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

David Applegate* Mateo Díaz† Oliver Hinder‡ Haihao Lu§
Miles Lubin* Brendan O'Donoghue¶ Warren Schudy*

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 standard practice in many areas of optimization [10]. 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 [15,19,59,62]. Software for solving LP problems, called *LP solvers*, originated in the earliest days of computing, predating the invention of operating systems [49]. The state-of-the-art methods for LP, namely Dantzig's simplex method [19,20] and interior-point (or barrier) methods [45], 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 (i.e., without a tailing-off effect) [21, 28, 41, 63, 64]. However, to our knowledge,

*Google Research ({dapplegate, mlubin, wschudy}@google.com);
†Center for Applied Mathematics, Cornell University (md825@cornell.edu);
‡Google Research and University of Pittsburgh (ohinder@pitt.edu);
§Google Research and University of Chicago (haihao.lu@chicagobooth.edu);
¶Deepmind (bodonoghue@deepmind.com);

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 [16] applied to a saddle point formulation of LP, we develop a series of algorithmic improvements. These enhancements include adaptive restarting [5], dynamic primal-dual step size selection [32,33], presolving techniques [1], and diagonal preconditioning (data equilibration) [29]. 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 [30], 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 [61], or distributed clusters [23]. 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 [40] and training neural networks that contain optimization layers [3].

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 [65], with subsequent analysis and extension by a number of authors [16,18,24,34,54]. PDHG is closely related to the Arrow-Hurwicz method [6]. PDHG is a form of operator-splitting [9,58] and can be interpreted as a variant of the alternating directions method of multipliers (ADMM) and Douglas-Rachford splitting (DRS) [14,22,50], which themselves are both instantiations of the proximal point method [22,52,56]. 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. Solvers based on Nesterov's accelerated gradients [44] include TFOCS [12], and FOM which is a suite of solvers employing both gradient and proximal algorithms [11]. Solvers based on operator splitting techniques like ADMM include SCS [46–48], OSQP [60], POGS [25], and COSMO [27]. 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.

FOMs for LP Lan, Lu and Monteiro [36] and Renegar [55] 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 [5] and presolving). A number of authors [21,28,41,63,64] have proposed linearly convergent FOMs for LP; to our knowledge, none have been subject of a comprehensive computational study. [53] apply PDHG with diagonal preconditioning to a limited set of test LP problems. [4] show how to extract infeasibility certificates when applying PDHG to LP. ECLIPSE [8] solves huge-scale industrial LP problems by accelerated gradient descent, without presenting comparisons on standard test problems.

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 symbol $K_{i,\cdot}$ and $K_{\cdot,j}$ corresponds to the ith column and jth row of the matrix K. 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:

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

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

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

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

where $\tau, \sigma > 0$ are primal and dual stepsizes, respectively. PDHG is known to converge to an optimal solution when $\tau\sigma\|K\|_2^2 < 1$ [16]. We reparameterize the stepsizes 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 *stepsize*. Under this reparameterization PDHG converges for all $\eta < 1/\|K\|_2$. 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 [17] 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 has no convergence guarantee.

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          $z_c^{n,t+1} \leftarrow$ 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. $z^{n+1,0} \leftarrow z_c^{n,t}$, $n \leftarrow n+1$;
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 [17, Equation (15)] of PDHG (equation (3)) relies on a small constant step-size

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

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

Algorithm 2: 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
        $x' \leftarrow \mathbf{proj}_X(x - \frac{\eta}{\omega^n}(c - K^\top y))$ ;
        $y' \leftarrow \mathbf{proj}_Y(y + \eta\omega^n(q - K(2x' - x)))$ ;
        $\bar{\eta} \leftarrow \frac{\|(x'-x, y'-y)\|_{\omega^n}^2}{2(y'-y)^\top K(x'-x)}$ ;
        $\eta' \leftarrow \min\left((1 - (k+1)^{-0.3})\bar{\eta}, (1 + (k+1)^{-0.6})\eta\right)$ ;
        if $\eta \leq \bar{\eta}$ then
            return $(x', y')$, $\eta$, $\eta'$
        end
        $\eta \leftarrow \eta'$ ;
    end
```

the step. This is described in Algorithm 2. Note that in Algorithm 2 $\bar{\eta} \geq \frac{1}{\|K\|_2}$ holds always, and from this one can show the resulting step size $\eta \geq \frac{1-o(1)}{\|K\|_2}$ holds as $k \to \infty$.

Our step size routine compares favorably in practice with the line search by Malitsky and Pock [37] (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 with radius r

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

introduced by [5]. 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, the normalized duality gap $\rho_r^n(z)$ is 0 for any $r > 0$ and ω^n if and only if the solution z is an optimal solution to (2) [5], thus it provides a valid metric for measuring progress towards the optimal solution. The normalized duality gap is computable in linear time [5]. 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 [5].

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**) $\mu_n(z_{\text{c}}^{n,t+1}, z^{n,0}) \leq \beta_{\text{sufficient}} \mu_n(z^{n,0}, z^{n-1,0})$,

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

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

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

The motivation for (i) is presented in [5]; 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 [51]. 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 $\exp\left(\theta \log\left(\frac{\Delta_y^n}{\Delta_x^n}\right) + (1-\theta)\log(\omega^{n-1})\right)$
    else
        return $\omega^{n-1}$ ;
    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 smooth ω^n out, 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 $\theta = 0.5$. Compared with the procedure of [33] for choosing the primal weight, which limits modifications with an exponential decay, our method is more aggressive.

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 [38]. We are not aware of presolve being combined with PDHG or other FOMs for LP.

As an experiment to measure the impact of presolve, we used PaPILO [26], 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 [53]:** Pock and Chambolle proposed a family of diagonal preconditioners[1] 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 [57]:** 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 [57] 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 [57] and the preconditioning technique proposed by Pock and Chambolle [53]. 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.

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 appendix contains extensive ablation studies.

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:

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

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

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

[1]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.

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 [31]. 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 [47]. 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 two 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 [39]. We also created a larger benchmark of 383 instances curated from LP relaxations of mixed-integer programming problems from the MIPLIB2017 collection [30] (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.

Software. PDLP is implemented in an open-source Julia [13] module. 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 [47] 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 $K^\top$. As 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 and then taking the geometric mean. Unsolved instances are assigned values corresponding to the limits specified in the next paragraph.

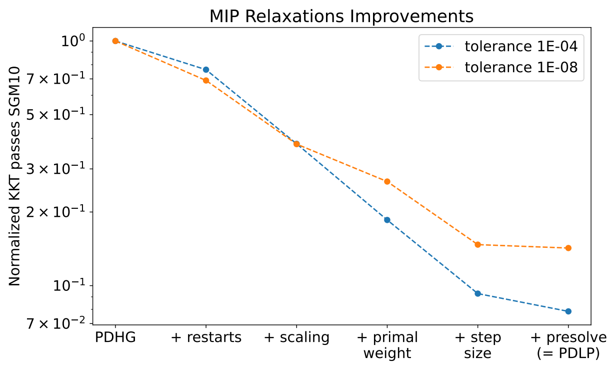
Time and KKT pass limits. For Section 4.2 we impose a limit on the KKT passes of $100,000$. For Section 4.3 we impose a time limit of 1 hour.

4.2 Impact of PDLP's improvements

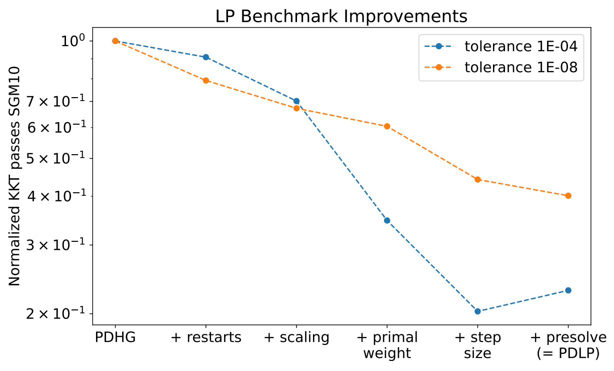
Section 3 details our algorithmic improvements. The y-axis of Figure 1a and 1b displays the SGM10 of the KKT passes normalized by the value for baseline PDHG. We can see, with the exception of presolve for `LP benchmark` at tolerance 10^{-4}, each of our modifications improves the performance of PDHG.

4.3 Comparison with other first-order baselines

We compared PDLP with several other first-order baselines: SCS [47], in both direct (default) mode and matrix-free mode, and our enhanced implementation of the extragradient method [35,42]. 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.



(a) `MIP Relaxations` improvements



(b) `LP benchmark` improvements

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

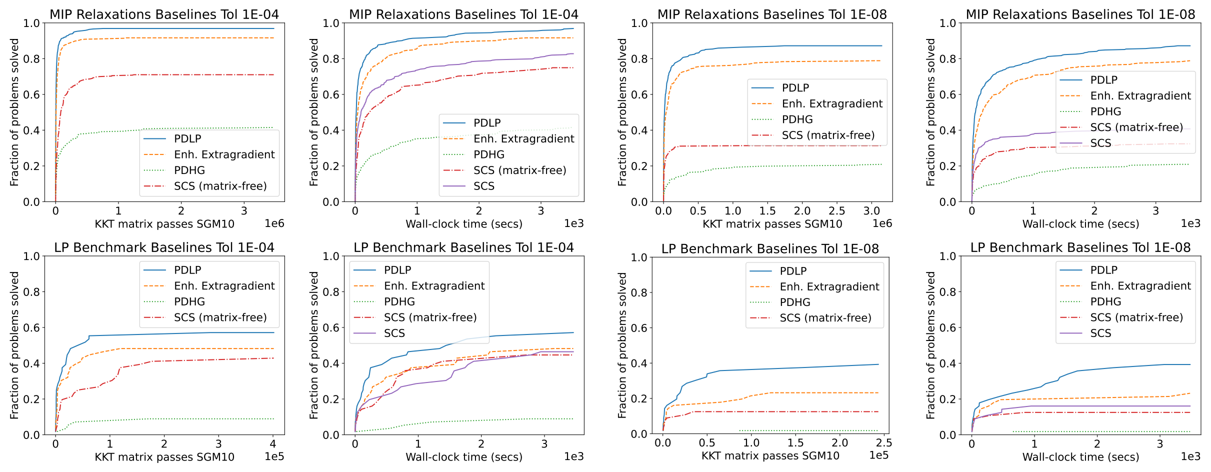


Figure 2: Number of problems solved for `MIP Relaxations` (top) and `LP benchmark` (bottom) datasets.

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.	-	-	-

4.4 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.

4.5 Large-scale application: PageRank

Nesterov [43, 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 [7] preferential attachment graphs with approximately three edges per node; see Appendix D for details. The results are summarized in Table 2.

5 Future work

In the introduction we outline new implementation possibilities that we intend to pursue. In addition, we would like to develop a stronger theoretical foundation for PDLP’s primal weight updates and step size choices. Finally, 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.

6 Source code

The Julia module underlying our experiments is open source and available at `https://github.com/google-research/FirstOrderLp.jl`.

References

[1] Tobias Achterberg, Robert E Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing*, 32(2):473–506, 2020.

[2] Jonas Adler, Holger Kohr, and Ozan Öktem. Operator discretization library (ODL), January 2017.

[3] Brandon Amos and J. Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In Doina Precup and Yee Whye 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.

[4] David Applegate, Mateo Díaz, Haihao Lu, and Miles Lubin. Infeasibility detection with primal-dual hybrid gradient for large-scale linear programming. *arXiv preprint arXiv:2102.04592*, 2021.

[5] David Applegate, Oliver Hinder, Haihao Lu, and Miles Lubin. Faster First-Order Primal-Dual Methods for Linear Programming using Restarts and Sharpness. *arXiv preprint arXiv:2105.12715*, 2021.

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

[7] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[8] Kinjal Basu, Amol Ghoting, Rahul Mazumder, and Yao Pan. ECLIPSE: An extreme-scale linear program solver for web-applications. In Hal Daumé III and Aarti 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.

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

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

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

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

[13] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

[14] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[15] Stephen Boyd and Lieven Vandenberghe. *Convex optimization.* Cambridge university press, 2004.

[16] Antonin Chambolle and Thomas 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.

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

[18] Laurent 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.

[19] George Dantzig. *Linear programming and extensions*. Princeton university press, 2016.

[20] George 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.

[21] Jonathan Eckstein and Dimitri 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.

[22] Jonathan Eckstein and Dimitri 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.

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

[24] Ernie Esser, Xiaoqun Zhang, and Tony 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.

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

[26] Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, et al. The SCIP optimization suite 7.0. ZIB-Report 20-10, Zuse Institut Berlin, 2020.

[27] Michael Garstka, Mark Cannon, and Paul Goulart. COSMO: A conic operator splitting method for large convex problems. In *European Control Conference*, 2019.

[28] Andrew Gilpin, Javier Pena, and Tuomas 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.

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

[30] Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp M. Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, Marco Lübbecke, Hans D. Mittelmann, Derya Ozyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*, 2021.

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

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

[33] Tom Goldstein, Min Li, Xiaoming Yuan, Ernie Esser, and Richard Baraniuk. Adaptive primal-dual hybrid gradient methods for saddle-point problems. *arXiv preprint arXiv:1305.0546*, 2013.

[34] Bingsheng He and Xiaoming 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.

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

[36] Guanghui Lan, Zhaosong Lu, and Renato 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.

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

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

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

[40] Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, Ravichandra Addanki, Tharindi Hapuarachchi, Thomas Keck, James Keeling, Pushmeet Kohli, Ira Ktena, Yujia Li, Oriol Vinyals, and Yori Zwols. Solving Mixed Integer Programs Using Neural Networks. *arXiv preprint arXiv:2012.13349*, December 2020.

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

[42] Arkadi 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.

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

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

[45] Yurii Nesterov and Arkadii Nemirovskii. *Interior-point polynomial algorithms in convex programming*, volume 13. SIAM, 1994.

[46] 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.

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

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

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

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

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

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

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

[54] Thomas Pock, Daniel Cremers, Horst Bischof, and Antonin Chambolle. An algorithm for minimizing the mumford-shah functional. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1133–1140. IEEE, 2009.

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

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

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

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

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

[60] 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.

[61] Yuhsiang M. Tsai, Terry Cojean, and Hartwig Anzt. Sparse linear algebra on AMD and NVIDIA GPUs – the race is on. In Ponnuswamy Sadayappan, Bradford L. Chamberlain, Guido Juckeland, and Hatem Ltaief, editors, *High Performance Computing*, pages 309–327, Cham, 2020. Springer International Publishing.

[62] Robert J Vanderbei et al. *Linear programming*, volume 3. Springer, 2015.

[63] Sinong Wang and Ness 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.

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

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

A Proof of scale invariance of primal weight initialization scheme

Proposition 1. *Suppose that $\hat{K} = \gamma K$, $\hat{c} = \gamma \alpha_y c$, $\hat{q} = \gamma \alpha_x q$, $\hat{l} = \alpha_x l$, and $\hat{u} = \alpha_x u$ for $\alpha_y, \alpha_x, \gamma \in (0, \infty)$ with $\|c\|_2, \|q\|_2, \|\hat{c}\|_2, \|\hat{q}\|_2 > \epsilon_{mach}$. Consider the PDHG algorithm given in (3) with $\omega =$ `InitializePrimalWeight(`c, q`)`. Let z^k be the PDHG iterates on the original problem and $\hat{z}^k$ be the PDHG iterates on the scaled problem with $\hat{x}^0 = \alpha_x x^0$ and $\hat{y}^0 = \alpha_y y^0$, then: $\hat{x}^k = \alpha_x x^k, \hat{y}^k = \alpha_y y^k$ for all $k \in \{0\} \cup \mathbb{N}$.*

Proof. We will prove this by induction. By definition the result holds for $k = 0$. Define $\hat{\eta} = \eta/\gamma$, and $\hat{\omega} = \|\hat{c}\|/\|\hat{q}\|_2 = \omega(\alpha_y/\alpha_x)$. Then,

$$\begin{aligned}
\hat{x}^{k+1} &= \operatorname*{\mathbf{proj}}_{\hat{X}} \left(\hat{x}^k - \hat{\eta}/\hat{\omega}(\hat{c} - \hat{K}^\top \hat{y}^k) \right) \\
&= \operatorname*{\mathbf{proj}}_{\hat{X}} \left(\alpha_x x^k - \alpha_x \eta/\omega(c - K^\top y^k) \right) \\
&= \alpha_x \operatorname*{\mathbf{proj}}_{X} \left(x^k - \eta/\omega(c - K^\top y^k) \right) \\
&= \alpha_x x^{k+1}.
\end{aligned}$$

Table 3: Performance statistics: MIP Relaxations Stepsize Tol 1E-04

Experiment	Solved count	KKT passes SGM10	Solve time secs SGM10
Fixed step-size	336	6207.9	71.4
PDLP	349	3058.3	42.2
Best fixed MP setting	352	3855.6	49.2
Best per-instance MP settings	363	2869.7	36.8

Similarly,

$$\begin{aligned}\hat{y}^{k+1} &= \underset{\hat{Y}}{\mathbf{proj}}\left(\hat{y}^k - \hat{\eta}\hat{\omega}(\hat{q} - \hat{K}(2\hat{x}^{k+1} - \hat{x}^k))\right)\\ &= \underset{\hat{Y}}{\mathbf{proj}}\left(\alpha_y y^k - \alpha_y \eta\omega(q - K(2x^{k+1} - x^k))\right)\\ &= \alpha_y \underset{Y}{\mathbf{proj}}\left(\alpha_y y^k - \alpha_y \eta\omega(q - K(2x^{k+1} - x^k))\right)\\ &= \alpha_y y^{k+1}.\end{aligned}$$

□

B `MIP Relaxation` dataset

MIPLIB 2017 [30] is a collection of mixed integer programming (MIP) problems used primarily for developing and benchmarking MIP solvers. MIPLIB contains both a larger collection set (1056 instances) and a smaller benchmark set (240 instances). We select 383 instances from the collection set that satisfy the following criteria:

- Not tagged as numerically unstable
- Not tagged as infeasible
- Not tagged as having indicator constraints
- Finite optimal objective (if known)
- The constraint matrix has between $100,000$ and $10,000,000$ nonzero coefficients.

For comparison, the MIPLIB benchmark set excludes instances whose constraint matrix has more than $1,000,000$ nonzero coefficients. The upper limit of $10,000,000$ was chosen for the convenience of running experiments. Our set both excludes small instances that may be in the benchmark set and includes instances deemed too large for the benchmark set. From each MIP instance we derive an LP instance by removing the integrality constraints.

C Ablation study

To study the impact of PDLP's improvements over baseline PDHG, we performed an ablation study, in which we evaluate the consequences of disabling each enhancement separately and evaluate alternative choices. All experiments in this section are performed on the `MIP Relaxations` dataset.

C.1 Step size choice

We compare PDLP's adaptive step size rule against three alternatives:

- "Fixed step size": (baseline PDHG) The step size η is fixed to $\eta = 0.9/\|K\|_2$ where $\|K\|_2$ is estimated via power iteration,

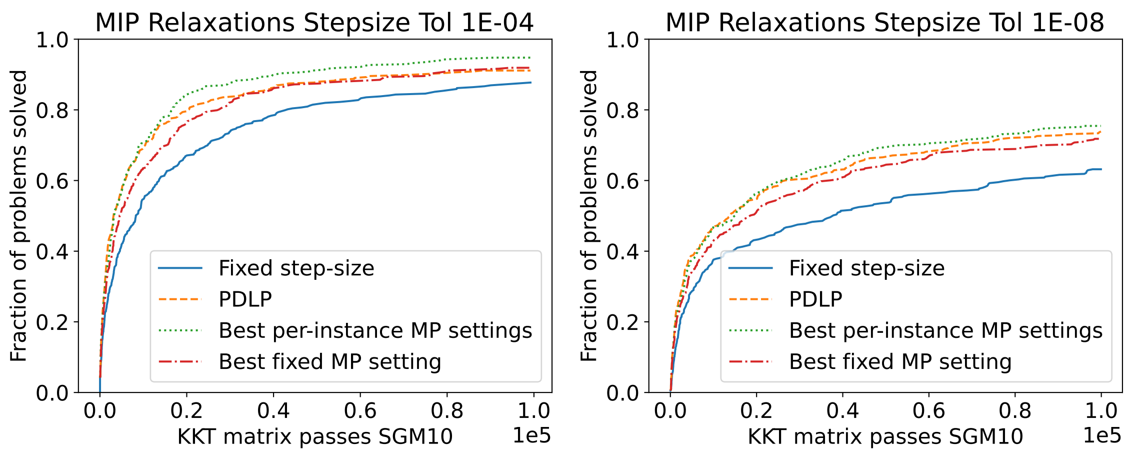


Figure 3: Step size ablation experiments on `MIP Relaxations`

Table 4: Performance statistics: MIP Relaxations Stepsize Tol 1E-08

Experiment	Solved count	KKT passes SGM10	Solve time secs SGM10
Fixed step-size	242	17339.5	238.3
Best fixed MP setting	275	11660.4	153.1
PDLP	283	9773.3	131.6
Best per-instance MP settings	289	9778.7	121.4

- "Best fixed Malitsky-Pock (MP) setting": Malitsky and Pock [37], tuning the hyperparameters via a hyperparameter search, and
- "Best per-instance Malitsky-Pock (MP) setting": Malitsky and Pock [37], choosing the best hyperparameters separately for each instance. This is a "virtual" solver that combines 42 hyperparameter configurations.

The results, in Figure 3 and Tables 3 and 4, show that PDLP is slightly better than tuned Malitsky-Pock, and at high accuracy, almost as good as per-instance tuned Malitsky-pock.

Description of Malitsky and Pock hyperparameters. Our implementation depends on three hyperparameters: `breaking_factor`, `downscaling_factor`, and `interpolation_coefficient`. We explain the role of each one by summarizing the linesearch rule. Suppose the algorithm finished iteration k and it does not execute a restart. Thus, the primal weight doesn't not change $\omega_k = \omega_{k+1}$. Mimicking the notation in [37] we define:

$$\theta_k = \frac{\eta_{k-1}}{\eta_k}.$$

Then, the algorithm does the following at iteration $k+1$:

1. Update primal iterate $x^{k+1} \leftarrow \mathbf{proj}_X \left(x^k - \frac{\eta_k}{\omega_k}\left(c - K^\top y^k\right)\right)$.
2. Pick a candidate for the next step size $\widehat{\eta}_{k+1} \in [\eta_k, \sqrt{1+\theta_k}\eta_k]$. By letting

$$\widehat{\eta}_{k+1} \leftarrow \eta_k + \texttt{interpolation\_coefficient} \cdot \left(\sqrt{1+\theta_k} - 1\right)\eta_k \quad \text{and} \quad \widehat{\theta}_{k+1} \leftarrow \frac{\eta_k}{\widehat{\eta}_{k+1}}.$$

3. Compute a candidate for next dual iterate y_{k+1}:

$$\widehat{y}^{k+1} \leftarrow \mathop{\mathbf{proj}}_{Y}\left(y^k + \omega_{k+1}\widehat{\eta}_{k+1}\left(q - K\left(x^{k+1} + \widehat{\theta}_{k+1}(x^{k+1} - x^k)\right)\right)\right).$$

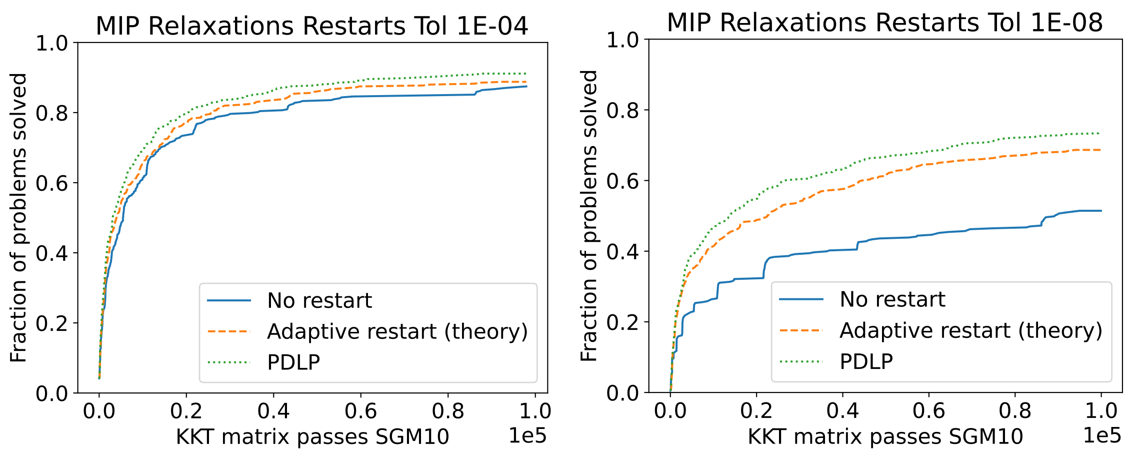


Figure 4: Restart ablation experiments on `MIP Relaxations`

Table 5: Performance statistics: MIP Relaxations Restarts Tol 1E-04

Experiment	Solved count	KKT passes SGM10	Solve time secs SGM10
No restart	335	4387.9	54.1
Adaptive restart (theory)	340	3680.5	50.6
PDLP	349	3058.3	42.1

4. Check if the linesearch is done;

If $\widehat{\eta}_k \|K^\top(\widehat{y}^{k+1} - y^k)\| \le \texttt{breaking\_factor} \cdot \|\widehat{y}^{k+1} - y^k\|$:

$$\eta_{k+1} \leftarrow \widehat{\eta}_{k+1}, \quad \theta_{k+1} \leftarrow \widehat{\theta}_{k+1}, \quad \text{and} \quad y^{k+1} \leftarrow \widehat{y}^{k+1}.$$

Else: reduce the step size as follows and then **go to** Step 3:

$$\widehat{\eta}_{k+1} \leftarrow \texttt{downscaling\_factor} \cdot \widehat{\eta}_{k+1}, \quad \widehat{\theta}_{k+1} \leftarrow \frac{\eta_k}{\widehat{\eta}_{k+1}}.$$

In our experiments, we fix `breaking_factor` $= 1$ on guidance from the authors of [37]. We then perform a grid search on `downscaling_factor` $\in \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ and `interpolation_coefficient` $\in \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$.

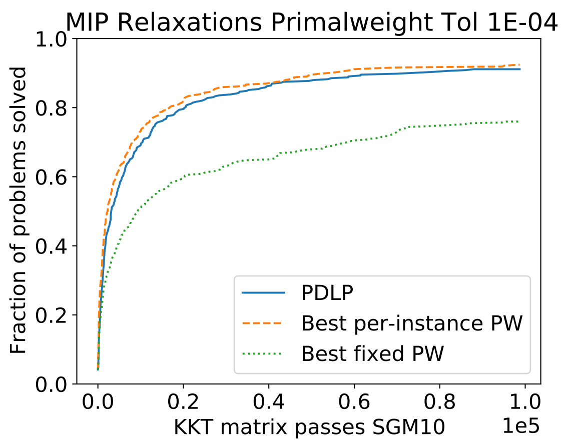
The single best configuration (by count of solved instances) for both $\epsilon = 10^{-4}$ and $\epsilon = 10^{-8}$ is `downscaling_factor` $=$ 0.5 and `interpolation_coefficient` $= 0.4$.

C.2 Adaptive restarts

For PDLP, we use $\beta_{\text{sufficient}} = 0.9$, $\beta_{\text{necessary}} = 0.1$, and $\beta_{\text{artificial}} = 0.5$ as the restart parameters. For "Adaptive restart (theory)" mode we match [5] and by setting $\beta_{\text{sufficient}} = \beta_{\text{necessary}} = 0.37 = \exp(-1)$. This

Table 6: Performance statistics: MIP Relaxations Restarts Tol 1E-08

Experiment	Solved count	KKT passes SGM10	Solve time secs SGM10
No restart	197	22488.2	397.0
Adaptive restart (theory)	263	12175.6	171.5
PDLP	283	9773.3	131.4



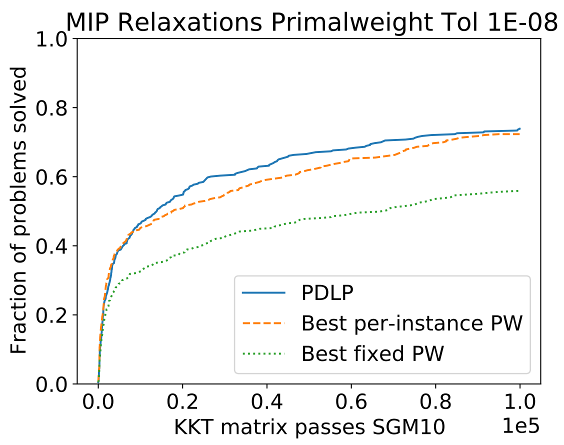


Figure 5: Primal weight ablation experiments on `MIP Relaxations`

Table 7: Performance statistics: MIP Relaxations Primalweight Tol 1E-04

Experiment	Solved count	KKT passes SGM10	Solve time secs SGM10
Best fixed PW	291	6548.0	116.5
PDLP	349	3058.3	42.5
Best per-instance PW	354	2091.3	34.8

is equivalent to removing condition (ii) from the restart criteria. For "no restart" mode we disable restarts. For this setting primal weight updates still occur when an artificial restart would have been triggered. In other words, the primal weights are updated on iteration $2, 2^2, 2^3, \ldots$.

The performance of adaptive restarts are summarized in Figure 4 and Tables 5 and 6. We can see PDLP outperforms "Adaptive restart (theory)" mode which in turn beats 'no restart" mode. This difference is much more pronounced at high accuracy.

C.3 Primal weight updates

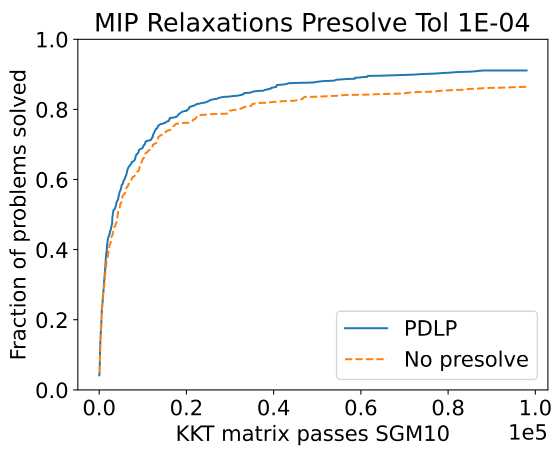
In PDLP, the smoothing parameter is set to $\theta = 0.5$. As baselines for PDLP's primal weight (PW) updating rule, we compare with using fixed primal weights, setting the primal weight to $\omega = \xi \cdot \texttt{InitializePrimalWeight}(c,\ q)$ with the bias $\xi \in \{10^{-5}, 10^{-4}, \ldots, 10^{0}, \ldots, 10^{4}, 10^{5}\}$ chosen by grid search. For these experiments, the smoothing parameter is set to $\theta = 0$ to fix the primal weight during the solve.

We compute both the single best value of ξ (by count of solved instances), and the best per-instance value, which defines a "virtual" solver. The single best value of ξ is 0.1 at both $\epsilon = 10^{-4}$ and $\epsilon = 10^{-8}$. Qualitatively, the performance of $\xi = 1$, which is a natural default, is very similar to that of $\xi = 0.1$.

From Figure 5 and Tables 7 and 8, we conclude that PDLP is competitive with the best per-instance fixed primal weight at low accuracy, and outperforms it at high accuracy.

Table 8: Performance statistics: MIP Relaxations Primalweight Tol 1E-08

Experiment	Solved count	KKT passes SGM10	Solve time secs SGM10
Best fixed PW	214	17852.7	315.6
Best per-instance PW	277	9846.8	146.0
PDLP	283	9773.3	132.2



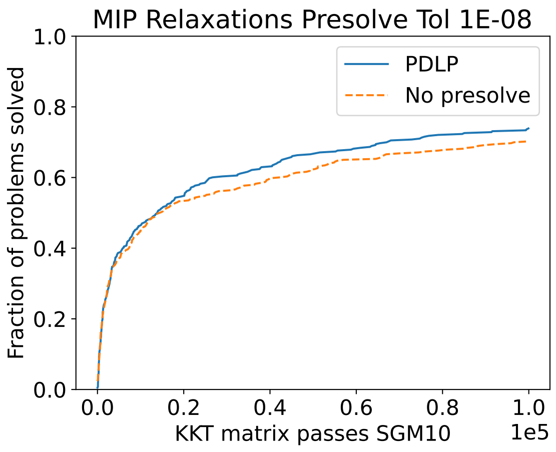


Figure 6: Presolve ablation experiments on `MIP Relaxations`

Table 9: Performance statistics: MIP Relaxations Presolve Tol 1E-04

Experiment	Solved count	KKT passes SGM10	Solve time secs SGM10
No presolve	332	3615.3	68.7
PDLP	349	3058.3	42.1

C.4 Presolve

Figure 6 and Tables 9 and 10 measure the impact of presolve. Note that the impact on solve time is greater than the impact on KKT passes, because presolve also makes each KKT pass faster by making the problem smaller.

C.5 Diagonal preconditioning

Tables 11 and 12 compare the performance of the four diagonal preconditioning techniques as mentioned in Section 3.5. As we can see, the number of solved problems of our proposed preconditioner (Ruiz and Pock-Chambolle) significantly outperform no scaling and the baselines (Pock-Chambolle or Ruiz individually).

Furthermore, Figure 7 shows the number of solved instances as a function of KKT passes for the four different diagonal preconditioners, which further shows a clear separation between PDLP and the baselines.

Table 10: Performance statistics: MIP Relaxations Presolve Tol 1E-08

Experiment	Solved count	KKT passes SGM10	Solve time secs SGM10
No presolve	269	10030.6	183.2
PDLP	283	9773.3	131.5

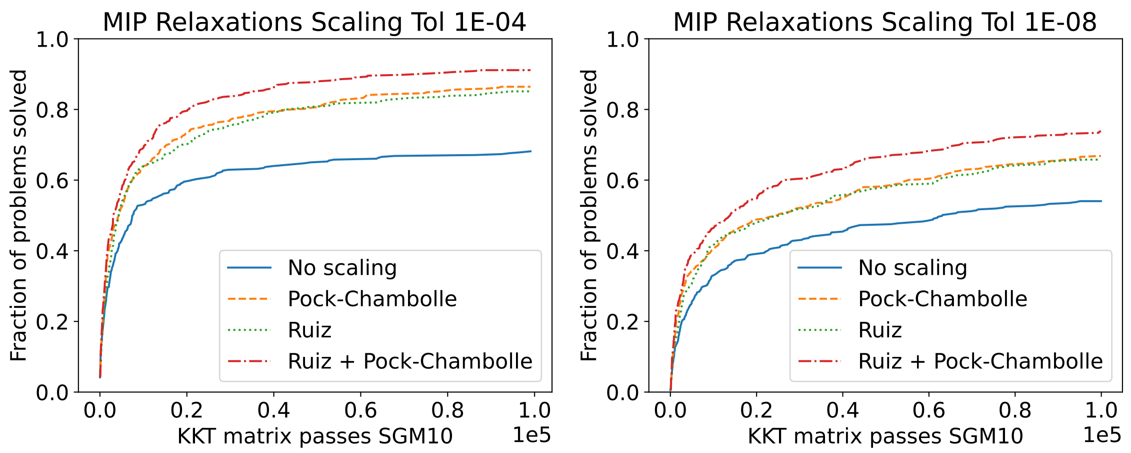


Figure 7: Diagonal preconditioning ablation experiments on `MIP Relaxations`

Table 11: Performance statistics: MIP Relaxations Scaling Tol 1E-04

Experiment	Solved count	KKT passes SGM10	Solve time secs SGM10
No scaling	261	6700.9	139.4
Ruiz	326	4487.7	65.3
Pock-Chambolle	331	3941.3	59.0
Ruiz + Pock-Chambolle	349	3058.3	42.0

Table 12: Performance statistics: MIP Relaxations Scaling Tol 1E-08

Experiment	Solved count	KKT passes SGM10	Solve time secs SGM10
No scaling	207	19770.0	340.3
Ruiz	252	14028.2	201.4
Pock-Chambolle	256	12960.0	197.8
Ruiz + Pock-Chambolle	283	9773.3	132.9

Table 13: Performance statistics: MIP Relaxations Improvements Tol 1E-04

Experiment	Solved count	KKT passes SGM10	Solve time secs SGM10
PDHG	113	38958.0	1088.3
+ restarts	140	29739.6	770.4
+ scaling	221	14801.5	313.6
+ primal weight	315	7228.1	110.8
+ step size	332	3615.3	67.6
+ presolve (= PDLP)	349	3058.3	42.1

Table 14: Performance statistics: MIP Relaxations Improvements Tol 1E-08

Experiment	Solved count	KKT passes SGM10	Solve time secs SGM10
PDHG	48	68588.8	2232.6
+ restarts	101	47301.1	1284.0
+ scaling	162	25985.7	595.7
+ primal weight	223	18273.4	331.3
+ step size	269	10091.0	181.8
+ presolve (= PDLP)	283	9773.3	131.5

D Additional details on the PageRank LP formulation

Based on Nesterov [43], we formulate the problem of finding a maximal right eigenvector of a stochastic matrix S as a feasible solution of the LP problem:

$$\begin{array}{rlcl} \text{find} & x & & \\ \text{subject to:} & Sx & \leq & x \\ & \mathbf{1}^\top x & = & 1 \\ & x & \geq & 0 \end{array} \tag{7}$$

Nesterov [43] states the constraint $\|x\|_\infty \geq 1$ to enforce $x \neq 0$. We instead use $\mathbf{1}^\top x = 1$ which is equivalent under scaling.

For a random scalable collection of pagerank instances, we used Barabási-Albert [7] preferential attachment graphs, using the Julia `LightGraphs.SimpleGraphs.barabasi_albert` generator with degree set to 3. We then computed the adjacency matrix and scaled the columns to make the matrix stochastic; call this matrix S'. Following the standard PageRank formulation we apply a damping factor to S' and consider $S := \lambda S' + (1-\lambda)J/n$ (where $J = \mathbf{1}\mathbf{1}^\top$ is the all-ones matrix). Intuitively, S encodes a random walk that follows a link in the graph with probability λ or jumps to a uniformly random node with probability $1-\lambda$.

The direct approach to the damping factor results in a completely dense matrix. Instead we use the fact that $Jx = 1$ to rewrite the constraint $Sx \leq x$ in (7) as

$$\lambda(S'x)_i + (1-\lambda)/n \;\leq\; x_i \;\; \forall i \;\; . \tag{8}$$

E Additional PDLP improvements results

Tables 13 and 14 give a tabular version of the impact of PDLP's improvements on the `MIP Relaxations` dataset (corresponding to Figure 1a). Tables 15 and 16 give a tabular version of the impact of PDLP's improvements on the `LP benchmark` dataset (corresponding to Figure 1b).

Table 15: Performance statistics: LP Benchmark Improvements Tol 1E-04

Experiment	Solved count	KKT passes SGM10	Solve time secs SGM10
PDHG	10	64120.0	2148.5
+ restarts	10	58285.8	2033.9
+ scaling	17	44984.7	1600.3
+ primal weight	37	22232.1	880.1
+ step size	36	13003.7	542.5
+ presolve (= PDLP)	36	14721.1	630.7

Table 16: Performance statistics: LP Benchmark Improvements Tol 1E-08

Experiment	Solved count	KKT passes SGM10	Solve time secs SGM10
PDHG	4	87478.5	2784.6
+ restarts	7	69299.7	2210.4
+ scaling	10	58808.8	1929.2
+ primal weight	14	52872.7	1644.8
+ step size	23	38630.3	1336.5
+ presolve (= PDLP)	23	35106.0	1281.7

Table 17: Performance statistics: MIP Relaxations Baselines Tol 1E-04

Experiment	Solved count	KKT passes SGM10	Solve time secs SGM10
PDHG	159	45720.5	922.9
SCS (matrix-free)	287	37027.2	257.0
SCS	317	-	149.7
Enh. Extragradient	351	6028.7	75.2
PDLP	371	3236.6	38.4

Table 18: Performance statistics: MIP Relaxations Baselines Tol 1E-08

Experiment	Solved count	KKT passes SGM10	Solve time secs SGM10
PDHG	80	79085.1	2026.7
SCS (matrix-free)	124	40486.1	1006.9
SCS	156	-	675.3
Enh. Extragradient	302	21216.9	207.4
PDLP	334	11381.1	106.4

Table 19: Performance statistics: LP Benchmark Baselines Tol 1E-04

Experiment	Solved count	KKT passes SGM10	Solve time secs SGM10
PDHG	5	98130.4	3510.9
SCS (matrix-free)	25	51040.9	1118.7
SCS	26	-	1155.6
Enh. Extragradient	27	25808.3	944.2
PDLP	32	16679.4	613.8

Table 20: Performance statistics: LP Benchmark Baselines Tol 1E-08

Experiment	Solved count	KKT passes SGM10	Solve time secs SGM10
PDHG	1	99962.1	3584.1
SCS (matrix-free)	7	63771.2	2155.6
SCS	9	-	2017.1
Enh. Extragradient	13	54795.9	1693.4
PDLP	22	37937.0	1213.4

F Additional baseline comparison results

Tables 17 and 18 give a tabular version of the comparison of PDLP with other first-order baselines on the `MIP Relaxations` dataset, and Tables 19 and 20 give a tabular version of the comparison of PDLP with other first-order baselines on the `LP benchmark` dataset (corresponding to Figure 2).

G Instructions for reproducing experiments

This section documents the precise commands and command-line arguments for each experiment in the paper. These instructions are supplemental to the READMEs in the `FirstOrderLp` code directory. We assume that readers have already followed the instructions in the READMEs to set up and "instantiate" the Julia environment, and collect or generate all the datasets. Examples assume that the current working directory is `FirstOrderLp`.

The full suite of experiments takes approximately 11,000 CPU-hours to run, and hence requires use of a cluster or cloud computing environment. Given the idiosyncrasies of these environments, we do not provide additional utilities for distributing the experiments. See Section 4.1 for details on the computing platforms we used.

The following base invocations show how to run the two main scripts without any custom arguments.

Listing 1: `solve_qp.jl` base invocation

```
julia --project=scripts scripts/solve_qp.jl
      --instance_path=$INSTANCE
      --output_dir=$OUTPUT_DIR
```

Listing 2: `solve_lp_external.jl` base invocation

```
julia --project=scripts scripts/solve_lp_external.jl
      --instance_path=$INSTANCE
      --output_dir=$OUTPUT_DIR
```

`solve_qp.jl` runs methods implemented in the `FirstOrderLp` module. `solve_lp_external.jl` runs external solvers (specifically, SCS).

G.1 Tolerances

In experiments we often solve at termination tolerances $\epsilon = 10^{-4}$ and $\epsilon = 10^{-8}$. The following command-line arguments to `solve_qp.jl` are used to set these tolerances.

Listing 3: `solve_qp.jl` arguments for $\epsilon = 10^{-4}$

```
--relative_optimality_tol 1e-4 --absolute_optimality_tol 1e-4
```

Listing 4: `solve_qp.jl` arguments for $\epsilon = 10^{-8}$

```
--relative_optimality_tol 1e-8 --absolute_optimality_tol 1e-8
```

G.2 Improvements experiment

This section documents the command-line settings corresponding to the experiments in Section 4.2 that measure the impact of PDLP's improvements over baseline PDHG.

The following common settings apply for each run:

Listing 5: Common settings for each run

```
--kkt_matrix_pass_limit=100000 --restart_to_current_metric=gap_over_distance
--verbosity=0 --method=pdhg
```

For each solve, use the base invocation for `solve_qp.jl` (Listing 1), a tolerance setting (Section G.1), common settings (Listing 5), and one set of parameters below. See the documentation in READMEs and source code for how to set `$OUTPUT_DIR` and process the results.

For example, the following command solves the "+ scaling" setting with $\epsilon = 10^{-4}$:

```
julia --project=scripts scripts/solve_qp.jl
      --instance_path=$INSTANCE
      --output_dir=$OUTPUT_DIR
      --relative_optimality_tol=1e-4
      --absolute_optimality_tol=1e-4
      --kkt_matrix_pass_limit=100000
      --restart_to_current_metric=gap_over_distance
      --verbosity=0 --method=pdhg
      --step_size_policy=constant
      --primal_weight_update_smoothing=0.0
      --scale_invariant_initial_primal_weight=false
```

Parameter settings:

1. "PDHG": (on original un-presolved dataset)

```
--step_size_policy=constant --l_inf_ruiz_iterations=0
--pock_chambolle_rescaling=false --l2_norm_rescaling=false
--restart_scheme=no_restart --primal_weight_update_smoothing=0.0
--scale_invariant_initial_primal_weight=false
```

2. "+ restarts":

```
--step_size_policy=constant --l_inf_ruiz_iterations=0
--pock_chambolle_rescaling=false --l2_norm_rescaling=false
--primal_weight_update_smoothing=0.0 --scale_invariant_initial_primal_weight=false
```

3. "+ scaling":

```
--step_size_policy=constant --primal_weight_update_smoothing=0.0
--scale_invariant_initial_primal_weight=false
```

4. "+primal weight":

```
--step_size_policy=constant
```

5. "+step size": No additional parameters
6. "+presolve (= PDLP)": Switch to presolved dataset.

G.3 Comparison with other first-order baselines

This section documents the command-line settings corresponding to the experiments in Section 4.3 that compare PDLP with SCS and enhanced Extragradient.

G.3.1 SCS (`solve_lp_external.jl`)

SCS is invoked via `solve_lp_external.jl`. The following common settings apply for all SCS runs:

Listing 6: Common settings for SCS runs

```
--scs-normalize=true --iteration_limit=1000000000
```

Because SCS does not support time limits, we use the `timeout` command to stop SCS after one hour. For example:

```
timeout 1h julia --project=scripts scripts/solve_lp_external.jl
--solver=scs-direct ...
```

The following arguments[2] are used to set $\epsilon = 10^{-4}$.

Listing 7: SCS arguments for $\epsilon = 10^{-4}$

```
--tolerance=1e-4 --scs-acceleration_lookback=0
```

The following arguments are used to set $\epsilon = 10^{-8}$.

Listing 8: SCS arguments for $\epsilon = 10^{-8}$

```
--tolerance=1e-8
```

The following arguments[3] select SCS in matrix-free mode:

Listing 9: Configuration for SCS (matrix-free)

```
--solver=scs-indirect --scs-cg_rate=1.01
```

The following arguments select SCS in its default mode that uses a cached LDL factorization to solve the linear system that arises at each iteration:

Listing 10: Configuration for SCS (default)

```
--solver=scs-direct
```

G.3.2 PDLP and Extragradient (`solve_qp.jl`)

PDLP and enhanced Extragradient are invoked via `solve_qp.jl`.

The following common settings apply to both PDLP and Extragradient.

Listing 11: Common settings for PDLP and Extragradient

```
--time_sec_limit=3600 --restart_to_current_metric=gap_over_distance --verbosity=0
```

The following two settings select either the PDLP or enhanced Extragradient methods.

Listing 12: Configuration for PDLP

```
--method=pdhg
```

Listing 13: Configuration for enhanced Extragradient

```
--method=mirror-prox
```

[2] In preliminary experiments on the MIP relaxations dataset, SCS performed better at 10^{-4} with this custom setting of acceleration lookback, which disables Anderson Acceleration.

[3] In preliminary experiments on the MIP relaxations dataset, SCS (matrix-free) performed better with `cg_rate=1.01`, which controls the rate at which the conjugate gradient convergence tolerance decreases as a function of the iteration number.

G.4 PDLP versus simplex and barrier

This section lists the commands corresponding to the experiments in Section 4.4 that compare PDLP with Gurobi's simplex and barrier algorithms.

Listing 14: Command for Gurobi Barrier

```
gurobi_cl TimeLimit=3600 Method=2 Crossover=0 Threads=1 $INSTANCE
```

Listing 15: Command for Gurobi Primal Simplex

```
gurobi_cl TimeLimit=3600 Method=0 Threads=1 $INSTANCE
```

Listing 16: Command for Gurobi Dual Simplex

```
gurobi_cl TimeLimit=3600 Method=1 Threads=1 $INSTANCE
```

Listing 17: Command for PDLP

```
julia --project=scripts scripts/solve_qp.jl
      --instance_path=$INSTANCE
      --output_dir=$OUTPUT_DIR
      --relative_optimality_tol=1e-8
      --absolute_optimality_tol=1e-8
      --time_sec_limit=3600
      --restart_to_current_metric=gap_over_distance
      --verbosity=0
      --method=pdhg
```

G.5 Large-scale application: PageRank

This section describes the commands corresponding to the experiments in Section 4.5 that compares PDLP, SCS, and Gurobi's methods on PageRank instances.

The commands for Gurobi methods are the same as in Listings 14, 15, and 16. The command for PDLP is the same as in Listing 17. The command for SCS follows:

Listing 18: Command for SCS

```
timeout 1h julia --project=scripts scripts/solve_lp_external.jl
      --instance_path=$INSTANCE
      --output_dir=$OUTPUT_DIR
      --scs-normalize=true
      --iteration_limit=1000000000
      --tolerance=1e-8
      --solver=scs-indirect
      --scs-cg_rate=1.01
```

G.6 Ablation study

In the ablation study, PDLP is invoked as:

Listing 19: PDLP configuration for the ablation study

```
  julia --project=scripts scripts/solve_qp.jl
        --instance_path=$INSTANCE
        --output_dir=$OUTPUT_DIR
        --relative_optimality_tol $TOLERANCE
        --absolute_optimality_tol $TOLERANCE
        --method=pdhg
        --restart_to_current_metric=gap_over_distance
        --kkt_matrix_pass_limit=100000
        --verbosity=0
```

on the `MIP Relaxations` dataset (to which presolve has been applied).

G.7 Step size choice

This section describes the commands corresponding to the ablation experiments in Section C.1 on the step size choice.

The fixed step-size rule is invoked by appending the following argument to the command in Listing 19:

```
--step_size_policy=constant
```

The Malitsky and Pock step size rule is invoked by appending the following arguments to the command in Listing 19:

```
--step_size_policy=malitsky-pock
--malitsky_pock_breaking_factor=1.0
--malitsky_pock_downscaling_factor=$DOWNSCALING_FACTOR
--malitsky_pock_interpolation_coefficient=$INTERPOLATION_COEFFICIENT
```

G.7.1 Adaptive restarts

This section describes the commands corresponding to the ablation experiments in Section C.2 on restarts.

The "No restart" setting is invoked by appending the following argument to the command in Listing 19:

```
--restart_scheme=no_restart
```

The "Adaptive restart (theory)" setting is invoked by appending the following arguments to the command in Listing 19:

```
--restart_to_current_metric=no_restart_to_current
--sufficient_reduction_for_restart=0.37 --necessary_reduction_for_restart=0.37
```

G.7.2 Primal weight updates

This section describes the commands corresponding to the ablation experiments in Section C.3 on primal weights.

The primal weight is fixed, with the bias `$XI` $= \xi$, by appending the following arguments to the command in Listing 19:

```
--primal_weight_update_smoothing=0.0
--primal_importance=$XI
```

G.7.3 Presolve

For the presolve ablation study in Section C.4, the "No presolve" setting is evaluated by applying PDLP to the original (non-presolved) version of the `MIP Relaxations` dataset. See `benchmarking/README.md` for more information on the dataset generation.

G.7.4 Diagonal preconditioning

This section describes the commands corresponding to the ablation experiments in Section C.5 on diagonal preconditioning.

The "No scaling" setting corresponds to appending the following arguments to the command in Listing 19:

```
--l_inf_ruiz_iterations=0
--pock_chambolle_rescaling=false
```

The “Ruiz” setting corresponds to appending the following argument to the command in Listing 19:

```
--pock_chambolle_rescaling=false
```

The “Pock-Chambolle” setting corresponds to appending the following argument to the command in Listing 19:

```
--l_inf_ruiz_iterations=0
```

The “Ruiz + Pock-Chambolle” setting is PDLP.