PDHG-Unrolled Learning-to-Optimize Method for Large-Scale Linear Programming

Bingheng Li [*1] **Linxin Yang** [*2 3] **Yupeng Chen** [3] **Senmiao Wang** [2] **Haitao Mao** [1] **Qian Chen** [3 4] **Yao Ma** [5] **Akang Wang** [2 3] **Tian Ding** [3] **Jiliang Tang** [1] **Ruoyu Sun** [2 3]

Abstract

Solving large-scale linear programming (LP) problems is an important task in various areas such as communication networks, power systems, finance, and logistics. Recently, two distinct approaches have emerged to expedite LP solving: (i) First-order methods (FOMs); (ii) Learning to optimize (L2O). In this work, we propose a FOM-unrolled neural network (NN) called PDHG-Net, and propose a two-stage L2O method to solve large-scale LP problems. The new architecture PDHG-Net is designed by unrolling the recently emerged PDHG method into a neural network, combined with channel-expansion techniques borrowed from graph neural networks. We prove that the proposed PDHG-Net can recover PDHG algorithm, thus can approximate optimal solutions of LP instances with a polynomial number of neurons. We propose a two-stage inference approach: first use PDHG-Net to generate an approximate solution, and then apply the PDHG algorithm to further improve the solution. Experiments show that our approach can significantly accelerate LP solving, achieving up to a 3× speedup compared to FOMs for large-scale LP problems.

1. Introduction

Linear programming (LP) (Luenberger et al., 1984; Dantzig, 1998; Schrijver, 1998) has wide applications in various areas such as commnication systems, power systems, logistics and finance (Charnes & Cooper, 1957; Rabinowitz, 1968; Garver, 1970; Dorfman et al., 1987). In the era of big data, the size of LP problems has increased dramatically, leading to a growing interest in accelerating and scaling up LP algorithms (Todd, 1983; Bixby et al., 1992; Basu et al., 2020; Applegate et al., 2021; Deng et al., 2022; Hinder, 2023). Classical algorithms such as the interior-point methods do not scale up due to the expensive per-iteration cost of solving linear systems. Recently, first-order methods (FOMs) such as Primal-Dual Hybrid Gradient (PDHG) have shown considerable potential in solving large-scale LP problems (Chambolle & Pock, 2016; Applegate et al., 2021; Fercoq, 2021; Lu & Yang, 2022; Applegate et al., 2023; Lu & Yang, 2023a;c).

Alongside first-order methods, "learning to optimize" (L2O) paradigm emerges as a promising approach to speed up optimization algorithms (Nair et al., 2020; Bengio et al., 2021; Chen et al., 2022a). L2O is an orthogonal line of pursuit compared to FOMs: L2O utilizes the information of existing problems to help solve new problems, thus can potentially be applied to almost any class of optimization problems. Interestingly, in the development of L2O methods, most efforts have been spent on integer programming, combinatorial optimization, and non-smooth optimization (Gupta et al., 2020; Gasse et al., 2022; Liu & Chen, 2019) but rarely on the more classical problem class LP problems. Only until very recently, there have been some explorations on L2O for LP (Li et al., 2022; Chen et al., 2022b; Kuang et al., 2023; Fan et al., 2023; Liu et al., 2023; Qian et al., 2023).

We notice that L2O-for-LP works have primarily focused on the emulation of interior-point methods (Qian et al., 2023) and simplex methods (Fan et al., 2023; Liu et al., 2023). Motivated by recent progress on FOMs for LP, we ask the following question:

Can we design L2O methods to emulate FOMs to solve large-scale LP problems?

In this paper, we introduce a FOM-unrolled neural network architecture PDHG-Net and a two-stage L2O framework for solving large-scale LP programs. To design the architecture, we unroll the PDHG algorithm into a neural network (NN)

[*]Equal contribution. Work done while visiting Prof. Ruoyu Sun. [1]Michigan State University [2]School of Data Science, The Chinese University of Hong Kong, Shenzhen, China [3]Shenzhen International Center For Industrial and Applied Mathematics, Shenzhen Research Institute of Big Data [4]School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen, China [5]Rensselaer Polytechnic Institute. Correspondence to: Ruoyu Sun <sunruoyu@cuhk.edu.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

by turning the primal and dual updates into NN blocks, and also incorporate the channel-expansion techniques from graph neural networks. The training part is similar to the standard L2O framework as we use a large number of LP instances to train PDHG-Net. The inference part consists of two stages, which is different from the standard L2O framework. In the first stage, we use PDHG-Net to generate an approximately optimal solution. In the second stage, to enhance the solution precision, we refine the solution of PDHG-Net through PDLP. Our inference stage is a hybrid method of neural net and PDLP.

Our main contributions are summarized as follows:

- **PDHG-Net Architecture:** We propose PDHG-Net, a new neural network architecture inspired by the PDHG algorithm. In PDHG-Net, we replace projection functions with ReLU activations and incorporate channel expansion.
- **Theoretical Foundation:** Our theoretical analysis establishes an exact alignment between PDHG-Net and the standard PDHG algorithm. Furthermore, given an approximation error threshold ϵ, we theoretically prove that a PDHG-Net with $\mathcal{O}(1/\epsilon)$ number of neurons can approximate the optimal solution of LP problems to ϵ-accuracy.
- **Two-Stage Solution Framework:** We introduce a two-stage L2O inference framework. In the first stage, PDHG-Net delivers high-quality solutions. In the second stage, we further optimize these solutions by using a PDLP solver. This framework effectively accelerates the solving process while achieving relatively high precision.
- **Empirical Results:** We conduct extensive experiments that showcase the superior performance of our framework in large-scale LP. **Remarkably, our framework achieves up to a 3× speedup compared to PDLP.** Furthermore, we carry out additional experiments aimed at elucidating the mechanisms behind the acceleration, potentially providing insights for future L2O design.

2. PDHG-Net for large-scale LP

2.1. Standard PDHG Algorithm

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

$$\begin{aligned} &\min c^\top x \\ &\text{s.t. } Gx \geq h \\ &\quad\; l \leq x \leq u \end{aligned} \tag{1}$$

where $G \in \mathbb{R}^{m\times n}, h \in \mathbb{R}^m, c \in \mathbb{R}^n, l \in (\mathbb{R} \cup \{-\infty\})^n, u \in (\mathbb{R} \cup \{+\infty\})^n$. Equivalently, we can solve the corresponding saddle point problem:

$$\min_{l\leq x\leq u} \max_{y\geq 0} L(x, y; \mathcal{M}) = c^\top x - y^\top Gx + h^\top y \tag{2}$$

Primal-Dual Hybrid Gradient (PDHG) method, also referred to as the Chambolle-Pock algorithm (Chambolle & Pock, 2011) in optimization literature, is a highly effective strategy for addressing the saddle point problem as outlined in (2). PDHG algorithm alternately updates the primal variable x and the dual variable y to find the saddle point of the Lagrangian L. To be more specific, it has the updating rule as follows.

Primal-Dual Hybrid Gradient (PDHG)

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

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

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

Under regular assumptions, the PDHG algorithm has an ergodic convergence rate of $\mathcal{O}(1/K)$.

Proposition 2.1 (Chambolle & Pock (2016))**.** *Let $(x^k, y^k)_{k\geq 0}$ be the primal-dual variables generated by the PDHG algorithm for the LP problem $\mathcal{M} = (G; l, u, c; h)$. If the step sizes τ, σ satisfy $\tau\sigma\|G\|_2^2 < 1$, then for any $(x, y) \in \mathbb{R}^n \times \mathbb{R}^m_{\geq 0}$ satisfying $l \leq x \leq u$, the primal-dual gap satisfies*

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

where $\bar{x}^k = (\sum_{j=1}^k x^j)/k$, $\bar{y}^k = (\sum_{j=1}^k y^j)/k$, and L is the Lagrangian defined by (2).

2.2. Design of PDHG-Net

In this part, we describe how we design PDHG-Net by unrolling PDHG algorithm and incorporating techniques from graph neural networks.

In the PDHG algorithm, each iteration involves period-2 updates of primal and dual variables, as illustrated in Figure 1. In particular, at the k-th iteration, the algorithm first gets the primal variable x^{k+1} starting from (x^k, y^k), and then gets the dual variable y^{k+1} using the newly obtained x^{k+1} along with y^k. We replace the primal-update and dual-update step with neural network (NN) blocks parameterized by Θ_{p}^k and Θ_{d}^k, respectively.

To provide insights for the design of PDHG-Net blocks, we focus on two components of the PDHG iteration: the proximal operator and the linear combination of the primal-dual variables. These two components respectively lead us to incorporate two distinct approaches in our design: ReLU activation and channel expansion.

Proximal operator – ReLU activation. The proximal operators, specifically the projection functions $\mathbf{Proj}_{y\geq 0}$ and

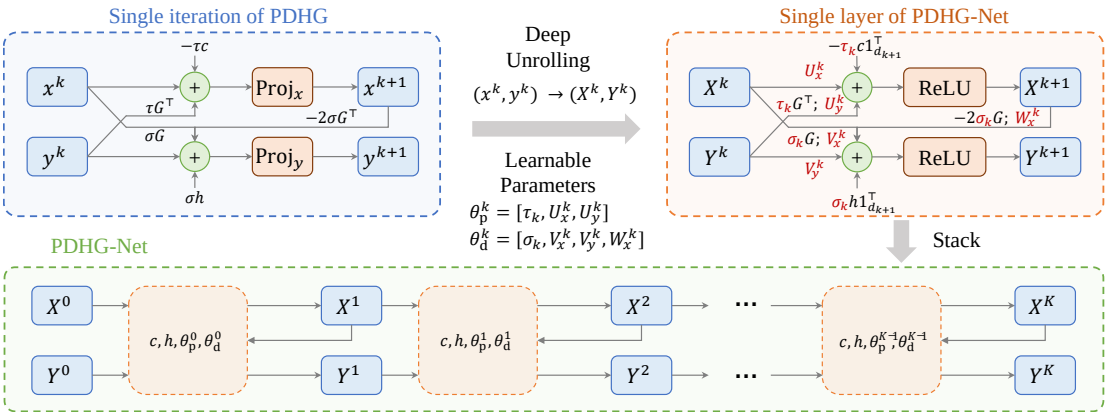


Figure 1. Overview of how each layer in PDHG-Net corresponds to each iteration of the traditional PDHG algorithm, along with the overall architecture of PDHG-Net

$\mathbf{Proj}_{l \le x \le u}$, exhibit a piece-wise linear behavior on each entry. This observation naturally leads to the use of ReLU activations in our network. In the forthcoming Theorem 2.2 and its proof, we will see that this property is useful to help prove that a PDHG-Net can accurately replicate the PDHG algorithm.

Channel expansion. In this part, we focus on the primal updates as an illustrative example and the analysis of the dual updates can be derived similarly. We expand the n-dimensional vectors x^k, y^k into $(n \times d_k)$-dimensional matrices X^k, Y^k with d_k columns (or called channels following the convention of neural network literature). The linear combination $x^k - \tau(c - G^\top y^k)$ of primal-dual is replaced by $X^k U_x^k - \tau_k(c \cdot \mathbf{1}_{d_{k+1}}^\top - G^\top Y^k U_y^k)$ where $\Theta_p^k = (\tau_k, U_x^k, U_y^k) \in \mathbb{R} \times \mathbb{R}^{d_k \times d_{k+1}} \times \mathbb{R}^{d_k \times d_{k+1}}$ is the trainable parameter of the k-th primal NN block. This design is for the network to generalize to LP instances of different sizes and enhance the expressivity.

- **Generalizability to LP instances of different sizes**: It is important to note that our NN design diverges from classical deep unrolling for sparse coding problems, such as the unrolled/unfolded iterative shrinkage thresholding algorithm (ISTA) (Gregor & LeCun, 2010). Following the principle of classical unrolling, a natural idea would be to unroll $x^k - \tau(c - G^\top y^k)$ to

$$x^k - \tau(c - W^k y^k) \tag{3}$$

  where W^k is trainable matrix. However, for different LP problems, the sizes of the constraint matrices G vary, so the replaced trainable matrix can only have a fixed size of $n \times m$, rendering it unsuitable for applying to LP problems with different sizes. Note that this issue does not exist in ISTA, since the transformation matrix in the ISTA algorithm is solely related to the dictionary matrix, which remains fixed in sparse coding problems. To address the generalizability issue of using the component (3) in neural nets, we expand x^k, y^k into X^k, Y^k with multiple channels and apply a trainable matrix on their right. This method, which is motivated by GNN, makes PDHG-Net generalizable to LP problems with different sizes.
- **Expressivity**: Intuitively, increasing the network width, i.e. the number of channels d_k in our setting, can enhance the network's expressivity. As we demonstrate in the subsequent Theorem 2.2, if d_k's are larger than a certain fixed constant (e.g. 10), PDHG-Net possesses sufficient expressive power to accurately recover iterations of PDHG algorithms.

After we design each NN block, we combine the K primal and K dual blocks by the same pattern that the classical PDHG algorithm forms. Notably, as depicted in Figure 1, the architecture resembles the form of a GNN, updating primal and dual variables in a period-2 manner. There are at least two major differences between PDHG-Net and the common GNN architectures designed for solving LP-based problems (Gasse et al., 2019b; Chen et al., 2022b): first, PDHG-Net employs a period-2 update cycle, contrasting with the period-1 update cycle in GNN. Second, within PDHG-Net's update period, dual iterations undergo twice as many aggregations compared to primal iterations, while variable and constraint nodes share the same number of aggregations in GNN's update period.

Based on the analysis above, we propose the architecture of PDHG-Net formally below.

Architecture of PDHG-Net

– Initialize $X^0 = [x^0, l, u, c]$, $Y^0 = [y^0, h]$

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

$$\left| \begin{aligned} X^{k+1} &= \mathrm{ReLU}\left(X^k U_x^k - \tau_k(c \cdot \mathbf{1}_{d_{k+1}}^\top - G^\top Y^k U_y^k)\right), \\ Y^{k+1} &= \mathrm{ReLU}\left(Y^k V_y^k \right. \\ &\qquad \left. + \sigma_k(h \cdot \mathbf{1}_{d_{k+1}}^\top - 2GX^{k+1}W_x^k + GX^k V_x^k)\right), \end{aligned} \right.$$

– Output $X^K \in \mathbb{R}^n$, $Y^K \in \mathbb{R}^m$

The trainable parameter is $\Theta = \{\Theta_{\mathrm{p}}^k, \Theta_{\mathrm{d}}^k\}_{k=0}^{K-1}$, where $\Theta_{\mathrm{p}}^k = (\tau_k, U_x^k, U_y^k) \in \mathbb{R} \times \mathbb{R}^{d_k \times d_{k+1}} \times \mathbb{R}^{d_k \times d_{k+1}}$ and $\Theta_{\mathrm{d}}^k = (\sigma_k, V_x^k, V_y^k, W_x^k) \in \mathbb{R} \times \mathbb{R}^{d_k \times d_{k+1}} \times \mathbb{R}^{d_k \times d_{k+1}} \times \mathbb{R}^{d_{k+1} \times d_{k+1}}$.

The following theorem reveals the capability of PDHG-Net to reproduce the classical PDHG algorithm. It indicates an adequate expressive power of PDHG-Net.

Theorem 2.2. *Given any pre-determined network depth K and the widths $\{d_k\}_{k \le K-1}$ with $d_k \ge 10$, there exists a K-layer PDHG-Net with its parameter assignment Θ_{PDHG} satisfying the following property: given any LP problem $\mathcal{M} = (G; l, u, c; h)$ and its corresponding primal-dual sequence $(x^k, y^k)_{k \le K}$ generated by PDHG algorithm within K iterations, we have*

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

Theorem 2.2 provides a guarantee of an *exact* alignment between a finite-width PDHG-Net and the standard PDHG algorithm. Combining Proposition 2.1 and Theorem 2.2, we propose an upper bound of the required number of neurons to achieve an approximation error of ϵ.

Theorem 2.3. *Given the approximation error bound ϵ, there exists a PDHG-Net with $\mathcal{O}(1/\epsilon)$ number of neurons and the parameter assignment Θ_{PDHG} fulfilling the following property. For any LP problem $\mathcal{M} = (G; l, u, c; h)$ and $(x, y) \in \mathbb{R}^n \times \mathbb{R}_{\ge 0}^m$ satisfying $l \le x \le u$, it holds that*

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

As previously mentioned, our PDHG-Net is closely related to GNN, but also has some differences due to the unrolling nature. We explain the benefit of our design from a theoretical perspective below.

A recent work Chen et al. (2022b) proves that GNNs possess strong expressive power to represent the optimal solutions of LP problems. The motivation is to understand the power of GNNs commonly used in L2O for optimization problems. The representation power result is rather non-trivial, since the expressive power of GNNs is well-known to be limited by the Weisfeiler-Lehman (WL) test (Xu et al., 2018). Chen et al. (2022b) ingeniously proves that this limitation does not pose an issue for LP, as those LP problems indistinguishable by WL-test are shown to share the same optimal solutions if exist.

Compared to Chen et al. (2022b), our proof of expressive power of PDHG-Net has two advantages. First, the proof is rather concise. This is because PDHG-Net is designed by unrolling the classical PDHG algorithm whose convergence has been confirmed, in contrast to the original design of GNNs (e.g. (Gasse et al., 2019a)) which just applies an existing architecture to optimization problems without much theoretical justification. Thus, the proof of Theorem 2.3 does not rely on analyzing the separation power of WL-test in LP problems, making it more concise. Second, perhaps more importantly, under a given error tolerance ϵ, we demonstrate that the number of neurons required is upper-bounded by $\mathcal{O}(1/\epsilon)$ in PDHG-Net. In contrast, Chen et al. (2022b), while utilizing the universal approximation properties of MLPs, does not explore (at least not explicitly) the relationship between network size and expressive power.

2.3. Network Training

Training dataset. The training dataset is a set of instances denoted by $\mathcal{I} = \{(\mathcal{M}, z^*)\}$. The input of an instance is an LP problem $\mathcal{M} = (G; l, u, c; h)$; the label $z^* = (x^*, y^*)$ is its corresponding near-optimal primal-dual solution, which is acquired by adopting a well-established solver such as PDLP with a given stopping criterion.

Loss Function. Given the parameter $\Theta = \{\Theta_{\mathrm{p}}^k, \Theta_{\mathrm{d}}^k\}_{k=0}^{K-1}$ and the input LP problem $\mathcal{M}$, we denote the output (X^K, Y^K) of the PDHG-Net by $Z^K(\mathcal{M}; \Theta)$. We seek to reduce the distance between $Z^K(\mathcal{M}; \Theta) = (X^K, Y^K)$ and $z^* = (x^*, y^*)$ for each training LP problem. Therefore, we train the PDHG-Net to minimize the ℓ_2 square loss

$$\min_{\Theta} \mathcal{L}_{\mathcal{I}}(\Theta) = \frac{1}{|\mathcal{I}|} \sum_{(\mathcal{M}, z^*) \in \mathcal{I}} \left\| Z^K(\mathcal{M}; \Theta) - z^* \right\|_2^2. \quad (4)$$

Theorem 2.2 indicates the existence of model Θ_{PDHG} that perfectly replicates the PDHG algorithm. This naturally leads us to the conclusion that a well-trained PDHG-Net may speed up the convergence to optimal solutions.

Corollary 2.4. *If Θ_{PDHG} is not the global minima of the loss function $\mathcal{L}_{\mathcal{I}}$, a K-layer PDHG-Net can potentially achieve primal-dual solutions closer to the optimal ones than the solutions generated by the standard PDHG algorithm implemented for K iterations.*

Through the training process, our goal is to find the parameter Θ_0 satisfying

$$\mathcal{L}_{\mathcal{I}}(\Theta_0) \approx \min_{\Theta} \mathcal{L}_{\mathcal{I}}(\Theta) < \mathcal{L}_{\mathcal{I}}(\Theta_{\mathrm{PDHG}}).$$

2.4. Two-Stage Framework

To enhance the solution precision, we propose a two-stage framework that first obtains the predicted solutions, and

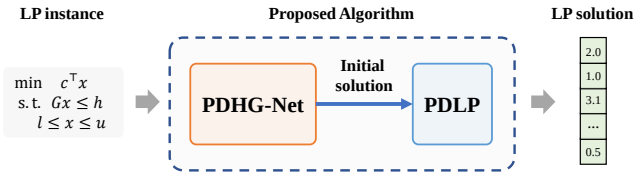


Figure 2. The proposed post-processing procedure warm-starts the PDLP solver using the prediction of PDHG-Net as initial solutions to ensure optimality.

then incorporates a post-processing step to refine these predictions and ensure optimality. For this purpose, we select the PDLP solver from Google's Ortools (Perron & Furnon), renowned for its exemplary implementation of the PDHG algorithm, as the key element in our post-processing module. As demonstrated in Figure 2, this module involves using the predicted solutions to warm-start the PDLP solver, thereby improving the efficiency in solving the original problem. Specifically, given an LP instance, we first feed it into the well-trained PDHG-Net to acquire the prediction of primal and dual solutions $\{X^K, Y^K\}$. Then, we warm-start PDLP by setting $x^0 = X^K$ and $y^0 = Y^K$, and let the solver iteratively update primal and dual solutions, until the termination criteria are met. For comprehensive details on the implementation, including features and training settings, please see Appendix D.

3. Implementation Settings

Datasets. We conduct comprehensive experiments on a selection of representative LP instances grounded in real-world scenarios. Specifically, our experiments involve the PageRank dataset for large-scale LP problems, as well as the Item Placement (`IP`) and the Workload Appropriation (`WA`) datasets (Gasse et al., 2022) for complex LP relaxations. Particularly, the PageRank dataset is acquired by generating instances in accordance with the methodology outlined in (Applegate et al., 2021). For the `IP` and `WA` datasets, the generation protocol is adjusted to ensure that each instance presents a significant computational challenge. To ensure broad applicability, each benchmark dataset encompasses instances across a spectrum of problem sizes. Detailed information on the generation settings and specific problem sizes is available in the Appendix E.

Software and Hardware settings. For the performance comparisons, we employ the off-the-shelf implementation of PDLP from Google's OrTools 9.8.3296 (Perron & Furnon). Gurobi 10.0.2's LP solvers (Gurobi Optimization, LLC, 2023) are also employed for experiments on the PageRank dataset. Our proposed framework was implemented utilizing PyTorch version 2.1.1. (Paszke et al., 2019), and all computational studies are carried out on a cloud server with one NVIDIA Tesla V100-32GB GPU, one Intel Xeon Platinum 8280 CPU and 3,072 GB RAM. For clarity and simplicity in subsequent discussions, we will refer to the performance of PDLP as 'PDLP' and that of our proposed framework as 'ours'. To assess the performance of our proposed framework, we calculate the improvement ratio over PDLP using the following equation:

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

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

Training. Considering the constraints of computing resources and the need for computational efficiency, we utilize a 4-layer PDHG-Net to strike an optimal balance between network performance and inference costs. Given the training dataset $\mathcal{I} = \{(\mathcal{M}_1, z_1^*), \ldots, (\mathcal{M}_{|\mathcal{I}|}, z_{|\mathcal{I}|}^*)\}$, we extract key information and encode it into each $\mathcal{M}_i$ to form $\bar{\mathcal{M}}_i$, as detailed in Appendix D. The dataset $\{(\bar{\mathcal{M}}_1, z_1^*), \ldots, (\bar{\mathcal{M}}_{|\mathcal{I}|}, z_{|\mathcal{I}|}^*)\}$ is then split into training and validation sets with a $9:1$ ratio. The training process entailed minimizing the ℓ_2 square loss, as described in Equation 4, and updating the trainable parameter Θ through backward propagation, where we utilize an Adam optimizer with a learning rate of 10^{-4}. After a certain number of iterations, the model exhibiting the lowest validation loss was selected as the output model.

4. Numerical Experiments

In this section, we embark on a detailed empirical examination of our proposed PDHG-based two-stage framework's effectiveness in managing complex linear programs. Initially, we showcase the framework's performance in tackling large-scale LP problems. We then showcase the framework's capability to produce solutions for LP instances, which typically entail considerable convergence time for FOMs. Following this, we conduct a series of experiments aimed at uncovering the underlying factors contributing to the PDHG-Net's remarkable performance. Finally, a detailed investigation of experiments is undertaken to assess the framework's generalization power and scalability. For reproducibility, our code can be found at `https://github.com/NetSysOpt/PDHG-Net.git`.

Table 1. Solve time comparison between the proposed framework and vanilla PDLP on PageRank instances. The improvement ratio of the solving time is also reported.

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

4.1. Comparing against vanilla PDLP on large-scale LP problems

In this section, we illustrate how our proposed framework improves the efficiency of PDLP in managing large-scale LP problems through comprehensive experiments conducted on the PageRank dataset across a range of sizes. Specifically, we benchmark our approach against the vanilla PDLP solver, and report the solving time along with the improvement ratio brought by our framework.

Detailed numerical results are presented in Table 1. A comparative analysis of the solving time reveals a notable efficiency of our proposed framework: on average, it solves the PageRank problem 64.3% faster than PDLP. Importantly, as the problem size increases, our proposed framework not only sustains a consistent level of performance but also demonstrates a steadily increasing improvement in efficiency. Particularly, our approach consistently outperforms standard PDLP on problems with more than 10^5 nodes, solving instances up to three times faster. Such observations not only underscore the inherent efficacy of PDLP in managing large-scale LP problems, but also highlight the significant enhancements our framework brings to these capabilities.

The significant performance improvement between our proposed framework and the vanilla PDLP can be elucidated by two key factors: **(1)** PDLP typically begins the solving process by initializing both primal and dual solutions as zero vectors. This approach overlooks the distinct characteristics of the specific LP instance. Conversely, our model strategically utilizes the unique aspects of the problem, generating initial points that are specifically tailored for the given instance. **(2)** The initial solution proposed by our model closely approximates the optimal solution, which likely reduces the number of iterations needed to achieve optimality, thereby significantly hastening the solving process. The collected data from these experiments distinctly demonstrate that our proposed method consistently outperforms both PDLP and Gurobi.

4.2. Comparing against PDLP on difficult linear relaxations

FOM-based methods, including PDLP, often experience slow convergence when applied to complex LP problems with stringent constraints. The objective of this section is to demonstrate how our proposed approach can overcome these challenges by providing high-quality initial solutions. Our evaluation encompasses two datasets, featuring LP relaxations of challenging Mixed Integer Programs: `IP` and `WA`. We conduct a comparative analysis of our approach's performance against the default PDLP. It's important to note that the solving precision was set at 10^{-4} to accommodate PDLP's precision limitations, ensuring comparisons stay within its operational boundaries.

In Table 2, we comprehensively present our numerical experiments, comparing our proposed framework with traditional PDLP. The table is organized into two sections: the upper part focuses on solving time as the primary metric, while the lower part assesses the number of iterations required for solving the problem. The data from Table 2 reveals significant efficiency enhancements of our framework in comparison to PDLP. In the `IP` dataset, our framework shows a notable improvement of 19.5% in both solving time and the number of iterations for smaller instances, compared to PDLP. This enhancement becomes even more pronounced in larger `IP` problems, with a 21.7% reduction in solving time and a 27.8% decrease in the number of iterations. These results align with PDLP's growing proficiency in handling larger LP instances. Additionally, the performance of our framework on the `WA` dataset also consistently surpasses that of PDLP. Furthermore, an increase in efficacy is also observed as the size of the problems in the `WA` dataset grows, further demonstrating the capability of our approach. Such consistency is evident across different instance sizes within both datasets, as detailed in Table 2, indicating that our model's efficiency is robust and not negatively impacted by variations in instance size. These findings underscore our proposed framework's ability to effectively tackle complex problems that are typically formidable for conventional FOMs.

Table 2. Results of comparing the proposed framework against default PDLP in solving `IP` and `WA` instances. The top and bottom sections present the solving time and number of iterations, respectively, where bold fonts represent better performance.

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

4.3. Understandings of why PDHG-Net works

In this section, we explore the underlying factors of how our proposed framework enhances the solving of LP problems through experiments. Specifically, we address the following questions:

Q1: Can the proposed PDHG-Net imitate the PDHG algorithm?

Q2: Why is our method capable of accelerating the solving process of PDLP solver?

Align with the PDHG (Q1). In Figure 3, we report the average ℓ_2 distance [1] between the predicted solution $Z^K(\mathcal{M};\Theta) = (X^K, Y^K)$ and the obtained optimal solu-

[1]During the network's training phase, the ℓ_2 square loss is employed as the loss function. For the evaluation of experimental

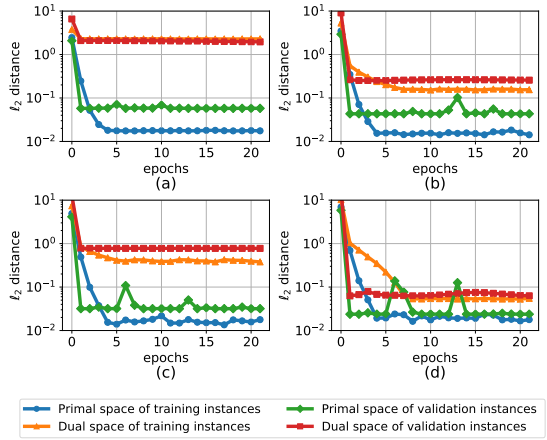
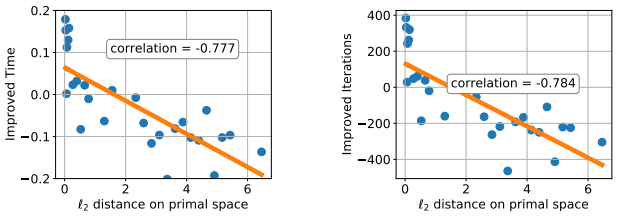


Figure 3. The distance between the predicted solution of PDHG-Net and optimal solution in PageRank training and validation instances with (a) 5×10^3, (b) 1×10^4, (c) 2×10^4, (d) 4×10^4 variable sizes.



(a) Solving time (b) Number of iterations

Figure 4. We present the improvement ratio in both solving time and the number of iterations for solutions extrapolated at varying distances from the optimal solution. Each blue dot symbolizes an extrapolated solution, while the yellow line represents the trend line fitted through these points. Results demonstrate a strong correlation.

tion $z^* = (x^*, y^*)$ for training and validation sets during the training process.

Our experimental results reveal that the solutions inferred by PDHG-Net increasingly converge to the optimal solutions throughout training, ultimately achieving a markedly small discrepancy for both primal and dual solutions. This consistency underscores that the proposed PDHG-Net aligns well with the PDHG algorithm. Furthermore, the validation ℓ_2 distances, maintaining the same order of magnitude across unseen instances, attest to our model's robust generalization ability.

Accelarating PDLP solving process (Q2). To explore the mechanisms behind our framework's improved solving efficiency, we examine the PDHG-Net on the PageRank dataset. By extrapolating PDHG-Net's output $Z^K(\mathcal{M}; \Theta) = (X^K, Y^K)$, from the collected optimal solution $z^* = (x^*, y^*)$, we generate solutions that vary in their proximity to z^*. In Figure 4, we illustrate the improvement ratio of warm-starting PDLP with these solutions in solving time and numbers of iterations compared against the vanilla PDLP solver. The findings demonstrate a strong correlation between solving performance and the extent of extrapolation, which is consistent with Proposition 2.1. Therefore, the proposed PDHG-Net accelerates the solving process by providing high-quality initial solutions.

In addition, we evaluate our framework's efficiency through the number of restarts required to solve the problem. The vanilla PDLP solver performs restarts under specific conditions to enhance performance and prevent stalling (Lu & Yang, 2023c). By comparing our framework to the standard PDLP solver on PageRank instances of differing sizes, we present the number of restarts during the solution process for both approaches in Table 3. The findings indicate a significant decrease in the number of restarts when PDLP is warm-started with predictions from PDHG-Net. This suggests that our framework enables a more efficient optimization trajectory, thus accelerating the problem-solving process.

Table 3. The average number of restarts in the PDLP solving process with our framework (ours) and default settings (represented by PDLP).

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

Therefore, we conclude that our framework markedly enhances the solving process by warm-starting the solver with high-quality solutions and facilitating a more efficient gap-closing approach.

4.4. Generalization on different sizes

We investigate the capability of the proposed method to generalize beyond the scope of its training set, focusing on its adaptability to larger problems. For this purpose, we utilize models trained on the PageRank, `IP`, and `WA` datasets to generate initial solutions for problems of larger scale.

We present a comprehensive overview of our model's generalization capabilities in Table 4, where we compare its solving time and number of iterations against those of the standard PDLP. This comparative analysis underscores that our model consistently surpasses vanilla PDLP, registering an average acceleration of 31.1% across these datasets. Notably, in the PageRank dataset, our model achieves performance levels comparable to those models trained on datasets

results, we measure the ℓ_2 distance between the predicted and optimal solutions, which corresponds to the square root of the ℓ_2 square loss.

Table 4. Solving time and number of iterations for PageRank, IP and WA instances larger than training set sizes. For clarity, we denote the size of the largest instance of IP and WA datasets in Table 2 as Large.

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

of identical size. Similarly, this performance improvement is evident when considering the metric of the number of iterations. As indicated in the bottom section of Table 4, our model, on average, reduces the number of required iterations to reach optimality by approximately 31.8%. These observations collectively underscore the robustness of our proposed framework, particularly in its ability to generalize effectively across a diverse range of problem sizes. On the other hand, the observed consistency in performance may be attributed to the model's proficiency in identifying and utilizing common features inherent in a series of similar LP instances.

4.5. Scalability of PDHG-Net

In this section, we evaluate the scalability of our proposed framework by analyzing the proportion of PDHG-Net inference time within the entire solving process. We gathered both the inference (GPU time) and the solving time (CPU time) for PDLP after warm-starting it with predictions given by PDHG-Net on PageRank instances of varying sizes. To simplify our analysis, we introduce a new metric termed 'ratio' to quantify the proportion of GPU time in relation to the total evaluation time, which is defined as ratio $= \frac{\text{GPU time}}{\text{Search time}}$. The findings in Table 5 reveal that PDLP-Net's inference time constitutes only a minor fraction of the total solving time. Notably, as the size of the dataset's variables increases, the share of GPU time for PDLP-Net inference markedly diminishes. This trend underscores our method's efficiency in scaling up to large-scale datasets, rendering the GPU time increasingly insignificant.

4.6. Comparing PDHG-Net Against GNN

In previous sections, we distinguish the PDHG-Net from Graph Neural Networks (GNN) based on network architectures. Another compelling aspect involves a direct performance comparison of the proposed framework when implemented using these two networks. To ensure fairness,

Table 5. Comparison of total solving time and GPU time for initial solutions, including the ratio of GPU time to total solving time.

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

Table 6. Comparison of improvement ratio and ℓ_2 distance between the proposed framework implemented with PDHG-Net and GNN.

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

models generating the initial solutions were trained under identical settings. In Table 6, we present the numerical results on the PageRank dataset. The performance is evaluated in two dimensions: the improvement ratio in solving time relative to vanilla PDLP and the ℓ_2 distance from the optimal solution. The results clearly indicate that the PDHG-Net implementation not only enhances solving efficiency but also achieves lower ℓ_2 distance in predictions. Moreover, the ℓ_2 distance between GNN's predictions and optimal solutions deteriorates with increasing problem size. This observation indicates that GNN lacks the inherent robustness of PDHG in managing instances of varied sizes, where the performance of PDHG-Net highlights a relatively stable performance on diverse problem dimensions. This study demonstrates PDHG-Net's superior efficacy in addressing challenging linear programming compared to GNN.

5. Conclusions and Future Direction

Inspired by recent advances in FOMs for LP problems and deep unrolling, this work introduces PDHG-Net, based on the PDHG algorithm, and proves that PDHG-Net can represent LP optimal solutions approximately. Building upon this architecture, we propose a two-stage inference framework for solving large-scale LP problems. Comprehensive experiments demonstrate that our proposed method significantly improves the speed of solving large-scale LP problems when there are enough historical data. This indicates the potential of applying L2O to solving large-scale LP instances. Our result also sheds light on the architecture design in L2O for other problems such as mixed integer programming (MIP): while GNN has been the architecture of choice in L2O for MIP, we propose a new architecture that has some similarities with GNN but enjoys a strong theoretical guarantee. A future direction is to explore our PDHG-Net in L2O for MIP.

Acknowledgments

Bingheng Li, Linxin Yang, Yupeng Chen, Senmiao Wang, Qian Chen, Akang Wang, Tian Ding, and Ruoyu Sun are supported by the National Key R&D Program of China under grant 2022YFA1003900; Hetao Shenzhen-Hong Kong Science and Technology Innovation Cooperation Zone Project (No.HZQSWS-KCCYB-2024016); University Development Fund UDF01001491, the Chinese University of Hong Kong, Shenzhen; Guangdong Provincial Key Laboratory of Mathematical Foundations for Artificial Intelligence (2023B1212010001); Longgang District Special Funds for Science and Technology Innovation (LGKCSDPT2023002). We also thank Prof. Haihao Lu for his invaluable assistance with the use of the PDLP solver.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

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. *Advances in Neural Information Processing Systems*, 34: 20243–20257, 2021.

Applegate, D., Hinder, O., Lu, H., and Lubin, M. Faster first-order primal-dual methods for linear programming using restarts and sharpness. *Mathematical Programming*, 201(1-2):133–184, 2023.

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

Beck, A. *First-order methods in optimization*. SIAM, 2017.

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

Bixby, R. E., Gregory, J. W., Lustig, I. J., Marsten, R. E., and Shanno, D. F. Very large-scale linear programming: A case study in combining interior point and simplex methods. *Operations Research*, 40(5):885–897, 1992.

Chambolle, A. and Pock, T. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of mathematical imaging and vision*, 40:120–145, 2011.

Chambolle, A. and Pock, T. On the ergodic convergence rates of a first-order primal–dual algorithm. *Mathematical Programming*, 159(1-2):253–287, 2016.

Charnes, A. and Cooper, W. W. Management models and industrial applications of linear programming. *Management science*, 4(1):38–91, 1957.

Chen, T., Chen, X., Chen, W., Heaton, H., Liu, J., Wang, Z., and Yin, W. Learning to optimize: A primer and a benchmark. *Journal of Machine Learning Research*, 23 (189):1–59, 2022a.

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

Dantzig, G. B. Origins of the simplex method. In *A history of scientific computing*, pp. 141–151. 1990.

Dantzig, G. B. *Linear programming and extensions*, volume 48. Princeton university press, 1998.

Deng, Q., Feng, Q., Gao, W., Ge, D., Jiang, B., Jiang, Y., Liu, J., Liu, T., Xue, C., Ye, Y., et al. New Developments of ADMM-based Interior Point Methods for Linear Programming and Conic Programming. *arXiv preprint arXiv:2209.01793*, 2022.

Dorfman, R., Samuelson, P. A., and Solow, R. M. *Linear programming and economic analysis*. Courier Corporation, 1987.

Fan, Z., Wang, X., Yakovenko, O., Sivas, A. A., Ren, O., Zhang, Y., and Zhou, Z. Smart initial basis selection for linear programs. In *International Conference on Machine Learning*, pp. 9650–9664. PMLR, 2023.

Fercoq, O. Quadratic error bound of the smoothed gap and the restarted averaged primal-dual hybrid gradient. 2021.

Garver, L. L. Transmission network estimation using linear programming. *IEEE Transactions on power apparatus and systems*, (7):1688–1697, 1970.

Gasse, M., Chételat, D., Ferroni, N., Charlin, L., and Lodi, A. Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, 32, 2019a.

Gasse, M., Chételat, D., Ferroni, N., Charlin, L., and Lodi, A. Exact Combinatorial Optimization with Graph Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 32*, 2019b.

Gasse, M., Bowly, S., Cappart, Q., Charfreitag, J., Charlin, L., Chételat, D., Chmiela, A., Dumouchelle, J., Gleixner, A., Kazachkov, A. M., et al. The machine learning for combinatorial optimization competition (ml4co): Results

and insights. In *NeurIPS 2021 Competitions and Demonstrations Track*, pp. 220–231. PMLR, 2022.

Gondzio, J. Interior point methods 25 years later. *European Journal of Operational Research*, 218(3):587–601, 2012.

Gregor, K. and LeCun, Y. Learning fast approximations of sparse coding. In *Proceedings of the 27th international conference on international conference on machine learning*, pp. 399–406, 2010.

Gupta, P., Gasse, M., Khalil, E., Mudigonda, P., Lodi, A., and Bengio, Y. Hybrid models for learning to branch. *Advances in neural information processing systems*, 33: 18087–18097, 2020.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL https://www.gurobi.com.

Hinder, O. Worst-case analysis of restarted primal-dual hybrid gradient on totally unimodular linear programs. *arXiv preprint arXiv:2309.03988*, 2023.

Kim, D. and Fessler, J. A. Optimized first-order methods for smooth convex minimization. *Mathematical programming*, 159:81–107, 2016.

Kuang, Y., Li, X., Wang, J., Zhu, F., Lu, M., Wang, Z., Zeng, J., Li, H., Zhang, Y., and Wu, F. Accelerate Presolve in Large-Scale Linear Programming via Reinforcement Learning, 2023.

Li, X., Qu, Q., Zhu, F., Zeng, J., Yuan, M., Mao, K., and Wang, J. Learning to Reformulate for Linear Programming, 2022.

Lin, T., Ma, S., Ye, Y., and Zhang, S. An ADMM-based interior-point method for large-scale linear programming. *Optimization Methods and Software*, 36(2-3):389–424, 2021.

Liu, J. and Chen, X. ALISTA: Analytic weights are as good as learned weights in LISTA. In *International Conference on Learning Representations (ICLR)*, 2019.

Liu, T., Pu, S., Ge, D., and Ye, Y. Learning to Pivot as a Smart Expert. *arXiv preprint arXiv:2308.08171*, 2023.

Lu, H. and Yang, J. On the Infimal Sub-differential Size of Primal-Dual Hybrid Gradient Method. *arXiv preprint arXiv:2206.12061*, 2022.

Lu, H. and Yang, J. cuPDLP. jl: A GPU implementation of restarted primal-dual hybrid gradient for linear programming in Julia. *arXiv preprint arXiv:2311.12180*, 2023a.

Lu, H. and Yang, J. On the Geometry and Refined Rate of Primal-Dual Hybrid Gradient for Linear Programming. *arXiv preprint arXiv:2307.03664*, 2023b.

Lu, H. and Yang, J. A Practical and Optimal First-Order Method for Large-Scale Convex Quadratic Programming. *arXiv preprint arXiv:2311.07710*, 2023c.

Lu, H. and Yang, J. On a Unified and Simplified Proof for the Ergodic Convergence Rates of PPM, PDHG and ADMM. *arXiv preprint arXiv:2305.02165*, 2023d.

Lu, H., Yang, J., Hu, H., Huangfu, Q., Liu, J., Liu, T., Ye, Y., Zhang, C., and Ge, D. cuPDLP-C: A strengthened implementation of cuPDLP for linear programming by C language. *arXiv preprint arXiv:2312.14832*, 2023.

Luenberger, D. G., Ye, Y., et al. *Linear and nonlinear programming*, volume 2. Springer, 1984.

Nair, V., Bartunov, S., Gimeno, F., Von Glehn, I., Lichocki, P., Lobov, I., O'Donoghue, B., Sonnerat, N., Tjandraatmadja, C., Wang, P., et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

Necoara, I., Nesterov, Y., and Glineur, F. Linear convergence of first order methods for non-strongly convex optimization. *Mathematical Programming*, 175:69–107, 2019.

Nesterov, Y. Subgradient methods for huge-scale optimization problems. *Mathematical Programming*, 146(1-2): 275–297, 2014.

Nesterov, Y. and Nemirovskii, A. *Interior-point polynomial algorithms in convex programming*. SIAM, 1994.

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

O'Donoghue, B., Chu, E., Parikh, N., and Boyd, S. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, 2016.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An Imperative Style, High-Performance Deep Learning Library, 2019.

Perron, L. and Furnon, V. OR-Tools. URL https://developers.google.com/optimization/.

Qian, C., Chételat, D., and Morris, C. Exploring the Power of Graph Neural Networks in Solving Linear Optimization Problems. *arXiv preprint arXiv:2310.10603*, 2023.

Rabinowitz, P. Applications of linear programming to numerical analysis. *SIAM Review*, 10(2):121–159, 1968.

Renegar, J. Efficient first-order methods for linear programming and semidefinite programming. *arXiv preprint arXiv:1409.5832*, 2014.

Schrijver, A. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

Todd, M. J. Large-scale linear programming: Geometry, working bases and factorizations. *Mathematical Programming*, 26:1–20, 1983.

Wang, S. and Shroff, N. A new alternating direction method for linear programming. *Advances in Neural Information Processing Systems*, 30, 2017.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How Powerful are Graph Neural Networks? In *International Conference on Learning Representations*, 2018.

Ye, Y. *Interior point algorithms: theory and analysis*. John Wiley & Sons, 2011.

Zhang, H. and Sra, S. First-order methods for geodesically convex optimization. In *Conference on Learning Theory*, pp. 1617–1638. PMLR, 2016.

A. Related Work

A.1. FOMs for Large-scale LP

First-order methods (FOMs), which are based on gradient information rather than Hessian information, have been widely applied in numerous optimization fields and have become the standard approach (Zhang & Sra, 2016; Kim & Fessler, 2016; Beck, 2017). These methods are noted for their efficiency and simplicity in dealing with a variety of optimization problems, as evidenced by significant works such as (Renegar, 2014; Wang & Shroff, 2017; Necoara et al., 2019). In the realm of linear programming, the Simplex method (Dantzig, 1990) and Interior Point Methods (IPMs) (Nesterov & Nemirovskii, 1994; Ye, 2011; Gondzio, 2012) have garnered attention for their exceptional performance in solving medium and small-sized LP problems. However, achieving additional scalability or acceleration remains a significant challenge. This challenge is particularly pronounced due to the necessity of solving linear systems, which leads to resource-intensive LU decomposition or Cholesky decomposition processes.

Recent research trends in the field of large-scale linear equation systems have shifted focus towards the exploration of FOMs. These methods are particularly attractive due to their low per-iteration cost and parallelization capabilities. Solvers based on FOMs have demonstrated notable success in empirical studies, as evidenced by the work of (O'Donoghue et al., 2016; O'Donoghue, 2021; Basu et al., 2020; Lin et al., 2021; Deng et al., 2022). This shift highlights the growing interest in and effectiveness of FOMs for handling large-scale computational challenges.

It is noteworthy that within the domain of FOMs, the Primal-Dual Hybrid Gradient (PDHG) method has exhibited faster convergence rates by effectively utilizing data matrices for matrix-vector multiplication. This advantage is well-documented in recent literature, including studies by (Applegate et al., 2023; Hinder, 2023; Lu & Yang, 2023b). Specifically, PDHG demonstrates sublinear rates on general convex-concave problems, as shown in (Chambolle & Pock, 2016; Lu & Yang, 2023d), and achieves linear rates in many instances (Fercoq, 2021; Lu & Yang, 2022). From a practical application standpoint, the PDLP, a universal LP solver based on the PDHG algorithm, has achieved state-of-the-art performance in large-scale LP problems (Applegate et al., 2021; Lu & Yang, 2023a; Lu et al., 2023).

A.2. Expressive Power of L2O architectures for LP problems

With the advent of deep learning, learning to optimize (L2O) has become an emerging research area (Nair et al., 2020; Bengio et al., 2021; Chen et al., 2022a). Numerous studies have approached optimization problems as inputs, employing trained parametric models to provide valuable information for traditional optimizers. In the field of LP, these enhancements include advanced pre-processing techniques (Li et al., 2022; Kuang et al., 2023), and the generation of high-quality initial solutions (Fan et al., 2023). Additionally, these models have been utilized to yield some approximate optimal solutions for the LP problems themselves (Chen et al., 2022b; Qian et al., 2023). Nevertheless, these methods are often tightly coupled with traditional algorithms, and their effectiveness is called into question with increasing problem sizes.

B. Proof of Theorem 2.2

Proof. We will prove a stronger claim by mathematical induction: there is a parameter assignment fulfilling the following requirement. For any hidden layer k, there exist (data-independent) matrices $P_x^k, P_y^k \in \mathbb{R}^{d_k \times d_k}$ satisfying the following property. Given any LP problem $\mathcal{M} = (G; l, u, c; h)$ and its corresponding primal-dual sequence $(x^k, y^k)_{k \le K}$ generated by PDHG algorithm within K iterations, the primal embedding X^k and the dual embedding Y^k at the k-th layer satisfy

$$\begin{aligned} X^k P_x^k &= [\bar{x}^k, x^k, l, u, c, \mathbf{0}_n, \dots \mathbf{0}_n] \in \mathbb{R}^{n \times d_k}, \\ Y^k P_y^k &= [\bar{y}^k, y^k, h, \mathbf{0}_n, \dots \mathbf{0}_n] \in \mathbb{R}^{n \times d_k}, \end{aligned} \tag{1}$$

where

$$\bar{x}^k = \frac{1}{k} \sum_{i=1}^{k} x^i, \quad \bar{y}^k = \frac{1}{k} \sum_{i=1}^{k} y^i, \quad 1 \le i \le K.$$

Suppose that G is an $m \times n$ real matrix. If this claim is true, $\bar{x}^k, x^k$ can be recovered by all the channels of X^k through

linear transformations

$$\bar{x}^k = (X^k P_x^k) \cdot e_1 = \underbrace{X^k}_{n\times d_k} \cdot (\underbrace{P_x^k e_1}_{d_k \times 1}),$$
$$x^k = (X^k P_x^k) \cdot e_2 = X^k \cdot (P_x^k e_2),$$

where $e_1 = (1, 0, \ldots, 0)^T$, $e_2 = (0, 1, 0, \ldots, 0)^T$. Similarly, $\bar{y}^h, y^h$ can be recovered through Y^h through

$$\bar{y}^k = (Y^k P_y^k) \cdot e_1 = \underbrace{Y^k}_{m\times d_k} \cdot (\underbrace{P_y^k e_1}_{d_k \times 1}),$$
$$y^k = (Y^h P_y^k) \cdot e_2 = Y^k \cdot (P_y^k e_2).$$

Moreover, the coefficient vectors of these linear channel combinations $P_x^k e_1, P_x^k e_2, P_y^k e_1, P_y^k e_2$ do not rely on $\mathcal{M}$ and $(x^k, y^k)_{k\leq K}$.

Now we prove this claim by mathematical induction.

1. Base case: $k = 0$. This claim is obviously true because the inputs are $X^0 = [x^0, l, u, c]$ and $Y^0 = [y^0, q]$.

2. By the induction hypothesis, we suppose that there is an assignment of parameters before the k-th layer such that (1) holds, then we consider the $(k + 1)$-th layer.

We assume here that l and u are vectors with only finite entries. In practice, the bounds for the (primal) variable x are constrained by the precision limitations of the computing machinery.

First, we will show that X^{k+1} satisfies (1). We define

$$Q_1 := \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & -1 & 0 & 0 \\ \tau & \tau & 0 & 0 & \tau & \tau & \tau & \tau & \tau+1 & \tau-1 \end{bmatrix} \in \mathbb{R}^{5\times 10},$$

$$Q_2 := \begin{bmatrix} 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 \\ \tau & \tau & \ldots & \tau \end{bmatrix} \in \mathbb{R}^{5\times(d_{k+1}-10)}.$$

and

$$Q_3 := \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{4\times 4}.$$

Then we get

$$\begin{aligned} &[\bar{x}^k, x^k, l, u, c] \cdot Q_1 \\ &= [\bar{x}^k + \tau c, -\bar{x}^k + \tau c, x^k - l, x^k - u, l + \tau c, -l + \tau c, u + \tau c, -u + \tau c, (\tau + 1)c, (\tau - 1)c] \in \mathbb{R}^{n\times 10}, \end{aligned}$$

$$[\bar{x}^k, x^k, l, u, c] \cdot Q_2 = [\tau c, \tau c, \ldots \tau c] \in \mathbb{R}^{n\times(d_{k+1}-10)},$$

and

$$[\bar{y}^k, y^k, h, \mathbf{0}_n] \cdot Q_3 = [\mathbf{0}_n, \mathbf{0}_n, y^k, y^k].$$

If we define

$$U_x^k := P_x^k \hat{U}_x^k, \quad U_y^k := P_y^k \hat{U}_y^k, \tag{2}$$

where

$$\hat{U}_x^k = \left[\begin{array}{c|c} Q_1 & Q_2 \\ \hline \mathbf{0}_{(d_k-5)\times 10} & \mathbf{0}_{(d_k-5)\times(d_{k+1}-10)} \end{array}\right] \in \mathbb{R}^{d_k\times d_{k+1}},$$

and

$$\hat{U}_y^k = \left[\begin{array}{c|c} Q_3 & \mathbf{0}_{4\times(d_{k+1}-4)} \\ \hline \mathbf{0}_{(d_k-4)\times 4} & \mathbf{0}_{(d_k-4)\times(d_{k+1}-4)} \end{array}\right] \in \mathbb{R}^{d_k\times d_{k+1}}.$$

By basic linear algebra, we have

$$\begin{aligned} X^k U_x^k = X^k P_x^k \hat{U}_x^k &= [\bar{x}^k + \tau c, -\bar{x}^k + \tau c, x^k - l, x^k - u, \\ &\qquad l + \tau c, -l + \tau c, u + \tau c, -u + \tau c, (\tau+1)c, (\tau-1)c, \tau c, \ldots, \tau c] \in \mathbb{R}^{d_k\times d_{k+1}}, \\ Y^k U_y^k = Y^k P_y^k \hat{U}_y^k &= [\mathbf{0}_n, \mathbf{0}_n, y^k, y^k, \mathbf{0}_n, \ldots, \mathbf{0}_n] \in \mathbb{R}^{d_k\times d_{k+1}}. \end{aligned}$$

We assign $\tau_k = \tau$ and denote $\tilde{x}^k = x^k - \tau(c - G^\top y^k)$, then we get

$$\begin{aligned} & X^k U_x^k - \tau_k(c \cdot \mathbf{1}_{d_{k+1}}^\top - G^\top Y^k U_y^k) \\ =& [\bar{x}^k + \tau c, -\bar{x}^k + \tau c, x^k - l, x^k - u, l + \tau c, -l + \tau c, u + \tau c, -u + \tau c, (\tau+1)c, (\tau-1)c, \tau c, \ldots, \tau c] \\ & \quad - [\tau c, \tau c, \tau c, \tau c, \ldots, \tau c] + \tau G^\top \cdot [\mathbf{0}_n, \mathbf{0}_n, y^k, y^k, \mathbf{0}_n, \ldots, \mathbf{0}_n] \\ =& [\bar{x}^k, -\bar{x}^k, x^k - \tau c + \tau G^\top y^k - l, x^k - \tau c + \tau G^\top y^k - u, l, -l, u, -u, c, -c, \mathbf{0}_n, \ldots, \mathbf{0}_n] \\ =& [\bar{x}^k, -\bar{x}^k, \tilde{x}^k - l, \tilde{x}^k - u, l, -l, u, -u, c, -c, \mathbf{0}_n, \ldots, \mathbf{0}_n], \end{aligned}$$

and further

$$\begin{aligned} X^{k+1} &= \mathrm{ReLU}\big(X^k U_x^k - \tau_k(c \cdot \mathbf{1}_{d_{k+1}}^\top - G^\top Y^k U_y^k)\big) \\ &= [(\bar{x}^k)^+, (\bar{x}^k)^-, (\tilde{x}^k - l)^+, (\tilde{x}^k - u)^+, l^+, l^-, u^+, u^-, c^+, c^-, \mathbf{0}_n, \ldots, \mathbf{0}_n]. \end{aligned}$$

Since

$$\begin{aligned} z &= z^+ - z^-, \\ \mathbf{Proj}_{l\le z\le u}(z) &= l + (z-l)^+ - (z-u)^+, \\ \bar{x}^{k+1} &= \frac{k}{k+1}\bar{x}^k + \frac{1}{k+1}x^{k+1}, \end{aligned} \tag{3}$$

for any $z \in \mathbb{R}^n$, we get

$$\begin{aligned} x^{k+1} &= \mathbf{Proj}_{l\le z\le u}(\tilde{x}^k) \\ &= l + (\tilde{x}^k - l)^+ - (\tilde{x}^k - u)^+ \\ &= l^+ - l^- + (\tilde{x}^k - l)^+ - (\tilde{x}^k - u)^+, \end{aligned} \tag{4}$$

and

$$\begin{aligned} \bar{x}^{k+1} &= \frac{k}{k+1}\bar{x}^k + \frac{1}{k+1}x^{k+1} \\ &= \frac{k}{k+1}(\bar{x}^k)^+ - \frac{k}{k+1}(\bar{x}^k)^- + \frac{1}{k+1}l^+ - \frac{1}{k+1}l^- + \frac{1}{k+1}(\tilde{x}^k - l)^+ - \frac{1}{k+1}(\tilde{x}^k - u)^+. \end{aligned} \tag{5}$$

We define

$$P_x^{k+1} = \left[\begin{array}{c|c} Q_4 & \mathbf{0}_{10\times(d_{k+1}-5)} \\ \hline \mathbf{0}_{(d_{k+1}-10)\times 5} & \mathbf{0}_{(d_{k+1}-10)\times(d_{k+1}-5)} \end{array}\right] \in \mathbb{R}^{d_{k+1}\times d_{k+1}},$$

where

$$Q_4 := \begin{bmatrix} \frac{k}{k+1} & 0 & 0 & 0 & 0 \\ \frac{-k}{k+1} & 0 & 0 & 0 & 0 \\ \frac{1}{k+1} & 1 & 0 & 0 & 0 \\ \frac{-1}{k+1} & -1 & 0 & 0 & 0 \\ \frac{1}{k+1} & 1 & 1 & 0 & 0 \\ \frac{-1}{k+1} & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix} \in \mathbb{R}^{10\times 5}.$$

Combining (3), (4), and (5), we have

$$X^{k+1}P_x^{k+1} = [\bar{x}^{k+1}, x^{k+1}, l, u, c, \mathbf{0}_n, \ldots, \mathbf{0}_n].$$

Obviously, P_x^{k+1} is independent of the LP problem $\mathcal{M} = (G; l, u, c; h)$ and the primal-dual sequence $(x^k, y^k)_{k\le K}$.

Second, we will show that Y^{k+1} satisfies (1). Similar to the induction of X^{k+1}, there exist data-independent matrices $\hat{V}_x^k, \hat{V}_y^k \in \mathbb{R}^{d_k\times d_{k+1}}$, and $\hat{W}_x^k \in \mathbb{R}^{d_{k+1}\times d_{k+1}}$ such that

$$\begin{aligned}
Y^kV_y^k &= Y^kP_y^k\hat{V}_y^k = [\bar{y}^k - \sigma h, -\bar{y}^k - \sigma h, y^k, (1-\sigma)h, -(1+\sigma)h, -\sigma h, \ldots, -\sigma h],\\
X^{k+1}W_x^k &= X^{k+1}P_x^{k+1}\hat{W}_x^{k+1} = [\mathbf{0}_n, \mathbf{0}_n, x^{k+1}, \mathbf{0}_n, \ldots, \mathbf{0}_n],\\
X^kV_x^k &= X^kP_x^k\hat{V}_x^k = [\mathbf{0}_n, \mathbf{0}_n, x^k, \mathbf{0}_n, \ldots, \mathbf{0}_n].
\end{aligned}$$

Here we assign the weight matrices

$$V_x^k = P_x^k\hat{V}_x^k, \qquad V^k = P_y^k\hat{V}_y^k, \qquad W_x^k = P_x^{h+1}\hat{W}_x^k.$$

If we assign $\sigma_k = \sigma$, we get

$$\begin{aligned}
&Y^kV_y^k + \sigma_k(h\cdot\mathbf{1}_{d_{k+1}}^\top - 2GX^{k+1}W_x^k + GX^kV_x^k)\\
&= [\bar{y}^k - \sigma h, -\bar{y}^k - \sigma h, y^k, h - \sigma h, -h - \sigma h, -\sigma h, \ldots, -\sigma h]\\
&\qquad + [\sigma h, \sigma h, \sigma h, \ldots, \sigma h] - \sigma G\cdot[\mathbf{0}_m, \mathbf{0}_m, 2x^{k+1} - x^k, \mathbf{0}_m, \ldots, \mathbf{0}_m]\\
&= [\bar{y}^k, -\bar{y}^k, y^k + \sigma(h - G(2x^{k+1} - x^k)), h, -h, \mathbf{0}_n, \ldots, \mathbf{0}_n]
\end{aligned}$$

and further

$$\begin{aligned}
Y^{k+1} &= \mathrm{ReLU}\big(Y^kV_y^k + \sigma_k(h\cdot\mathbf{1}_{d_{k+1}} - 2GX^{k+1}W_x^k + GX^kV_x^k)\big)\\
&= [(\bar{y}^k)^+, (\bar{y}^k)^-, y^{k+1}, q^+, q^-, \mathbf{0}_n, \ldots, \mathbf{0}_n].
\end{aligned}$$

Similar to the induction of X^{k+1}, there exists a data-independent P_y^{k+1} such that

$$Y^{k+1}P_y^{k+1} = [\bar{y}^{k+1}, y^{k+1}, h, \mathbf{0}_m, \ldots, \mathbf{0}_m].$$

3. By the mathematical induction, we get that the claim (1) holds for any hidden layer. In terms of the output layer K, we only need to replace the linear transformations after the ReLU activations P_x^K and P_y^K with $P_x^Ke_1$ and $P_y^Ke_1$, respectively. These small modifications ensure $X^K = \bar{x}^K$ and $Y^K = \bar{y}^K$.

□

C. Proof of Theorem 2.3

Proof. Given the approximation error bound ϵ, we consider a $K(\epsilon)$-layer PDHG-Net with network width $d_k = 20 \ge 10$. We assign the parameter Θ_{PDHG} as chosen in Theorem 1. By Theorem 1, for any LP problem $\mathcal{M} = (G; l, u, c; h)$ and its corresponding primal-dual sequence $(x^k, y^k)_{k\le K(\epsilon)}$, the network output (X^K, Y^K) is equal to $(\bar{x}^k, \bar{y}^k)$.

Recalling Proposition 2.1, for any $(x, y) \in \mathbb{R}^n \times \mathbb{R}^m_{\ge 0}$ satisfying $l \le x \le u$, we have

$$\begin{aligned}
L(X^K, y; \mathcal{M}) - L(x, Y^K; \mathcal{M}) &= L(\bar{x}^k, y; \mathcal{M}) - L(x, \bar{y}^k; \mathcal{M})\\
&\le \frac{C}{2K(\epsilon)},
\end{aligned}$$

where $C := \frac{\|x-x^0\|^2}{\tau} + \frac{\|y-y^0\|^2}{\sigma} - (y-y^0)^\top G(x-x^0)$ for simplicity.

We only need to take $K(\epsilon) = [C/\epsilon] + 1$, then we get

$$L(X^K, y; \mathcal{M}) - L(x, Y^K; \mathcal{M}) \le \epsilon/2 < \epsilon.$$

Since the network width is a fixed number 20, the number of neurons at each layer is bounded by a constant. Thus, the total number of neurons has the same order as $K(\epsilon)$, which is $\mathcal{O}(1/\epsilon)$.

□

D. Feature extraction

This section illustrates all used features to extract initial encoding of LP instances. We used 3-dimension feature vector for variables, 4-dimensions for constraints and a single scalar value to describe the connection between variables and constraints. Table 7 lists the detailed information of each utilized feature.

	feature	type	dimension	description
Variables	lb	float	1	the lower bound of the corresponding variable
	ub	float	1	the upper bound of the corresponding variable
	c	float	1	the cost coefficient of the corresponding variable
Constraints	ctype	$\{0, 1\}$	3	One-hot encoding of the type of the constraint ($[\leq, =, \geq]$).
	rhs	float	1	the right-hand-side value of the corresponding constraint

Table 7. initial feature encoding of the LP instance.

E. Datasets

E.1. PageRank

The generation of PageRank instances adheres to the methodology described in (Applegate et al., 2021), wherein each problem is formulated into an LP following the approach in (Nesterov, 2014). We employ the identical network configurations as in (Applegate et al., 2021), but vary the number of nodes to create instances of different sizes. Sizes of all utilized instances can be found in Table 8.

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

Table 8. Sizes of utilized PageRank instances.The number of variables, constraints, and non-zeros are reported.

E.2. Item Placement and Workload Appropriation

We adopted the generation code from (Gasse et al., 2022), and modified controlling parameters to obtain instances with complex LP relaxations. Specifically, in the `IP-S` dataset, each item is set to have 100 dimensions with a configuration of 50 bins and 525 items. Conversely, for the `IP-L` dataset, items are set to have 200 dimensions, with an increased scale of 150 bins and 1,575 items. In the `WA-S` and `WA-L` datasets, we enhanced the probability of each task being compatible with corresponding machines, which leads to a much denser model, consequently increasing the complexity of the LP relaxation and making it more challenging to solve. Furthermore, we adjust the quantities of machines and tasks to 800 and 100, and 2,000 and 220, respectively. Sizes of all utilized instances can be found in Table 9.

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

Table 9. Sizes of utilized instances.The average number of variables, constraints, and non-zeros are reported.

F. Comparison with Gurobi

Moreover, it's crucial to point out that Gurobi faced substantial difficulties in resolving instances with node counts exceeding 10^5. This observation further highlights the superior capability of FOMs in efficiently tackling large-scale problems. The collected data from these experiments distinctly demonstrate that our proposed method consistently outperforms both PDLP and Gurobi. Furthermore, we also conducted the comparitive experiments between our framework and the ABIP method, which is an Interior point method accelerated by FOM.

Table 10. Solve time for PageRank instances. Gurobi's LP solvers are set to default and use 1 thread. The precision of PDLP is set to the 10^{-8} relative gap. A time limit of 10 hours is set for all tested approaches, and presolve not applied. The number of nonzero coefficients per instance is $8 \times (\#\text{ nodes}) - 18$.

# nodes	ours	PDLP	Gurobi Barrier	Gurobi P. Simp.	Gurobi D. Simp.	ABIP
10^3	0.01s	0.04s	0.04s	0.1s	0.2s	0.05s
10^4	0.4s	1.1s	273.9s	40.9s	125.8 s	3.73s
10^5	22.4s	71.3s	>15,000s	>15,000s	>15,000s	277.3s
10^6	4,508s	>15,000s	>15,000s	>15,000s	>15,000s	Fail