$\mathbf{0.2}_{\pm \mathbf{0.2}}$
SC-1	$262.6K_{\pm 147.0K}$	$207.5K_{\pm 111.0K}$	$\mathbf{64.9K}_{\pm \mathbf{36.5K}}$	$21.3_{\pm 18.7}$	$62.9_{\pm 50.0}$	$\mathbf{11.1}_{\pm \mathbf{11.2}}$
SC-2	$1.2M_{\pm 165.5K}$	$1.1M_{\pm 128.8K}$	$\mathbf{214.3K}_{\pm \mathbf{40.4K}}$	$194.4_{\pm 58.9}$	$336.6_{\pm 103.6}$	$\mathbf{65.0}_{\pm \mathbf{25.1}}$

Table 2. Evaluation of the performance of the initial-basis strategy for the dual simplex method using the HiGHS solver.

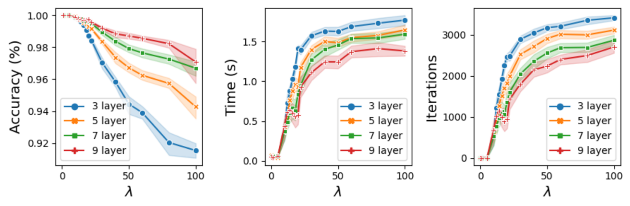

Figure 4. **Left:** Test accuracy versus λ. **Middle:** Total running time versus λ. **Right:** Simplex iterations versus λ.

	Inference time (s)		Prediction performance		
Dataset	GNN inference	Basis adjustment	Accuracy (%)	Precision (%)	Recall (%)
LIBSVM	$0.1_{\pm 8e\text{-}3}$	$0.1_{\pm 2e\text{-}3}$	$87.1_{\pm 4e\text{-}2}$	$84.6_{\pm 4e\text{-}2}$	$87.7_{\pm 2e\text{-}2}$
MIRP	$0.1_{\pm 5e\text{-}2}$	$3e\text{-}3_{\pm 2e\text{-}3}$	$88.9_{\pm 1.8}$	$78.4_{\pm 1.6}$	$80.8_{\pm 5.4}$
STOCH	$2e\text{-}2_{\pm 7e\text{-}3}$	$4e\text{-}3_{\pm 2e\text{-}3}$	$81.7_{\pm 1.9}$	$81.3_{\pm 1.9}$	$81.3_{\pm 1.9}$
GEN	$2e\text{-}2_{\pm 6e\text{-}3}$	$0.1_{\pm 2e\text{-}2}$	$99.9_{\pm 0.1}$	$99.9_{\pm 0.1}$	$99.9_{\pm 0.1}$
SC-1	$0.1_{\pm 4e\text{-}2}$	$0.3_{\pm 0.2}$	$93.0_{\pm 2.1}$	$89.0_{\pm 5.4}$	$84.6_{\pm 6.5}$
SC-2	$0.3_{\pm 0.1}$	$0.3_{\pm 0.1}$	$90.4_{\pm 0.8}$	$90.8_{\pm 0.7}$	$78.4_{\pm 2.0}$

Table 3. Test performance of the GNN prediction model.

similarly, with detailed formulas shown in the appendix A. With test accuracy, precision, and recall metrics exceeding 81%, 78%, and 78%, respectively, across all datasets, it is verified that our approach produces near-optimal bases.

6.2. Initial-RMP strategy for the column generation algorithm

In this section, we evaluate the performance of the proposed methodology as a warm-up strategy for the column generation (CG) method. As we introduced in Section 4.3, the CG method starts with a restricted master problem (RMP) which only contains a subset of the variables. Ideally, if the predicted basis is optimal, then we can construct an RMP and solve it in zero iteration. This motivates us to accelerate the CG algorithm with GNN predicted basis.

The CG method necessitates that the initial RMP is feasible. To meet this requirement, we first compute a basic solution corresponding to the GNN-predicted basis. However, this basic solution may not be feasible. To circumvent this issue, we construct an equivalent auxiliary LP by introducing artificial slack variables for the infeasible constraints. This auxiliary LP then serves as the initial RMP for the CG method.

We compare our initial-RMP strategy with the default strategy (DEFAULT), which heuristically builds an initial feasible solution with many variables fixed at bounds, and then constructs an initial RMP with the unfixed variables.

Performance evaluations using the CG method are presented in Table 5, which compares the number of CG iterations and total running time. As demonstrated by the results, the proposed initial-RMP strategy consistently produces fewer CG iterations and shorter running times than the DEFAULT across all datasets.

6.3. The impact of dataset diversity

In this section, we aim to evaluate the performance of our proposed approach on datasets with varying degrees of diversity. The efficacy of data-dependent machine learning techniques for solving LP problems is contingent on the similarity of the mapping from an LP instance to its optimal basis in a given dataset. To this end, we devise a problem-generation strategy that allows for the controllable manipulation of the diversity of the mapping.

Our problem-generation strategy is designed based on the LP-generation technique developed by Bowly et al. (2020). Constraint matrix $A \in \mathbb{R}^{m \times n}$ is generated in the same way, where the maximum numbers of nonzeros in rows and columns will be less than thresholds τ_{row} and τ_{column}. Then we select basic entries according to the number of nonzeros in A and generate an optimal basic solution (x, s) through the following steps:

- $k = \lfloor \gamma m \rfloor$;
- Choose k indices $\mathcal{B}_x \subseteq [n]$ according to probabilities $\text{softmax}_\lambda([nnz(A[:, i])])$ without replacement; Here, $\text{softmax}_\lambda(z)_i = \frac{\exp(z_i/\lambda)}{\sum_{j=1}^{d} \exp(z_j/\lambda)}$, $\forall i \in [d]$.
- Choose $m - k$ indices $\mathcal{B}_s \subseteq [m]$ according to probabil-

Source \ Target	Iteration Ratio=GNN Iterations / DEFAULT Iterations						Time Ratio=GNN Time / DEFAULT Time					
	LIBSVM	MIRP	STOCH	GEN	SC-1	SC-2	LIBSVM	MIRP	STOCH	GEN	SC-1	SC-2
LIBSVM	$\mathbf{0.8}_{\pm 0.3}$	$1.0_{\pm 0.1}$	$0.8_{\pm 3e\text{-}2}$	$1.7_{\pm 0.2}$	$0.9_{\pm 0.1}$	$0.9_{\pm 3e\text{-}2}$	$\mathbf{0.8}_{\pm 0.2}$	$1.1_{\pm 0.2}$	$2.1_{\pm 0.4}$	$1.8_{\pm 0.3}$	$1.8_{\pm 0.5}$	$2.5_{\pm 0.4}$
MIRP	$1.1_{\pm 0.4}$	$\mathbf{0.6}_{\pm 0.1}$	$1.0_{\pm 4e\text{-}2}$	$1.4_{\pm 0.9}$	$1.0_{\pm 3e\text{-}2}$	$1.0_{\pm 5e\text{-}3}$	$1.1_{\pm 0.4}$	$\mathbf{0.7}_{\pm 0.1}$	$1.0_{\pm 0.1}$	$3.5_{\pm 6.1}$	$1.2_{\pm 0.1}$	$1.2_{\pm 0.2}$
STOCH	$1.0_{\pm 0.4}$	$1.3_{\pm 0.1}$	$\mathbf{0.4}_{\pm 0.2}$	$1.6_{\pm 0.5}$	$0.6_{\pm 0.3}$	$1.2_{\pm 0.2}$	$1.0_{\pm 0.4}$	$1.6_{\pm 0.4}$	$\mathbf{0.9}_{\pm 0.5}$	$0.7_{\pm 0.3}$	$1.7_{\pm 0.9}$	$3.9_{\pm 2.4}$
GEN	$2.1_{\pm 1.1}$	$1.0_{\pm 0.1}$	$0.8_{\pm 3e\text{-}2}$	$\mathbf{0.2}_{\pm 0.3}$	$1.2_{\pm 0.1}$	$1.3_{\pm 0.1}$	$2.0_{\pm 1.0}$	$1.1_{\pm 0.3}$	$2.1_{\pm 0.3}$	$\mathbf{0.1}_{\pm 0.1}$	$3.1_{\pm 1.0}$	$7.4_{\pm 4.3}$
SC-1	$1.0_{\pm 0.4}$	$0.8_{\pm 0.1}$	$1.3_{\pm 0.2}$	$1.1_{\pm 0.1}$	$\mathbf{0.1}_{\pm 2e\text{-}2}$	$\mathbf{0.1}_{\pm 2e\text{-}2}$	$1.2_{\pm 0.4}$	$0.8_{\pm 0.1}$	$3.1_{\pm 0.7}$	$1.1_{\pm 0.1}$	$\mathbf{0.3}_{\pm 0.1}$	$\mathbf{0.3}_{\pm 3e\text{-}2}$
SC-2	$1.0_{\pm 0.4}$	$0.8_{\pm 0.1}$	$0.8_{\pm 0.1}$	1.0	$0.3_{\pm 0.1}$	$\mathbf{0.1}_{\pm 2e\text{-}2}$	$1.0_{\pm 0.3}$	$1.0_{\pm 0.1}$	$1.2_{\pm 0.1}$	$1.0_{\pm 0.1}$	$0.6_{\pm 0.3}$	$\mathbf{0.3}_{\pm 4e\text{-}2}$

Table 4. Performance of models trained on a source dataset transferring to a target dataset. The entries show the Iteration/Time ratio between utilizing the GNN model trained on a source dataset and adopting the DEFAULT strategy towards a target dataset. **Bold** entries mean that on the corresponding target dataset (column-wise) the fastest iterations/time is achieved.

Dataset	Iterations		Time (s)	
	DEFAULT	GNN	DEFAULT	GNN
LIBSVM	$34.3K_{\pm 45.1}$	$\mathbf{17.6K}_{\pm 1.9K}$	$24.9_{\pm 0.2}$	$\mathbf{19.0}_{\pm 2.7}$
MIRP	$59.2K_{\pm 46.0K}$	$\mathbf{31.2K}_{\pm 25.9K}$	$54.7_{\pm 65.2}$	$\mathbf{23.7}_{\pm 28.0}$
STOCH	$99.3K_{\pm 6.8K}$	$\mathbf{33.3K}_{\pm 12.1K}$	$29.7_{\pm 10.2}$	$\mathbf{23.3}_{\pm 19.0}$
GEN	$4.2K_{\pm 2.4K}$	$\mathbf{95.7}_{\pm 120.9}$	$1.4_{\pm 0.5}$	$\mathbf{0.5}_{\pm 0.3}$
SC-1	$345.2K_{\pm 194.0K}$	$\mathbf{68.1K}_{\pm 41.4K}$	$64.8_{\pm 56.3}$	$\mathbf{31.7}_{\pm 35.0}$
SC-2	$1.6M_{\pm 238.6K}$	$\mathbf{384.1K}_{\pm 92.7K}$	$721.6_{\pm 299.4}$	$\mathbf{463.5}_{\pm 586.1}$

Table 5. Evaluation of the performance of the initial-RMP strategy for the column generation method using the OptVerse solver.

ities $\mathsf{softmax}_\lambda([nnz(A[j,:])])$ without replacement;

- Randomly fill in remaining values for (x, s).

Finally, following the same way, c and b are computed using complementary slackness, and LPs are generated.

In our LP-generating strategy, the scalar λ controls the similarity between the mapping from an LP instance to an optimal basis. When $\lambda = 1$ the $\mathsf{softmax}_\lambda$ coincides with $\mathsf{softmax}$, when $\lambda \to \infty$ the $\mathsf{softmax}_\lambda$ tends to uniform distribution, and when $\lambda \to 0$ the $\mathsf{softmax}_\lambda$ approximates one-hot distribution. To conclude, as λ increases, the generated dataset becomes more diverse.

The evaluation results are presented in Figure 4, where we test the performance of GNN models with a varying number of layers. The left figure illustrates the relationship between test accuracy and the diversity of the dataset (λ). As depicted in the left figure, the test accuracy decreases as the diversity of the dataset increases, indicating that the diversity of the instances has a significant impact on the GNN prediction. The middle and right figures demonstrate the effect of dataset diversity on our initial-basis strategy for the simplex method. The results show that both the running time and the number of simplex iterations increase as the diversity of the dataset increases, which is consistent with the findings in the left figure, as the predicted bases are farther from the optimal ones.

6.4. Cross-dataset evaluation

In this section, we evaluate the transferability of our proposed approach. The motivation for this evaluation is that, in practice, datasets may originate from different sources, such as airplane scheduling and product planning. It is unknown whether our approach could produce a general model transferable to various sources.

To test this, we train the GNN model on one dataset and then test it on another, with the two datasets having distinct sources. The results are presented in Table 4. It is observed that most models (except the one trained on SC-2) perform best on their corresponding test sets. The model trained on SC-1 performs well on the SC-2 test set, which may be because SC-1 and SC-2 have the same source (supply chain demand). For the remaining models, their performance on test sets from other sources is poor, as they perform similarly to the DEFAULT initial-basis strategy.

In summary, the numerical results suggest that our proposed approach does not possess good model transferability. This may be attributed to the different problem structures of LPs from different sources, such as the block structures in the constraint matrix and the topology of the corresponding simplex polyhedron.

7. Conclusion and Future Directions

In this paper, by employing the graph neural network, we develop a model aiming to learn the relationship between LP problems and their optimal bases. We show that our model leads to warm-start strategies for both simplex and column generation methods. Extensive numerical experiments with different optimization solvers all show that our proposed warm-start strategies outperform existing ones.

Moreover, we believe our model can assist the LP solvers besides providing warm-start strategies. For example, it may help to design smart pivoting strategies in the simplex method. Alternatively, the predicted basis information may be utilized in the interior-point method as a preconditioner for the augmented system (Schork & Gondzio, 2020).

References

Applegate, D., Díaz, M., Hinder, O., Lu, H., Lubin, M., O'Donoghue, B., and Schudy, W. Practical large-scale linear programming using primal-dual hybrid gradient. *NeurIPS*, 34:20243–20257, 2021.

Bertsimas, D. and Tsitsiklis, J. N. *Introduction to linear* *optimization*, volume 6. Athena Scientific, 1997.

Bixby, R. E. Implementing the simplex method: The initial basis. *INFORMS J. Comput.*, 4(3):267–284, 1992.

Bixby, R. E. and Saltzman, M. J. Recovering an optimal LP basis from an interior point solution. *Oper. Res. Lett.*, 15 (4):169–178, 1994.

Bowly, S., Smith-Miles, K., Baatar, D., and Mittelmann, H. Generation techniques for linear programming instances with controllable properties. *Math. Program. Comput.*, 12 (3):389–415, 2020. doi: 10.1007/s12532-019-00170-6.

Castro, J. and de la Lama-Zubirán, P. A new interior-point approach for large separable convex quadratic two-stage stochastic problems. *Optim. Methods Software*, pp. 1–29, 2020.

Chen, Z., Liu, J., Wang, X., Lu, J., and Yin, W. On representing linear programs by graph neural networks. *arXiv:2209.12288*, 2022.

Chvatal, V. *Linear programming*. W.H. Freeman, 1983.

Cplex, I. I. V12. 1: User's manual for CPLEX. *International Business Machines Corporation*, 46(53):157, 2009.

Dantzig, G. B. and Orchard-Hays, W. The product form for the inverse in the simplex method. *Math. Comp.*, pp. 64–67, 1954. doi: 10.1090/S0025-5718-1954-0061469-8.

Dantzig, G. B. and Wolfe, P. Decomposition principle for linear programs. *Oper. Res.*, 8(1):101–111, 1960.

Ding, J.-Y., Zhang, C., Shen, L., Li, S., Wang, B., Xu, Y., and Song, L. Accelerating primal solution findings for mixed integer programs based on solution prediction. In *AAAI*, volume 34, pp. 1452–1459, 2020. doi: 10.1609/AAAI.V34I02.5503.

Fan, Z., Zhou, Z., Pei, J., Friedlander, M. P., Hu, J., Li, C., and Zhang, Y. Knowledge-injected federated learning. *arXiv:2208.07530*, 2022.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Forrest, J. J. and Goldfarb, D. Steepest-edge simplex algorithms for linear programming. *Math. Program.*, 57(1): 341–374, 1992.

Galabova, I. L. and Hall, J. A. J. The 'Idiot' crash quadratic penalty algorithm for linear programming and its application to linearizations of quadratic assignment problems. *Optim. Methods Software*, 35(3):488–501, 2020. doi: 10.1080/10556788.2019.1604702.

Gass, S. I. and Vinjamuri, S. Cycling in linear programming problems. *Comput. Oper. Res.*, 31(2):303–311, 2004. doi: 10.1016/S0305-0548(02)00226-5.

Gasse, M., Chételat, D., Ferroni, N., Charlin, L., and Lodi, A. Exact combinatorial optimization with graph convolutional neural networks. *NeurIPS*, 32, 2019.

Gearhart, J. L., Adair, K. L., Durfee, J. D., Jones, K. A., Martin, N., and Detry, R. J. Comparison of open-source linear programming solvers. Technical report, Sandia National Lab., 2013.

Gill, P. E., Murray, W., Saunders, M. A., and Wright, M. H. A practical anti-cycling procedure for linearly constrained optimization. *Math. Program.*, 45(1):437–474, 1989. doi: 10.1007/BF01589114.

Gilmore, P. C. and Gomory, R. E. A linear programming approach to the cutting-stock problem. *Oper. Res.*, 9(6): 849–859, 1961.

Gould, N. I. and Reid, J. K. New crash procedures for large systems of linear constraints. *Math. Program.*, 45(1): 475–501, 1989.

Gupta, P., Gasse, M., Khalil, E., Mudigonda, P., Lodi, A., and Bengio, Y. Hybrid models for learning to branch. *NeurIPS*, 33:18087–18097, 2020.

Gupta, P., Khalil, E. B., Chetélat, D., Gasse, M., Bengio, Y., Lodi, A., and Kumar, M. P. Lookback for learning to branch. *arXiv:2206.14987*, 2022.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022. URL `https://www.gurobi.com`.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. *NeurIPS*, 30, 2017.

Harris, P. M. Pivot selection methods of the Devex LP code. *Math. Program.*, 5(1):1–28, 1973.

Huang, M., Zhong, Y., Yang, H., Wang, J., Zhang, F., Bai, B., and Shi, L. Simplex initialization: A survey of techniques and trends. *arXiv:2111.03376*, 2021.

Huangfu, Q. and Hall, J. J. Parallelizing the dual revised simplex method. *Math. Program. Comput.*, 10(1):119–142, 2018. doi: 10.1007/s12532-017-0130-5.

Huawei. Optverse solver, 2021. URL `https://www.huaweicloud.com/product/modelarts/optverse.html`.

Junior, H. V. and Lins, M. P. E. An improved initial basis for the simplex algorithm. *Comput. Oper. Res.*, 32(8): 1983–1993, 2005.

Klee, V. and Minty, G. J. How good is the simplex algorithm. *Inequalities*, 3(3):159–175, 1972.

Li, X., Qu, Q., Zhu, F., Zeng, J., Yuan, M., Mao, K., and Wang, J. Learning to reformulate for linear programming. *arXiv:2201.06216*, 2022.

Maros, I. *Computational techniques of the simplex method*, volume 61. Springer Science & Business Media, 2002.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *AAAI*, volume 33, pp. 4602–4609, 2019. doi: 10.1609/aaai.v33i01.33014602.

Papageorgiou, D. J., Nemhauser, G. L., Sokol, J., Cheon, M.-S., and Keha, A. B. MIRPLib–a library of maritime inventory routing problem instances: Survey, core model, and benchmark results. *Eur. J. Oper. Res.*, 235(2):350–366, 2014.

Ploskas, N., Sahinidis, N. V., and Samaras, N. A triangulation and fill-reducing initialization procedure for the simplex algorithm. *Math. Program. Comput.*, 13:491–508, 2021.

Qu, Q., Li, X., Zhou, Y., Zeng, J., Yuan, M., Wang, J., Lv, J., Liu, K., and Mao, K. An improved reinforcement learning algorithm for learning to branch. *arXiv:2201.06213*, 2022.

Ryan, D. M. and Osborne, M. R. On the solution of highly degenerate linear programmes. *Math. Program.*, 41(1): 385–392, 1988.

Sangngern, K. and Boonperm, A.-a. A new initial basis for the simplex method combined with the nonfeasible basis method. In *J. Phys. Conf. Ser.*, volume 1593, pp. 012002. IOP Publishing, 2020. doi: 10.1088/1742-6596/1593/1/012002.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Trans. Neural Netw.*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

Schork, L. and Gondzio, J. Implementation of an interior point method with basis preconditioning. *Math. Program. Comput.*, 12(4):603–635, 2020.

Spielman, D. A. and Teng, S.-H. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *JACM*, 51(3):385–463, 2004.

Vanderbei, R. J. et al. *Linear programming*. Springer, 2020.

Zhu, J., Rosset, S., Tibshirani, R., and Hastie, T. $1-$norm support vector machines. *NeurIPS*, 16, 2003.

A. Implementation details

The information in c, A, l^x, l^u, l^s, l^u will be extracted to build the node features for constraints and variables. The features for each constraint and variable node are vectors in $\mathbb{R}^8$. Denote by $A_{i:}$ and $A_{:j}$ respectively the i-th row and j-th column in constraint matrix A. Denote by $\langle \cdot, \cdot \rangle$ the cosine similarity between two vectors. The meaning for each feature dimension is shown in Tab. 6. To ensure numerical stability, a tag dimension is added. If the lower/upper bound does not exist, the tag dimension will be marked as -1/+1.

Feature index	Constraint node j	Variable node i
1	$\langle A_{j:}, c \rangle$	c_j
2	$nnz(A_{j:})/n$	$nnz(A_{:i})/m$
3	$\langle A_{j:}, l^x \rangle$	$\langle l^s, A_{:i} \rangle$
4	$\langle A_{j:}, u^x \rangle$	$\langle l^u, A_{:i} \rangle$
5	$\begin{cases} l^s_j & \text{if } l^s_j \neq -\infty \\ 0 & \text{else} \end{cases}$	$\begin{cases} l^x_i & \text{if } l^x_i \neq -\infty \\ 0 & \text{else} \end{cases}$
6	$\begin{cases} 0 & \text{if } l^s_j \neq -\infty \\ -1 & \text{else} \end{cases}$	$\begin{cases} 0 & \text{if } l^x_i \neq -\infty \\ -1 & \text{else} \end{cases}$
7	$\begin{cases} u^s_j & \text{if } u^s_j \neq \infty \\ 0 & \text{else} \end{cases}$	$\begin{cases} u^x_i & \text{if } u^x_i \neq \infty \\ 0 & \text{else} \end{cases}$
8	$\begin{cases} 0 & \text{if } u^s_j \neq \infty \\ 1 & \text{else} \end{cases}$	$\begin{cases} 0 & \text{if } u^x_i \neq \infty \\ 1 & \text{else} \end{cases}$

Table 6. Feature for each constraint node j and variable node i

For the benchmark datasets, we follow the default setup (Papageorgiou et al., 2014; Applegate et al., 2021; Castro & de la Lama-Zubirán, 2020; Bowly et al., 2020) and only customize such that LPs comes from a shared source because we are considering the scenario where a series of similar LP problems are to be solved. In LIBSVM, the source is set to Cod-RNA dataset. The 488,565 data points are split into train and test set by the ratio 1:1. The train and test LPs are constructed using randomly selected 20K data points respectively from the train and test set. In MIPR, Group-1 LPs are selected without any modification. In STOCH, to enrich the LPs, we generate the 2-stage stochastic supply chain problems by setting the number of first-stage variables ranging from 75 to 85. In GEN, the diversity parameter λ defined in Section 6.3 is set to 10. Table. 7 shows the dataset statistics.

For the model architecture, we adopt the GNN proposed by Morris et al. (2019) and customize the graph convolution operation. To efficiently handle the large-scale sparse constraint-variable bipartite graph, one graph convolution layer is implemented as two sparse matrix multiplication respectively for message passing from constraint to variable node and back. By default, 3 layers lightweight GNN is used for SC-1 and SC-2 datasets, and 5 layers GNN is used for other datasets. The size of the hidden layers is 128 and the dropout ratio is 0.1. Let m represent the number of constraints, n the number of variables, and d the average degree of the bipartite graph. Taking message-passing as the basic operation, the complexity of a single LP is given by $\mathcal{O}(d(m+n))$. This is because computing the representation of each node involves its neighbors, and the sum of all nodes is equal to the number of edges. In practice, constraint matrices tend to be sparse, which means that the average degree is typically small. As a result, the complexity scales roughly linearly with the problem size, making

Dataset	#LPs	m=#constraints	n=#variables	density	#(basic variables)/n (%)	#(basic slacks)/m (%)
LIBSVM	100	$20.0K$	$20.0K$	5e-4$_{\pm 2e\text{-}19}$	31.3$_{\pm 2e\text{-}3}$	68.7$_{\pm 2e\text{-}3}$
MIRP	28	$28.2K_{\pm 25.2K}$	$28.7K_{\pm 25.0K}$	2e-4$_{\pm 1e\text{-}4}$	40.0$_{\pm 3.1}$	58.6$_{\pm 4.3}$
STOCH	100	$52.3K_{\pm 1.9K}$	$107.0K_{\pm 3.8K}$	5e-5	48.9$_{\pm 5e\text{-}4}$	0.0$_{\pm 2e\text{-}4}$
GEN	100	$1.0K$	$1.0K$	0.1$_{\pm 1e\text{-}3}$	60	40
SC-1	525	$312.9K_{\pm 177.4K}$	$659.1K_{\pm 386.4K}$	3e-5$_{\pm 2e\text{-}4}$	38.8$_{\pm 4.5}$	20.1$_{\pm 4.6}$
SC-2	190	$1.4M_{\pm 199.1K}$	$2.9M_{\pm 450.6K}$	2e-6$_{\pm 3e\text{-}7}$	36.7$_{\pm 1.5}$	22.6$_{\pm 1.4}$

Table 7. The statistics of datasets.

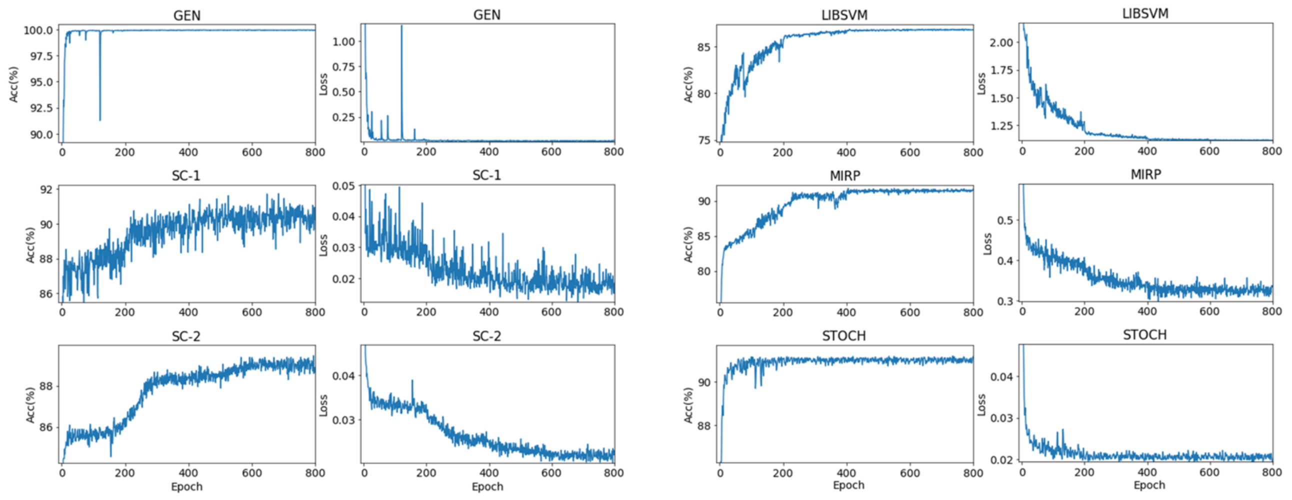

Figure 5. The convergence plots for training accuracy and loss

Dataset	Iterations			Time (s)		
	Simplex	GNN	CA	Simplex	GNN	CA
LIBSVM	$20.2K_{\pm 2.4K}$	$\mathbf{19.5K}_{\pm \mathbf{3.1K}}$	$20.2K_{\pm 2.4K}$	$18.2_{\pm 2.8}$	$\mathbf{18.1}_{\pm \mathbf{2.7}}$	$18.5_{\pm 2.8}$
STOCH	$56.2K_{\pm 30.1K}$	$\mathbf{15.4K}_{\pm \mathbf{9.5K}}$	$19.3K_{\pm 8.4K}$	$44.7_{\pm 36.1}$	$\mathbf{8.3}_{\pm \mathbf{7.0}}$	$9.1_{\pm 7.3}$
MIRP	$36.3K_{\pm 26.9K}$	$\mathbf{24.1K}_{\pm \mathbf{18.4K}}$	$32.8K_{\pm 22.9K}$	$24.5_{\pm 26.3}$	$\mathbf{15.7}_{\pm \mathbf{16.9}}$	$20.4_{\pm 21.7}$
SC-1	$359.6K_{\pm 210.4K}$	$\mathbf{34.9K}_{\pm \mathbf{23.1K}}$	$211.2K_{\pm 126.1K}$	$173.4_{\pm 205.1}$	$\mathbf{34.8}_{\pm \mathbf{65.2}}$	$142.7_{\pm 164.6}$
SC-2	$1.8M_{\pm 275.9K}$	$\mathbf{393.5K}_{\pm \mathbf{72.1K}}$	$1.2M_{\pm 184.9K}$	$1.4K_{\pm 627.7}$	$\mathbf{324.5}_{\pm \mathbf{133.1}}$	$937.2_{\pm 449.0}$

Table 8. Performance evaluation with **primal** simplex algorithm in the **commercial** solver

our approach highly scalable. Considering the graphs in SC-2 can have more than 2M nodes and 10M edges, even 1 graph per batch requires more 32G memory. Thus, we adopt Neighbourhood Sampling (Hamilton et al., 2017) when #edges is more than 10M. We do not tune hyper-parameter for training GNN and just adopt the commonly used one, like the Adam optimizer with initial learning rate 1e-3 and weight decay 1e-4. The learning rate decays by 0.1 every 200 epochs and the total training epoch is 800. The model is trained on Nvidia V100 with 32 G memory. The training time is less than 1 hour on GEN, LIBSVM, STOCH, and MIRP, around 4 hours on SC-1, and 8 hours on SC-2. The convergence plots including training accuracy and loss are shown in Figure 5.

The reported metrics for our task are different from that in the traditional machine learning community because the basis generation step in our approach ensures that exactly m entries are predicted as basic. Thus, given one LP, if we calculate on $m+n$ entries, then the precision and recall will be equal (and also equal to accuracy when $m=n$). These metrics will not be ideal measurements for model performance. Only with both good precision and recall, we can say that the model is high-performance. However, with original metrics, we will fail to measure the model's performance when labels inside variables or constraints are imbalanced.

Denote by $\left\{\hat{y}_{x,i}^{(k)}\right\}_{i=1}^{n}$ and $\left\{\hat{y}_{s,j}^{(k)}\right\}_{j=1}^{m}$ the predicted statuses of variables and constraints for k-th testing LP, and by $\left\{y_{x,i}^{(k)}\right\}_{i=1}^{n}$ and $\left\{y_{s,j}^{(k)}\right\}_{j=1}^{m}$ the corresponding ground-truth labels. Denote our reported metric as $\tilde{M}$ and the original metric in ML community as M, where M can be the accuracy, macro-precision, or macro-recall metrics. $\tilde{M}$ extends M in the following way:

$$\tilde{M} = \frac{1}{2K}\sum_{k=1}^{K}\left[M\left(\left\{\hat{y}_{x,i}^{(k)}\right\}_{i=1}^{n}, \left\{y_{x,i}^{(k)}\right\}_{i=1}^{n}\right) + M\left(\left\{\hat{y}_{s,j}^{(k)}\right\}_{j=1}^{m}, \left\{\hat{y}_{s,j}^{(k)}\right\}_{j=1}^{m}\right)\right].$$

	Iterations			Time (s)		
Dataset	Simplex	GNN	CA	Simplex	GNN	CA
GEN	$3.9K_{\pm 661.8}$	$\mathbf{279.6}_{\pm \mathbf{444.2}}$	$3.9K_{\pm 661.8}$	$1.8_{\pm 0.5}$	$\mathbf{0.2}_{\pm \mathbf{0.3}}$	$1.9_{\pm 0.5}$
LIBSVM	$9.1K_{\pm 726.2}$	$\mathbf{6.1K}_{\pm \mathbf{794.1}}$	$9.1K_{\pm 726.2}$	$6.0_{\pm 0.6}$	$\mathbf{3.7}_{\pm \mathbf{0.6}}$	$6.0_{\pm 0.6}$
STOCH	$332.5K_{\pm 39.6K}$	$\mathbf{314.8K}_{\pm \mathbf{68.5K}}$	$318.5K_{\pm 25.9K}$	$635.7_{\pm 169.4}$	$\mathbf{558.5}_{\pm \mathbf{70.1}}$	$581.3_{\pm 101.2}$
MIRP	$148.2K_{\pm 120.0K}$	$\mathbf{116.2K}_{\pm \mathbf{91.1K}}$	$131.6K_{\pm 111.1K}$	$111.4_{\pm 127.1}$	$\mathbf{82.1}_{\pm \mathbf{87.0}}$	$98.1_{\pm 112.9}$
SC-1	$387.4K_{\pm 199.5K}$	$\mathbf{100.8K}_{\pm \mathbf{86.2K}}$	$213.6K_{\pm 106.6K}$	$1.6K_{\pm 1.5K}$	$\mathbf{258.8}_{\pm \mathbf{613.7}}$	$593.5_{\pm 598.9}$
SC-2	$1.5M_{\pm 231.2K}$	$\mathbf{146.2K}_{\pm \mathbf{27.2K}}$	$1.7M_{\pm 313.2K}$	$17.9K_{\pm 4.6K}$	$\mathbf{2.0K}_{\pm \mathbf{651.5}}$	$23.2K_{\pm 7.5K}$

Table 9. Performance evaluation for **primal** simplex algorithm in **HiGHS** solver

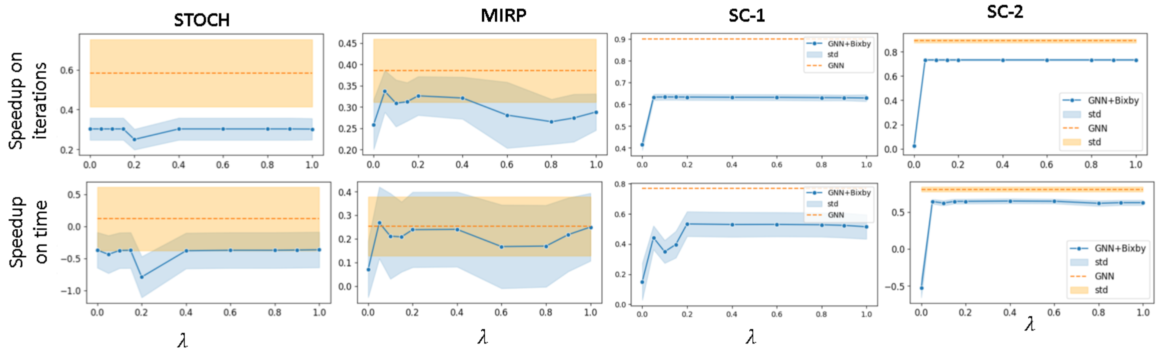

Figure 6. Speedup v.s. the weight of GNN prediction on Mirp, STOCH, SC-1, and SC-2 dataset.

B. Performance evaluation with Primal Simplex Algorithm

Table 8-9 shows the performance of our method with the Primal Simplex Algorithm. In Table 9, the primal simplex algorithm on HiGHS spends around 4 hours solving 1 LP in SC-2. It will be time-consuming to solve all LPs from scratch. Thus, we use the dual simplex algorithm to obtain the GT label, train a GNN model and predict the basis for the primal simplex algorithm. The results reported in row 'SC-2' of Table 9 are all tested on 10 randomly selected LPs in the test set. With our proposed strategy, the solving time on SC-2 is reduced to around 0.5 hours, which is an acceptable time in practice. Overall, our proposed approach consistently reduces the iteration and time on all datasets for primal simplex method.

C. Combine CPLEX Crash and GNN predicted basis

CPLEX Crash (CA) constructs a basis with better numerical properties. It is composed of 3 steps. **1).** Add the slacks for all one-side inequality constraints to the basis. **2).** Sort the variables according to heuristic penalties. **3).** Visit each variable by order and select the variables that maintain triangularity of $[A_{\mathcal{B}_x} \quad - I^m_{\mathcal{B}_s}]$. In this section, we will enhance the heuristic step **2)** through GNN prediction.

Denote by $q \in \mathbb{R}^n$ the penalty (Bixby, 1992) for the variables. A variable is heuristically more free and of higher priority if its penalty is smaller. Denote by $p = [\mathbb{P}(l^x_1 < x_1 < u^x_1), \cdots, \mathbb{P}(l^x_n < x_n < u^x_n)]$ the probability of variables being basic given by GNN prediction. Since the penalty q and negative probability $-p$ is of different scale, z-score normalization is applied to two vectors. The variable will be selected according to a new penalty $q' = \lambda \times zscore(-p) + (1-\lambda) \times zscore(q)$, where λ is the weight for GNN prediction.

Figure 6 shows the speedup v.s. the weight of GNN prediction. Speedup on iterations is defined as $\frac{\text{Iterations of DEFAULT - Iterations of X}}{\text{Iterations of DEFAULT}}$, where X can be CA or GNN. The speedup on time is defined similarly. Since GEN and LIBSVM have no equality constraint and no artificial variables, CA plays no role in them. Thus, we skip these datasets. Figure 6 shows only the curves on Mirp, STOCH, SC-1, and SC-2. The advantage of step **3)** in CPLEX Crash is that $[A_{\mathcal{B}_x} \quad - I^m_{\mathcal{B}_s}]$ will be close to triangular, and thus requires almost no factorization and accelerating the initial several iterations. However, a basis with good numerical properties is not necessarily closer to optimal, and thus the enhanced CPLEX Crash (blue line) is still inferior to just using GNN prediction (orange line).

On all datasets, the enhanced CPLEX Crash ($\lambda > 1$) is better than the original CPLEX Crash ($\lambda = 1$). This verifies that GNN predictions are better than CPLEX Crash's heuristic. However, integrating GNN prediction into CPLEX Crash is always inferior to purely relying on GNN prediction. This is because step 1) and 3) in CPLEX crash can be incorrect and the initial basis get far away from optimal basis.

D. Pseduo Code

Algorithm 1 Smart Initial Basis Selection Algorithm for Linear Programs

```
**Training Stage**: Given the historical LPs, train a basis prediction model.
**Input:** Past solved LPs set $\mathcal{D} = \{[(P^k), (x^k, s^k)]\}_{k=1}^{K}$. Here, $(P^k)$ is the $k$-th problems and $(x^k, s^k)$ is its optimal solution.
**Output:** A GNN model $f(\theta; \cdot)$.
// Constructed trainset
for $[(P^k), (x^k, s^k)] \in \mathcal{D}$ do
    // Construct graph representation from past solved LP
    1. Given past solved $(P^k)$, extract the LP data, including cost vector $c$, bounds vectors $l^x, l^u, l^s, l^u$, and constraint matrix $A$.
    2. From LP data, construct constraint nodes features $\{w_j\}_{j=1}^m$ and variable nodes features $\{v_i\}_{i=1}^n$. The edge $E_{(w_j, v_i)}$ is connected if $A_{ji} \neq 0$ and weighted by $A_{ji}$.
    // Construct labels
    3. Given $(x^k, s^s)$, transform it into one-hot labels $\{y_{x,i}, y_{s,j} \in \mathsf{OneHot}(3) \mid i \in [n], j \in [m]\}$.
end for
Train $f(\theta; \cdot)$ with constructed trainset: $\theta^* \leftarrow \text{argmin}_\theta \mathcal{L}_{\mathcal{D}}(\theta)$
return $f(\theta; \cdot)$

**Inference Stage**: Given a testing LP, predict an initial basis for it.
**Input:** Testing LP $(P^{\text{test}})$
**Output:** Predicted basis $(\mathcal{B}'_x, \mathcal{B}'_s)$
// Basis generation
Model inference and obtain basis status probability $\{p_{x,i}, p_{s,j} \mid i \in [n], j \in [m]\}$.
Select top-$m$ likely entries into basis $\mathcal{B}_x, \mathcal{B}_s$ according to $\mathbb{P}(\ell_i^x < x_i < u_i^x)$ and $\mathbb{P}(\ell_j^s < s_j < u_j^s)$.
// Basis adjustment
Adjust the basis to make the corresponding basis matrix non-singular. Determine statuses for the non-basic entries.
return $(\mathcal{B}'_x, \mathcal{B}'_s)$

function Define GNN model $f\left(\theta; \{v_i\}_{i=1}^n, \{w_j\}_{j=1}^m\right)$
    1. Initialize the messages by nodes features: $\forall i \in [n], v_i^0 \leftarrow v_i$ and $\forall j \in [m], w_j^0 \leftarrow w_j$.
    2. Conduct $L$ steps message passing with learnable message-passing functions $f_l^V$ and $f_l^W$:
    for $l = 1, \ldots, L$ do
        $v_i^l = f_l^V\left(\theta_l^V; v_i^{l-1}, \sum_{j=1}^m E_{(v_i,w_j)} w_j^{l-1}\right)$ and $w_i^l = f_l^W\left(\theta_l^W; w_i^{l-1}, \sum_{i=1}^n E_{(v_i,w_j)} v_i^{l-1}\right)$.
    end for
    // Knowledge masking: produce probability satisfying the feasibility of non-basic entries
    3. $p_{x,i} = \mathsf{softmax}(v_i^L + h_{x,i})$ and $p_{s,j} = \mathsf{softmax}(w_j^L + h_{s,j})$, where $h_{x,i}$ and $h_{s,j}$ are the large penalty for unreachable bounds.
    return $\{p_{x,i}, p_{s,j} \mid i \in [n], j \in [m]\}$
end function
```