Practical Large-Scale Linear Programming using Primal-Dual Hybrid Gradient

David Applegate
Google Research
dapplegate@google.com

Mateo Díaz
California Institute of Technology*
mateodd@caltech.edu

Oliver Hinder
Google Research
University of Pittsburgh
ohinder@pitt.edu

Haihao Lu
University of Chicago†
haihao.lu@chicagobooth.edu

Miles Lubin
Google Research
mlubin@google.com

Brendan O'Donoghue
DeepMind
bodonoghue@deepmind.com

Warren Schudy
Google Research
wschudy@google.com

Abstract

We present PDLP, a practical first-order method for linear programming (LP) that can solve to the high levels of accuracy that are expected in traditional LP applications. In addition, it can scale to very large problems because its core operation is matrix-vector multiplications. PDLP is derived by applying the primal-dual hybrid gradient (PDHG) method, popularized by Chambolle and Pock (2011), to a saddle-point formulation of LP. PDLP enhances PDHG for LP by combining several new techniques with older tricks from the literature; the enhancements include diagonal preconditioning, presolving, adaptive step sizes, and adaptive restarting. PDLP improves the state of the art for first-order methods applied to LP. We compare PDLP with SCS, an ADMM-based solver, on a set of 383 LP instances derived from MIPLIB 2017. With a target of 10^{-8} relative accuracy and 1 hour time limit, PDLP achieves a 6.3x reduction in the geometric mean of solve times and a 4.6x reduction in the number of instances unsolved (from 227 to 49). Furthermore, we highlight standard benchmark instances and a large-scale application (PageRank) where our open-source prototype of PDLP, written in Julia, outperforms a commercial LP solver.

1 Introduction

First-order methods (FOMs), which use gradient and not Hessian information, are now applied as standard practice in many areas of optimization [12]. A known weakness of FOMs is the *tailing-off* effect, where FOMs quickly find moderately accurate solutions, but progress towards an optimal solution slows down over time. While moderately accurate solutions are often sufficient for large machine learning applications, other applications traditionally demand higher precision. One such area is Linear Programming (LP), the focus of this work.

LP is a fundamental class of optimization problems in applied mathematics, operations research, and computer science with a huge range of applications, including mixed-integer programming, scheduling, network flow, chip design, budget allocation, and many others [17, 22, 65, 69]. Software

*Part of the work was done at Cornell and Google Research.

†Part of the work was done at Google Research.

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

for solving LP problems, called *LP solvers*, originated in the earliest days of computing, predating the invention of operating systems [55]. The state-of-the-art methods for LP, namely Dantzig's simplex method [22,23] and interior-point (or barrier) methods [51], are quite mature and reliable at delivering highly accurate solutions. These widely successful methods have left little room for FOMs to make inroads. Furthermore, practitioners who use LP solvers are not accustomed to reasoning about the trade-off between accuracy and computing times typically intrinsic to FOMs.

In this paper, we provide evidence that, if properly enhanced, FOMs can obtain high quality solutions to LP problems quickly. Indeed, there's reason to expect this, as authors have developed FOMs for LP with linear rates of convergence [24, 32, 47, 70, 71]. On the other hand, the linear rates depend on potentially loose and hard-to-compute constants; hence, tailing off may still be observed in practice. To our knowledge, ours is the first work to combine both theoretical enhancements with practical heuristics, demonstrating their combined effectiveness with extensive computational experiments on standard benchmark instances. In fact, our experiments will expose a substantial gap between algorithms presented in the literature and what's needed to obtain good performance.

Starting from a baseline primal-dual hybrid gradient (PDHG) method [19] applied to a saddle point formulation of LP, we develop a series of algorithmic improvements. These enhancements include adaptive restarting [7], dynamic primal-dual step size selection [36, 37], presolving techniques [1], and diagonal preconditioning (data equilibration) [33]. Most of these enhancements, while inspired by existing literature, are novel. We name our collection of enhancements *PDLP* (PDHG for LP).

The impact of these improvements is substantial. For example, on 383 LP instances derived from the MIPLIB 2017 collection [34], our implementation of a baseline version of PDHG solved only 50 problems to 10^{-8} relative accuracy given a limit of approximately 100,000 iterations per problem. By contrast, PDLP solves 283 of the 383 problems under the same conditions. We demonstrate that PDLP outperforms FOM baselines and, in a small number of cases, obtains performance competitive with a commercial LP solver.

Although not the focus of this paper, we believe that our results open the door to a new set of possibilities and computational trade-offs when solving LP problems. PDLP has the potential to solve extremely large scale instances where the simplex method and interior-point methods are unable to run because of their reliance on matrix factorization. Since PDLP uses matrix-vector operations at its core, it can effectively run on multi-threaded CPUs, GPUs [68], or distributed clusters [26]. Furthermore, a GPU implementation of PDLP could efficiently solve batches of similar problems, a setup that has already been successfully applied with other optimization algorithms in applications like strong branching [46] and training neural networks that contain optimization layers [5].

Outline. The remainder of this section focuses on related work. Section 2 introduces LP and PDHG. Section 3 describes the set of enhancements that define PDLP. Section 4 presents numerical experiments, and Section 5 concludes and outlines future directions.

1.1 Literature review

PDHG PDHG was first developed by Zhu and Chan [72], with subsequent analysis and extension by a number of authors [3,18,19,21,27,38,60]. PDHG is closely related to the Arrow-Hurwicz method [8]. PDHG is a form of operator-splitting [11, 64] and can be interpreted as a variant of the alternating directions method of multipliers (ADMM) and Douglas-Rachford splitting (DRS) [16, 25, 56], which themselves are both instantiations of the proximal point method [25, 58, 62]. As opposed to ADMM or DRS, PDHG is 'matrix-free' in that the data matrix is only used for matrix-vector multiplications. This allows PDHG to scale to problems even larger than those tackled by these other techniques, and to make better use of parallel and distributed computation.

FOM-based solvers Recent interest in large-scale cone programming has sparked the development several first-order solvers based on competing methods. ProxSDP [66] is a solver for semidefinite programming based on PDHG. Solvers based on Nesterov's accelerated gradients [49] include TFOCS [14], and FOM which is a suite of solvers employing both gradient and proximal algorithms [13]. Solvers based on operator splitting techniques like ADMM include SCS [52–54], OSQP [67], POGS [28], and COSMO [30]. Of these both SCS and POGS offer a matrix-free implementation where the linear system, that arises from the proximal operator used in ADMM, is solved using the conjugate gradient method. However, we shall show experimentally that our method can be significantly faster and more robust than this approach. Finally, [4] considers applying a truncated semismooth Newton method to the system of equations defining a fixed point of the SCS operator.

FOMs for LP Lan, Lu and Monteiro [40] and Renegar [61] develop FOMs for LP as a special case of semidefinite programming, with sublinear convergence rates. The FOM-based solvers above all apply to more general problem classes like cone programming or quadratic programming. In contrast, some of the enhancements that constitute PDLP are specialized, either in theory or practice, for LP (namely restarts [7] and presolving). A number of authors [24, 32, 47, 70, 71] have proposed linearly convergent FOMs for LP; to our knowledge, none have been subject of a comprehensive computational study. ECLIPSE [10] solves huge-scale industrial LP problems by accelerated gradient descent, without presenting comparisons on standard test problems. Lin et al. [42] propose an ADMM-based interior point method. In contrast with PDLP which solves to high accuracy (i.e., 10^{-8} relative error), [42] perform experiments with 10^{-3} and 10^{-5} relative error. SNIPAL [41] is a semismooth Newton method based on the proximal augmented Lagrangian. SNIPAL has fast asymptotic convergence, yet, to get good performance, the authors use ADMM for warm-starts. Given PDLP's favorable comparisons with SCS, it's plausible that PDLP could provide a more effective warm-start. Finally, Pock and Chambolle [59] apply PDHG with diagonal preconditioning to a limited set of test LP problems and Applegate et al. [6] show how to extract infeasibility certificates when applying PDHG to LP.

2 Preliminaries

In this section, we introduce the notation we use throughout the paper, summarize the LP formulations we solve, and introduce the baseline PDHG algorithm.

Notation. Let $\mathbb{R}$ denote the set of real numbers, $\mathbb{R}^+$ the set of nonnegative real numbers, and $\mathbb{R}^-$ the set of nonpositive real numbers. Let $\mathbb{N}$ denote the set of natural numbers (starting from one). Let $\|\cdot\|_p$ denote the ℓ_p norm for a vector, and let $\|\cdot\|_2$ denote the spectral norm for a matrix. For a vector $v \in \mathbb{R}^n$, we use v^+ and v^- for their positive and negative parts, i.e., $v_i^+ = \max\{0, v_i\}$ and $v_i^- = \min\{0, v_i\}$. The symbol $v_{1:m}$ denotes the vector with the first m components of v. The symbols $K_{i,\cdot}$ and $K_{\cdot,j}$ correspond to the ith column and jth row of the matrix K, respectively. The symbol $\mathbf{1}$ denotes the vector of all ones. Given a convex set X, we use $\mathbf{proj}_X$ to denote the map that projects onto X.

Linear Programming. We solve primal-dual LP problems of the form:

$$\begin{array}{lll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & c^\top x & \\ \text{subject to:} & Gx \geq h & \\ & Ax = b & \\ & l \leq x \leq u & \end{array} \qquad \begin{array}{lll} \underset{y\in\mathbb{R}^{m_1+m_2},\lambda\in\mathbb{R}^n}{\text{maximize}} & q^\top y + l^\top\lambda^+ - u^\top\lambda^- & \\ \text{subject to:} & c - K^\top y = \lambda & \\ & y_{1:m_1} \geq 0 & \\ & \lambda \in \Lambda\,, & \end{array} \tag{1}$$

where $G \in \mathbb{R}^{m_1\times n}$, $A \in \mathbb{R}^{m_2\times n}$, $c \in \mathbb{R}^n$, $h \in \mathbb{R}^{m_1}$, $b \in \mathbb{R}^{m_2}$, $l \in (\mathbb{R}\cup\{-\infty\})^n$, $u \in (\mathbb{R}\cup\{\infty\})^n$, $K^\top = \left(G^\top, A^\top\right)$, $q^\top := \left(h^\top, b^\top\right)$, and

$$\Lambda = \Lambda_1 \times \cdots \times \Lambda_n \quad \Lambda_i := \begin{cases} \{0\} & l_i = -\infty,\ u_i = \infty, \\ \mathbb{R}^- & l_i = -\infty,\ u_i \in \mathbb{R} \\ \mathbb{R}^+ & l_i \in \mathbb{R},\ u_i = \infty \\ \mathbb{R} & \text{otherwise} \end{cases}$$

is the set of variables λ such that the dual objective is finite. This pair of primal-dual problems is equivalent to the saddle-point problem:

$$\min_{x\in X}\max_{y\in Y} \mathcal{L}(x,y) := c^\top x - y^\top K x + q^\top y \tag{2}$$

with $X := \{x \in \mathbb{R}^n : l \leq x \leq u\}$, and $Y := \{y \in \mathbb{R}^{m_1+m_2} : y_{1:m_1} \geq 0\}$.

PDHG. When specialized to (2), the PDHG algorithm takes the form:

$$\begin{aligned} x^{k+1} &= \mathbf{proj}_X(x^k - \tau(c - K^\top y^k)) \\ y^{k+1} &= \mathbf{proj}_Y(y^k + \sigma(q - K(2x^{k+1} - x^k))) \end{aligned} \tag{3}$$

where $\tau, \sigma > 0$ are primal and dual step sizes, respectively. PDHG is known to converge to an optimal solution when $\tau\sigma\|K\|_2^2 \leq 1$ [20, 21]. We reparameterize the step sizes by

$$\tau = \eta/\omega \quad \text{and} \quad \sigma = \omega\eta \qquad \text{with } \eta \in (0, \infty) \quad \text{and} \quad \omega \in (0, \infty). \tag{4}$$

We call $\omega \in (0, \infty)$ the *primal weight*, and $\eta \in (0, \infty)$ the *step size*. Under this reparameterization PDHG converges for all $\eta \leq 1/\|K\|_2$. This allows us to control the scaling between the primal and dual iterates with a single parameter ω. We use the term *primal weight* to describe ω because it weights the primal variables in the following norm:

$$\|z\|_\omega := \sqrt{\omega\|x\|_2^2 + \frac{\|y\|_2^2}{\omega}}.$$

This norm plays a role in the theory for PDHG [20] and later algorithmic discussions.

For the *baseline PDHG algorithm* that we use for comparisons, we consider two simple choices for η and ω. For the step size, set $\eta = 0.9/\|K\|_2$ where $\|K\|_2$ is estimated via power iteration, and for the primal weight we set $\omega = 1$; this is similar to the default parameters in the standard PDHG implementation in ODL [2].

3 Practical algorithmic improvements

In this section, we detail these enhancements, and defer further experimental testing of them to Section 4 and ablation studies to Appendix C. While our enhancements are inspired by theory, our focus is on practical performance. The algorithm as a whole has no convergence guarantee, although some individual enhancements do; see Section 3.6 for further discussion.

Algorithm 1 presents pseudo-code for PDLP after preprocessing steps. We modify the step sizes (Section 3.1), add restarts (Section 3.2), and dynamically update the primal weights (Section 3.3). Before running Algorithm 1 we apply presolve (Section 3.4) and diagonal preconditioning (Section 3.5). There are some minor differences between the pseudo-code and the actual code. In particular, we only evaluate the restart or termination criteria (Line 10) every 40 iterations. This reduces the associated overheads with minimal impact on the total number of iterations. We also check the termination criteria before beginning the algorithm or if we detect a numerical error.

Algorithm 1: PDLP (after preconditioning and presolve)

1 **Input:** An initial solution $z^{0,0}$;
2 Initialize outer loop counter $n \leftarrow 0$, total iterations $k \leftarrow 0$, step size $\hat{\eta}^{0,0} \leftarrow 1/\|K\|_\infty$, primal weight $\omega^0 \leftarrow$ `InitializePrimalWeight(`c, q`)`;
3 **repeat**
4 $t \leftarrow 0$;
5 **repeat**
6 $z^{n,t+1}, \eta^{n,t+1}, \hat{\eta}^{n,t+1} \leftarrow$ `AdaptiveStepOfPDHG(`$z^{n,t}, \omega^n, \hat{\eta}^{n,t}, k$`)` ;
7 $\bar{z}^{n,t+1} \leftarrow \frac{1}{\sum_{i=1}^{t+1} \eta^{n,i}} \sum_{i=1}^{t+1} \eta^{n,i} z^{n,i}$;
8 $z_{\text{c}}^{n,t+1} \leftarrow$ `GetRestartCandidate(`$z^{n,t+1}, \bar{z}^{n,t+1}, z^{n,0}$`)` ;
9 $t \leftarrow t + 1, k \leftarrow k + 1$;
10 **until** *restart or termination criteria holds*;
11 **restart the outer loop.** $z^{n+1,0} \leftarrow z_{\text{c}}^{n,t}, n \leftarrow n + 1$;
12 $\omega^n \leftarrow$`PrimalWeightUpdate(`$z^{n,0}, z^{n-1,0}, \omega^{n-1}$`)` ;
13 **until** *termination criteria holds*;
14 **Output:** $z^{n,0}$.

3.1 Step size choice

The convergence analysis [20, Equation (15)] of PDHG (equation (3)) relies on a small constant step size

$$\eta \leq \frac{\|z^{k+1} - z^k\|_\omega^2}{2(y^{k+1} - y^k)^\top K(x^{k+1} - x^k)} \tag{5}$$

Algorithm 2: One step of PDHG using our step size heuristic

Function `AdaptiveStepOfPDHG` $(z^{n,t},\ \omega^n,\ \hat{\eta}^{n,t}, k)$**:**

- $(x, y) \leftarrow z^{n,t}$, $\eta \leftarrow \hat{\eta}^{n,t}$;
- **for** $i = 1, \ldots, \infty$ **do**
  - $x' \leftarrow \mathbf{proj}_X(x - \frac{\eta}{\omega^n}(c - K^\top y))$;
  - $y' \leftarrow \mathbf{proj}_Y(y + \eta\omega^n(q - K(2x' - x)))$;
  - $\bar{\eta} \leftarrow \frac{\|(x'-x, y'-y)\|^2_{\omega^n}}{2(y'-y)^\top K(x'-x)}$;
  - $\eta' \leftarrow \min\left((1 - (k+1)^{-0.3})\bar{\eta}, (1 + (k+1)^{-0.6})\eta\right)$;
  - **if** $\eta \le \bar{\eta}$ **then**
    - **return** (x', y'), η, η'
  - **end**
  - $\eta \leftarrow \eta'$;
- **end**

where $z^k = (x^k, y^k)$. Classically one would ensure (5) by picking $\eta = \frac{1}{\|K\|_2}$. This is overly pessimistic and requires estimation of $\|K\|_2$. Instead our `AdaptiveStepOfPDHG` adjusts η dynamically to ensure that (5) is satisfied. If (5) isn't satisfied, we abort the step; i.e., we reduce η, and try again. If (5) is satisfied we accept the step. This is described in Algorithm 2. Note that in Algorithm 2 $\bar{\eta} \ge \frac{1}{\|K\|_2}$ holds always, and from this one can show the resulting step size $\eta \ge \frac{1-o(1)}{\|K\|_2}$ holds as $k \to \infty$.

Our step size routine compares favorably in practice with the line search by Malitsky and Pock [43] (See Appendix C.1).

3.2 Adaptive restarts

In PDLP, we adaptively restart the PDHG algorithm in each outer iteration. The key to our restarts at the n-th outer iteration is the normalized duality gap at z which for any radius $r \in (0, \infty)$ is defined by

$$\rho_r^n(z) := \frac{1}{r} \underset{(\hat{x},\hat{y}) \in \{\hat{z} \in Z : \|\hat{z} - z\|_{\omega^n} \le r\}}{\text{maximize}} \{\mathcal{L}(x, \hat{y}) - \mathcal{L}(\hat{x}, y)\},$$

introduced by [7]. Unlike the standard duality gap

$$\underset{(\hat{x},\hat{y}) \in Z}{\text{maximize}} \{\mathcal{L}(x, \hat{y}) - \mathcal{L}(\hat{x}, y)\},$$

the normalized duality gap is always a finite quantity. Furthermore, for any value of r and ω^n, the normalized duality gap $\rho_r^n(z)$ is 0 if and only if the solution z is an optimal solution to (2) [7]; thus, it provides a valid metric for measuring progress towards the optimal solution. The normalized duality gap is computable in linear time [7]. For brevity, define $\mu_n(z, z_{\text{ref}})$ as the normalized duality gap at z with radius $\|z - z_{\text{ref}}\|_{\omega^n}$, i.e.,

$$\mu_n(z, z_{\text{ref}}) := \rho^n_{\|z - z_{\text{ref}}\|_{\omega^n}}(z),$$

where z_{ref} is a user-chosen reference point.

Choosing the restart candidate. To choose the restart candidate $z_{\text{c}}^{n,t+1}$ we call

$$\texttt{GetRestartCandidate}(z^{n,t+1}, \bar{z}^{n,t+1}, z^{n,0}) := \begin{cases} z^{n,t+1} & \mu_n(z^{n,t+1}, z^{n,0}) < \mu_n(\bar{z}^{n,t+1}, z^{n,0}) \\ \bar{z}^{n,t+1} & \text{otherwise .} \end{cases}$$

This choice is justified in Remark 5 of [7].

Restart criteria. We define three parameters: $\beta_{\text{sufficient}} \in (0, 1)$, $\beta_{\text{necessary}} \in (0, \beta_{\text{sufficient}})$ and $\beta_{\text{artificial}} \in (0, 1)$. In PDLP we use $\beta_{\text{sufficient}} = 0.9$, $\beta_{\text{necessary}} = 0.1$, and $\beta_{\text{artificial}} = 0.5$. The algorithm restarts if one of three conditions holds:

(i) (**Sufficient decay in normalized duality gap**) $\mu_n(z_{\text{c}}^{n,t+1}, z^{n,0}) \le \beta_{\text{sufficient}} \mu_n(z^{n,0}, z^{n-1,0})$,

(ii) (**Necessary decay + no local progress in normalized duality gap**)

$$\mu_n(z_{\text{c}}^{n,t+1}, z^{n,0}) \le \beta_{\text{necessary}} \mu_n(z^{n,0}, z^{n-1,0}) \quad \text{and} \quad \mu_n(z_{\text{c}}^{n,t+1}, z^{n,0}) > \mu_n(z_c^{n,t}, z^{n,0}) ,$$

(iii) (**Long inner loop**) $t \geq \beta_{\text{artificial}} k$.

The motivation for (i) is presented in [7]; it guarantees the linear convergence of restarted PDHG on LP problems. The second condition in (ii) is inspired by adaptive restart schemes for accelerated gradient descent where restarts are triggered if the function value increases [57]. The first inequality in (ii) provides a safeguard for the second one, preventing the algorithm restarting every inner iteration or never restarting. The motivation for (iii) relates to the primal weights (Section 3.3). In particular, primal weight updates only occur after a restart, and condition (iii) ensures that the primal weight will be updated infinitely often. This prevents a bad choice of primal weight in earlier iterations causing progress to stall for a long time.

3.3 Primal weight updates

The primal weight is initialized using

$$\texttt{InitializePrimalWeight}(c,\, q) := \begin{cases} \frac{\|c\|_2}{\|q\|_2} & \|c\|_2, \|q\|_2 > \epsilon_{\text{zero}} \\ 1 & \text{otherwise} \end{cases}$$

where ϵ_{zero} is a small nonzero tolerance. This primal weight update scheme guarantees scale invariance. In particular, in Appendix A we consider PDHG with $\epsilon_{\text{zero}} = 0$, $\eta = 0.9/\|K\|_2$ and $\omega = \texttt{InitializePrimalWeight}(c,\, q)$. In this simplified setting, we prove that if we multiply the objective, constraints, or the right hand side and variable bounds by a scalar then the iterate behaviour remain identical (up to a scaling factor).

Algorithm 3: Primal weight update

Function `PrimalWeightUpdate`$(z^{n,0}, z^{n-1,0}, \omega^{n-1})$**:**
  $\Delta_x^n = \|x^{n,0} - x^{n-1,0}\|_2, \quad \Delta_y^n = \|y^{n,0} - y^{n-1,0}\|_2$;
  if $\Delta_x^n > \epsilon_{zero}$ *and* $\Delta_y^n > \epsilon_{zero}$ **then**
    return $\exp\left(\theta \log\left(\frac{\Delta_y^n}{\Delta_x^n}\right) + (1-\theta)\log\left(\omega^{n-1}\right)\right)$
  else
    return ω^{n-1} ;
  end

Algorithm 3 aims to choose the primal weight ω^n such that distance to optimality in the primal and dual is the same, i.e., $\|(x^{n,t} - x^\star, \mathbf{0})\|_{\omega^n} \approx \|(\mathbf{0}, y^{n,t} - y^\star)\|_{\omega^n}$. By definition of $\|\cdot\|_\omega$,

$$\|(x^{n,t} - x^\star, \mathbf{0})\|_{\omega^n} = \omega^n \|x^{n,t} - x^\star\|_2, \quad \|(\mathbf{0}, y^{n,t} - y^\star)\|_{\omega^n} = \frac{1}{\omega^n}\|y^{n,t} - y^\star\|_2.$$

Setting these two terms equal yields $\omega^n = \frac{\|y^{n,t} - y^\star\|_2}{\|x^{n,t} - x^\star\|_2}$. Of course, the quantity $\frac{\|y^{n,t} - y^\star\|_2}{\|x^{n,t} - x^\star\|_2}$ is unknown beforehand, but we attempt to estimate it using Δ_y^n/Δ_x^n. However, the quantity Δ_y^n/Δ_x^n can change wildly from one restart to another, causing ω^n to oscillate. To dampen variations in ω^n, we first move to a log-scale where the primal weight is symmetric, i.e., $\log(1/\omega^n) = -\log(\omega^n)$, and perform a exponential smoothing with parameter $\theta \in [0, 1]$. In PDLP, we use $\theta = 0.5$.

There are several important differences between our primal weight heuristic and literature [36, 37]. For example, [36,37] make relatively small changes to the primal weights at each iteration, attempting to balance the primal and dual residual. These changes have to be diminishingly small because, in our experience, PDHG may be unstable if they are too big. In contrast, in our method the primal weight is only updated during restarts, which in practice allows for much larger changes without instability issues. Moreover, our scheme tries to balance the weighted distance traveled in the primal and dual rather than the residuals [36, 37].

3.4 Presolve

Presolving refers to transformation steps that simplify the input problem before starting the optimization solver. These steps span from relatively easy transformations such as detecting inconsistent bounds, removing empty rows and columns of K, and removing variables whose lower and upper bounds are equal, to more complex operations such as detecting duplicate rows in K and tightening bounds. Presolve is a standard component of traditional LP solvers [44]. We are not aware of presolve being combined with PDHG for LP. However, [41, 42] combine presolve with other FOMs.

As an experiment to measure the impact of presolve, we used PaPILO [29], an open-source presolving library. For technical reasons, it was easier to use PaPILO as a standalone executable than as a library. We simulate its effect by simply solving the preprocessed instances. Convergence criteria are evaluated with respect to the presolved instance, not the original problem.

3.5 Diagonal Preconditioning

Preconditioning is a popular heuristic in optimization for improving the convergence of FOMs. To avoid factorizations, we only consider diagonal preconditioners. Our goal is to rescale the constraint matrix $K = (G, A)$ to $\tilde{K} = (\tilde{G}, \tilde{A}) = D_1 K D_2$ with positive diagonal matrices D_1 and D_2, so that the resulting matrix $\tilde{K}$ is "well balanced". Such preconditioning creates a new LP instance that replaces A, G, c, b, h, u, and l in (1) with $\tilde{G}$, $\tilde{A}$, $\hat{x} = D_2^{-1}x$, $\tilde{c} = D_2 c$, $(\tilde{b}, \tilde{h}) = D_1(b, h)$, $\tilde{u} = D_2^{-1}u$ and $\tilde{l} = D_2^{-1}l$. Common choices for D_1 and D_2 include:

- **No scaling:** Solve the original LP instance (1) without additional scaling, namely $D_1 = D_2 = I$.
- **Pock-Chambolle [59]:** Pock and Chambolle proposed a family of diagonal preconditioners[3] for PDHG parameterized by α, where the diagonal matrices are defined by $(D_1)_{jj} = \sqrt{\|K_{j,\cdot}\|_{2-\alpha}}$ for $j = 1, ..., m_1 + m_2$ and $(D_2)_{ii} = \sqrt{\|K_{\cdot,i}\|_{\alpha}}$ for $i = 1, ..., n$. We use $\alpha = 1$ in PDLP (we also tested $\alpha = 0$ and $\alpha = 2$). This is the baseline diagonal preconditioner in the PDHG literature.
- **Ruiz [63]:** Ruiz scaling is a popular algorithm in numerical linear algebra to equilibrate matrices. In an iteration of Ruiz scaling, the diagonal matrices are defined as $(D_1)_{jj} = \sqrt{\|K_{j,\cdot}\|_{\infty}}$ for $j = 1, ..., m_1 + m_2$ and $(D_2)_{ii} = \sqrt{\|K_{\cdot,i}\|_{\infty}}$ for $i = 1, ..., n$. Ruiz [63] shows that if this rescaling is applied iteratively, the infinity norm of each row and each column converge to 1.

For the default PDLP settings, we apply a combination of Ruiz rescaling [63] and the preconditioning technique proposed by Pock and Chambolle [59]. In particular, we apply 10 iterations of Ruiz scaling and then apply the Pock-Chambolle scaling. To illustrate the effectiveness of our proposed scaling technique, we compare it against these three common techniques in Appendix C.5.

3.6 Theoretical guarantees for the above enhancements

While PDLP's enhancements are motivated by theory, some of them may not preserve theoretical guarantees as discussed below:

- We do not have a proof of convergence for the adaptive step size rule (Section 3.1).
- One can show our restart criteria (Section 3.2) preserve convergence guarantees by modifying the proof of [7] to a more general setting.
- Primal weight updates (Section 3.3) do not readily preserve convergence guarantees, but we conjecture that a proof of convergence is possible if they are updated infrequently.
- Presolve (Section 3.4) and diagonal preconditioning (Section 3.5) preserve theoretical guarantees because they can be viewed as applying PDHG to an LP instance with different data.

4 Numerical experiments

Our numerical experiments study the effectiveness of PDLP primarily with respect to traditional LP applications and benchmark sets. Section 4.1 describes the setup for the experiments. Section 4.2 demonstrates PDLP's improvements over baseline PDHG. Section 4.3 compares PDLP with other FOMs. Section 4.4 highlights benchmark instances where PDLP outperforms a commercial LP solver. Finally, Section 4.5 illustrates the ability of PDLP to scale to a large application where barrier and simplex-based solvers run out of memory. The supplemental materials contain extensive ablation studies and additional instructions for reproducing the experiments.

[3] Diagonal preconditioning is equivalent to changing to a weighted ℓ_2 norm in the proximal step of PDHG (weight defined by D_2 and D_1 for the primal and dual respectively). Pock and Chambolle use this weighted norm perspective.

4.1 Experimental setup

Optimality termination criteria. PDLP terminates with an approximately optimal solution when the primal-dual iterates $x \in X$, $y \in Y$, $\lambda \in \Lambda$, satisfy:

$$\left|q^\top y + l^\top \lambda^+ - u^\top \lambda^- - c^\top x\right| \leq \epsilon(1 + \left|q^\top y + l^\top \lambda^+ - u^\top \lambda^-\right| + \left|c^\top x\right|) \tag{6a}$$

$$\left\| \begin{pmatrix} Ax - b \\ (h - Gx)^+ \end{pmatrix} \right\|_2 \leq \epsilon(1 + \|q\|_2) \tag{6b}$$

$$\|c - K^\top y - \lambda\|_2 \leq \epsilon(1 + \|c\|_2) \tag{6c}$$

where $\epsilon \in (0, \infty)$ is the termination tolerance. Note that if (6) is satisfied with $\epsilon = 0$, then by LP duality we have found an optimal solution [35]. Indeed, (6a) is the duality gap, (6b) is primal feasibility, and (6c) is dual feasibility. We use these criteria to be consistent with those of SCS [54]. The PDHG algorithm does not explicitly include a reduced costs variable λ. Therefore, to evaluate the optimality termination criteria we compute $\lambda = \mathbf{proj}_\Lambda(c - K^\top y)$. All instances considered have an optimal primal-dual solution. We use $\epsilon = 10^{-8}$ as a benchmark for high-quality solutions and $\epsilon = 10^{-4}$ for moderately accurate solutions.

Benchmark datasets. We use three datasets to compare algorithmic performance. One is the `LP benchmark` dataset of 56 problems, formed by merging the instances from "Benchmark of Simplex LP Solvers", "Benchmark of Barrier LP solvers", and "Large Network-LP Benchmark" from [45]. We also created a larger benchmark of 383 instances curated from LP relaxations of mixed-integer programming problems from the MIPLIB2017 collection [34] (see Appendix B) that we label `MIP Relaxations`. `MIP Relaxations` was used extensively during algorithmic development, e.g., for hyperparameter choices; we held out `LP benchmark` as a test set. Finally, we also performed some experiments on the `Netlib` LP benchmark [31], an historically important benchmark that is no longer state of the art for large-scale LP.

Software. PDLP is implemented in an open-source Julia [15] module available at `https://github.com/google-research/FirstOrderLp.jl`. The module also contains a baseline implementation of the extragradient method with many of the same enhancements as PDLP (labeled 'Enh. Extragradient'). We compare with two external packages: SCS [54] version 2.1.3, an open-source generic cone solver based on ADMM, and Gurobi version 9.0.1, a state-of-the-art commercial LP solver. SCS supports two modes for solving the linear system that arises at each iteration, a direct method based on a cached LDL factorization (which is the default 'SCS') and an indirect method based on the conjugate gradient method (which we label 'SCS (matrix-free)'). All solvers are run single-threaded. SCS and Gurobi are provided the same presolved instances as PDLP.

Computing environment. We used two computing environments for our experiments: 1) `e2-highmem-2` virtual machines (VMs) on Google Cloud Platform (GCP). Each VM provides two virtual CPUs and 16GB RAM. 2) A dedicated workstation with an Intel Xeon E5-2669 v3 processor and 128 GB RAM. This workstation has a license for Gurobi that permits at most one concurrent solve. Total compute time on GCP for all preliminary and final experiments was approximately $72,000$ virtual CPU hours.

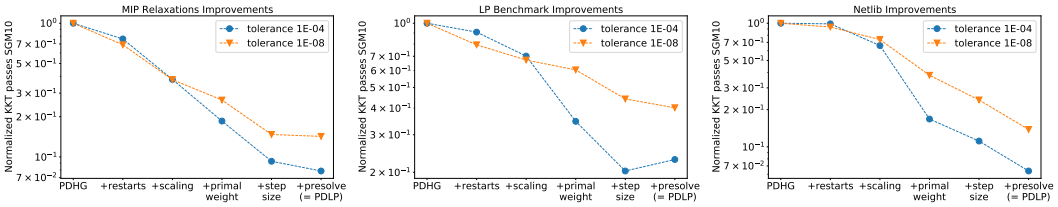
Initialization. All first-order methods use all-zero vectors as the initial starting points.

Metrics. We use the term *KKT passes* to refer to the number of matrix multiplications by both K and $K^\top$. Given that the most expensive operation in our algorithm is matrix-vector multiplication, this metric is less noisy than runtime for comparing performance between matrix-free solvers. SGM10 stands for shifted geometric mean with shift 10, which is computed by adding 10 to all data points, taking the geometric mean, and then subtracting 10. Unsolved instances are assigned values corresponding to the limits specified in the next paragraph.

Time and KKT pass limits. For Section 4.2 we impose a limit on the KKT passes of $100,000$. For Section 4.3 we impose a time limit of 1 hour.

4.2 Impact of PDLP's improvements

The y-axes of Figure 1 display the SGM10 of the KKT passes normalized by the value for baseline PDHG. We can see, with the exception of presolve for `LP benchmark` at tolerance 10^{-4}, each of our modifications described in Section 3 improves the performance of PDHG.



(a) `MIP Relaxations` (b) `LP benchmark` (c) `Netlib`

Figure 1: Summary of relative impact of PDLP's improvements

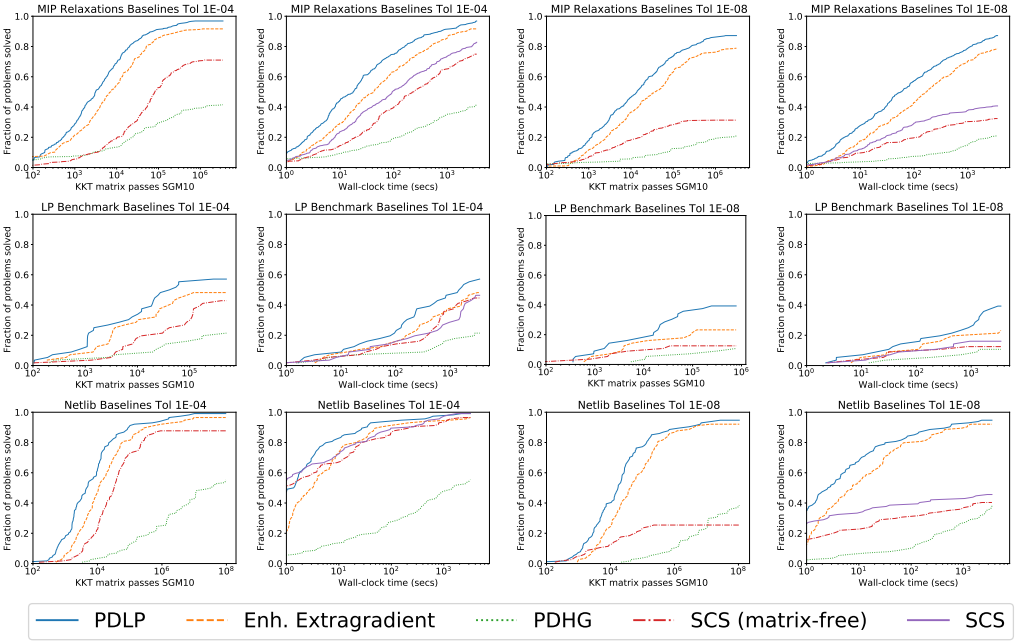


Figure 2: Number of problems solved for `MIP Relaxations` (top), `LP benchmark` (middle), and `Netlib` (bottom) datasets.

4.3 Comparison with other first-order baselines

We compared PDLP with several other first-order baselines: SCS [54], in both direct (default) mode and matrix-free mode, and our enhanced implementation of the extragradient method [39, 48]. For SCS in matrix-free mode, we include the KKT passes from the conjugate gradient solves; for SCS in direct mode there is no reasonable measure of KKT passes for the factorization and direct solve, so we only measure running time. The comparisons are summarized in Figure 2.

4.4 PDLP versus simplex and barrier

In this section, we test the performance of PDLP against the three methods available in Gurobi: barrier, primal simplex, and dual simplex. By default when provided multiple threads, Gurobi runs these three methods concurrently and terminates when the first method completes. We used default termination for Gurobi and set $\epsilon = 10^{-8}$ for PDLP. We ran experiments with instances from the `MIP Relaxations` and `LP benchmark`. Although, for most instances, Gurobi outperforms PDLP, we found problems for which PDLP exhibits moderate to significant gains. Table 1 gives examples of instances where our prototype implementation is within a factor of two of the best of the three Gurobi methods. While further improvements are needed for PDLP to truly compete with the portfolio of methods that Gurobi offers, we interpret these results as evidence that PDLP itself could be of value in this portfolio.

Table 1: Instances from `MIP Relaxations` (top) and `LP benchmark` (bottom) where PDLP is within a factor of 2 of the best of all Gurobi methods. Time to solve in seconds.

Instance	PDLP	Gurobi Barrier	Gurobi Primal Simp.	Gurobi Dual Simp.
ex9	1.6	102.6	181.3	47.6
genus-sym-g62-2	2.1	10.7	6.7	33.2
highschool1-aigio	72.6	243.8	>3600	>3600
neos-578379	1.4	0.7	1.7	1.8
rwth-timetable	1870.3	>3600	>3600	>3600
ex10	4.9	63.1	16.8	7.9
nug08-3rd	2.2	3.2	2219.2	24.1
savsched1	35.9	25.9	56.0	261.3

Table 2: Solve time for PageRank instances. Gurobi barrier has crossover disabled, 1 thread. PDLP and SCS solve to 10^{-8} relative accuracy. SCS is matrix-free. Baseline PDHG is unable to solve any instances. Presolve not applied. OOM = Out of Memory. The number of nonzero coefficients per instance is $8 \times (\#\text{ nodes}) - 18$.

# nodes	PDLP	SCS	Gurobi Barrier	Gurobi Primal Simp.	Gurobi Dual Simp.
10^4	7.4 sec.	1.3 sec.	36 sec.	37 sec.	114 sec.
10^5	35 sec.	38 sec.	7.8 hr.	9.3 hr.	>24 hr.
10^6	11 min.	25 min.	OOM	>24 hr.	-
10^7	5.4 hr.	3.8 hr.	-	-	-

4.5 Large-scale application: PageRank

Nesterov [50, equation (7.3)] gives an LP formulation of the standard "PageRank" problem. Although the LP formulation is not the best approach to computing PageRank, it is a source of very large instances. For a random scalable collection of PageRank instances, we used Barabási-Albert [9] preferential attachment graphs with approximately three edges per node; see Appendix D for details. The results are summarized in Table 2.

5 Conclusions and future work

We find our experimental results encouraging for the application of FOMs like PDHG to LP. At a minimum, they provide evidence against the claim that FOMs are useful only when moderately accurate solutions are desired. The practical success of our heuristics that lack theoretical guarantees provides fresh motivation for theoreticians to study these methods. It is important, as well, to understand what drives the difficulty of some instances and how they could be transformed to solve more quickly. We hope the community will use the benchmarks and baselines released with this work as a starting point for further investigating new FOMs for LP. With additional algorithmic and implementation refinements, we believe that PDLP or similar approaches could become part of the standard toolkit for linear programming.

Acknowledgments and Disclosure of Funding

We thank Yura Malitsky for advice on parameter choices for the linesearch rule of [43].

The authors have no third-party funding or competing interests to declare.

References

[1] T. Achterberg, R. E. Bixby, Z. Gu, E. Rothberg, and D. Weninger. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing*, 32(2):473–506, 2020.

[2] J. Adler, H. Kohr, and O. Öktem. Operator discretization library (ODL), Jan. 2017.

[3] A. Alacaoglu, O. Fercoq, and V. Cevher. On the convergence of stochastic primal-dual hybrid gradient. *arXiv preprint arXiv:1911.00799*, 2019.

[4] A. Ali, E. Wong, and J. Z. Kolter. A semismooth Newton method for fast, generic convex programming. In *International Conference on Machine Learning*, pages 70–79. PMLR, 2017.

[5] B. Amos and J. Z. Kolter. OptNet: Differentiable optimization as a layer in neural networks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 136–145. PMLR, 06–11 Aug 2017.

[6] D. Applegate, M. Díaz, H. Lu, and M. Lubin. Infeasibility detection with primal-dual hybrid gradient for large-scale linear programming. *arXiv preprint arXiv:2102.04592*, 2021.

[7] D. Applegate, O. Hinder, H. Lu, and M. Lubin. Faster First-Order Primal-Dual Methods for Linear Programming using Restarts and Sharpness. *arXiv preprint arXiv:2105.12715*, 2021.

[8] K. J. Arrow, L. Hurwicz, and H. Uzawa. *Studies in linear and non-linear programming*. Stanford University Press, 1958.

[9] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[10] K. Basu, A. Ghoting, R. Mazumder, and Y. Pan. ECLIPSE: An extreme-scale linear program solver for web-applications. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 704–714, Virtual, 13–18 Jul 2020. PMLR.

[11] H. H. Bauschke and P. L. Combettes. *Convex analysis and monotone operator theory in Hilbert spaces*, volume 408. Springer, 2 edition, 2017.

[12] A. Beck. *First-Order Methods in Optimization*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2017.

[13] A. Beck and N. Guttmann-Beck. FOM–a MATLAB toolbox of first-order methods for solving convex optimization problems. *Optimization Methods and Software*, 34(1):172–193, 2019.

[14] S. R. Becker, E. J. Candès, and M. C. Grant. Templates for convex cone problems with applications to sparse signal recovery. *Mathematical programming computation*, 3(3):165, 2011.

[15] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

[16] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[17] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[18] A. Chambolle, M. J. Ehrhardt, P. Richtárik, and C.-B. Schonlieb. Stochastic primal-dual hybrid gradient algorithm with arbitrary sampling and imaging applications. *SIAM Journal on Optimization*, 28(4):2783–2808, 2018.

[19] A. Chambolle and T. Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of mathematical imaging and vision*, 40(1):120–145, 2011.

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

[21] L. Condat. A primal–dual splitting method for convex optimization involving Lipschitzian, proximable and linear composite terms. *Journal of Optimization Theory and Applications*, 158(2):460–479, 2013.

[22] G. Dantzig. *Linear programming and extensions*. Princeton university press, 2016.

[23] G. B. Dantzig. Origins of the simplex method. In *A history of scientific computing*, pages 141–151. Association for Computing Machinery, New York, NY, USA, 1990.

[24] J. Eckstein and D. P. Bertsekas. An alternating direction method for linear programming. Technical Report LIDS-P-1967, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1990.

[25] J. Eckstein and D. P. Bertsekas. On the Douglas–Rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55(1-3):293–318, 1992.

[26] J. Eckstein and G. Matyasfalvi. Efficient distributed-memory parallel matrix-vector multiplication with wide or tall unstructured sparse matrices. *arXiv preprint arXiv:1812.00904*, 2018.

[27] E. Esser, X. Zhang, and T. F. Chan. A general framework for a class of first order primal-dual algorithms for convex optimization in imaging science. *SIAM Journal on Imaging Sciences*, 3(4):1015–1046, 2010.

[28] C. Fougner and S. Boyd. Parameter selection and preconditioning for a graph form solver. In *Emerging Applications of Control and Systems Theory*, pages 41–61. Springer, 2018.

[29] G. Gamrath, D. Anderson, K. Bestuzheva, W.-K. Chen, L. Eifler, M. Gasse, P. Gemander, A. Gleixner, L. Gottwald, K. Halbig, et al. The SCIP optimization suite 7.0. ZIB-Report 20-10, Zuse Institut Berlin, 2020.

[30] M. Garstka, M. Cannon, and P. Goulart. COSMO: A conic operator splitting method for large convex problems. In *European Control Conference*, 2019.

[31] D. M. Gay. Electronic mail distribution of linear programming test problems. *Mathematical Programming Society COAL Newsletter*, 13:10–12, 1985.

[32] A. Gilpin, J. Pena, and T. Sandholm. First-order algorithm with $\mathcal{O}(\ln(1/\epsilon))$ convergence for ϵ-equilibrium in two-person zero-sum games. *Mathematical programming*, 133(1):279–298, 2012.

[33] P. Giselsson and S. Boyd. Linear convergence and metric selection for Douglas-Rachford splitting and ADMM. *IEEE Transactions on Automatic Control*, 62(2):532–544, 2016.

[34] A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. M. Christophel, K. Jarck, T. Koch, J. Linderoth, M. Lübbecke, H. D. Mittelmann, D. Ozyurt, T. K. Ralphs, D. Salvagnin, and Y. Shinano. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*, 2021.

[35] A. J. Goldman and A. W. Tucker. Theory of linear programming. *Linear inequalities and related systems*, 38:53–97, 1956.

[36] T. Goldstein, M. Li, and X. Yuan. Adaptive primal-dual splitting methods for statistical learning and image processing. In *Advances in Neural Information Processing Systems*, pages 2089–2097, 2015.

[37] T. Goldstein, M. Li, X. Yuan, E. Esser, and R. Baraniuk. Adaptive primal-dual hybrid gradient methods for saddle-point problems. *arXiv preprint arXiv:1305.0546*, 2013.

[38] B. He and X. Yuan. Convergence analysis of primal-dual algorithms for a saddle-point problem: from contraction perspective. *SIAM Journal on Imaging Sciences*, 5(1):119–149, 2012.

[39] G. M. Korpelevich. The extragradient method for finding saddle points and other problems. *Matecon*, 12:747–756, 1976.

[40] G. Lan, Z. Lu, and R. D. C. Monteiro. Primal-dual first-order methods with $\mathcal{O}(1/\epsilon)$ iteration-complexity for cone programming. *Mathematical Programming*, 126(1):1–29, Jan 2011.

[41] X. Li, D. Sun, and K.-C. Toh. An asymptotically superlinearly convergent semismooth newton augmented lagrangian method for linear programming. *SIAM Journal on Optimization*, 30(3):2410–2440, 2020.

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

[43] Y. Malitsky and T. Pock. A first-order primal-dual algorithm with linesearch. *SIAM Journal on Optimization*, 28(1):411–432, 2018.

[44] I. Maros. *Computational Techniques of the Simplex Method*. International Series in Operations Research & Management Science. Springer US, 2002.

[45] H. Mittelmann. Decision tree for optimization software. `http://plato.asu.edu/guide.html`, 2021.

[46] V. Nair, S. Bartunov, F. Gimeno, I. von Glehn, P. Lichocki, I. Lobov, B. O'Donoghue, N. Sonnerat, C. Tjandraatmadja, P. Wang, R. Addanki, T. Hapuarachchi, T. Keck, J. Keeling, P. Kohli, I. Ktena, Y. Li, O. Vinyals, and Y. Zwols. Solving Mixed Integer Programs Using Neural Networks. *arXiv preprint arXiv:2012.13349*, Dec. 2020.

[47] I. Necoara, Y. Nesterov, and F. Glineur. Linear convergence of first order methods for non-strongly convex optimization. *Mathematical Programming*, 175(1):69–107, May 2019.

[48] A. Nemirovski. Prox-method with rate of convergence $O(1/t)$ for variational inequalities with Lipschitz continuous monotone operators and smooth convex-concave saddle point problems. *SIAM Journal on Optimization*, 15(1):229–251, 2004.

[49] Y. Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

[50] Y. Nesterov. Subgradient methods for huge-scale optimization problems. *Mathematical Programming*, 146:275–297, 2014.

[51] Y. Nesterov and A. Nemirovskii. *Interior-point polynomial algorithms in convex programming*, volume 13. SIAM, 1994.

[52] B. O'Donoghue. Operator splitting for a homogeneous embedding of the monotone linear complementarity problem. *arXiv preprint arXiv:2004.02177*, 2020.

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

[54] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. SCS: Splitting conic solver, version 2.1.0. `https://github.com/cvxgrp/scs`, Nov. 2017.

[55] W. Orchard-Hays. History of mathematical programming systems. *IEEE Annals of the History of Computing*, 6(3):296–312, 1984.

[56] D. O'Connor and L. Vandenberghe. On the equivalence of the primal-dual hybrid gradient method and Douglas–Rachford splitting. *Mathematical Programming*, 179(1):85–108, 2020.

[57] B. O'Donoghue and E. Candes. Adaptive restart for accelerated gradient schemes. *Foundations of computational mathematics*, 15(3):715–732, 2015.

[58] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

[59] T. Pock and A. Chambolle. Diagonal preconditioning for first order primal-dual algorithms in convex optimization. In *2011 International Conference on Computer Vision*, pages 1762–1769. IEEE, 2011.

[60] T. Pock, D. Cremers, H. Bischof, and A. Chambolle. An algorithm for minimizing the mumford-shah functional. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1133–1140. IEEE, 2009.

[61] J. Renegar. Accelerated first-order methods for hyperbolic programming. *Mathematical Programming*, 173(1):1–35, Jan 2019.

[62] R. T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM journal on control and optimization*, 14(5):877–898, 1976.

[63] D. Ruiz. A scaling algorithm to equilibrate both rows and columns norms in matrices. Technical report, CM-P00040415, 2001.

[64] E. K. Ryu and S. Boyd. Primer on monotone operator methods. *Appl. Comput. Math*, 15(1):3–43, 2016.

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

[66] M. Souto, J. D. Garcia, and Á. Veiga. Exploiting low-rank structure in semidefinite programming by approximate operator splitting. *Optimization*, pages 1–28, 2020.

[67] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

[68] Y. M. Tsai, T. Cojean, and H. Anzt. Sparse linear algebra on AMD and NVIDIA GPUs – the race is on. In P. Sadayappan, B. L. Chamberlain, G. Juckeland, and H. Ltaief, editors, *High Performance Computing*, pages 309–327, Cham, 2020. Springer International Publishing.

[69] R. J. Vanderbei et al. *Linear programming*, volume 3. Springer, 2015.

[70] S. Wang and N. Shroff. A new alternating direction method for linear programming. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[71] T. Yang and Q. Lin. RSG: Beating subgradient method without smoothness and strong convexity. *The Journal of Machine Learning Research*, 19(1):236–268, 2018.

[72] M. Zhu and T. Chan. An efficient primal-dual hybrid gradient algorithm for total variation image restoration. *UCLA CAM Report*, 34:8–34, 2008.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Our comparisons with baselines show that PDLP is not always the best, hence demonstrating its limitations. We also note which of our heuristics are lacking theoretical guarantees.
   (c) Did you discuss any potential negative societal impacts of your work? [No] As a purely algorithmic paper, we do not believe such a discussion is relevant. Linear programming is a mature area whose societal impact is well understood.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] The only theoretical result appear in the appendix (Proposition 1).
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code, data, and instructions needed to reproduce the main experimental results are publicly released in an open source repository at `https://github.com/google-research/FirstOrderLp.jl`. Additional instructions are included in the supplementary materials.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We discussed that we used `MIP Relaxations` to build our algorithm

and `LP benchmark` and `Netlib` as held out evaluation sets. We run a large ablation study to justify algorithmic decisions and most hyperparameter settings.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] We used large datasets that would have been computational intensive to run multiple times. When possible, we use the "KKT pass" metric that's reproducible and not subject to measurement noise. Only the PageRank instances use a random seed, and these are too large to run multiple times because we have a single concurrent license for Gurobi.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 4.1.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] We use `LP Benchmark`, `MIP Relaxations`, and `Netlib` datasets. We cite the sources for the datasets.

(b) Did you mention the license of the assets? [No] Although these datasets are widely used in the community, we were unable to find explicit licenses.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We include our code and data processing scripts in the supplemental material.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]