arXiv:2603.03150v1 [math.OC] 3 Mar 2026

混合 PDHG 与内点法

Edward Rothberg
Gurobi Optimization, LLC
rothberg@gurobi.com

2026年3月3日

摘要

原始-对偶混合梯度（PDHG）算法是一种一阶方法，可利用图形处理器求解大规模线性规划问题。这种方法通常比其他方法（单纯形法和内点法）更快，但代价往往是精度低得多。本文探讨 PDHG 能否与内点法混合，从而在保留前者部分速度优势的同时，兼具后者的精度优势。

1 引言

原始-对偶混合梯度（PDHG）算法 [3] 近来受到了大量关注，主要是因为它提供了一种算法框架，可用于构建适合图形处理器的方法来求解大规模线性规划（LP）问题。对该方法的近期改进 [2, 3, 4, 10, 11] 使其在求解此类问题时，已能与当前最先进算法——单纯形法 [5] 和内点法 [16]——相当具有竞争力。不过，作为一阶方法，PDHG的收敛可能相当缓慢。因此，它通常会在所得解的精度远低于其他方法所计算解的精度时终止。

本文探讨能否将PDHG与内点法混合，以同时获得两种方法的部分优势。具体而言，我们先用PDHG计算一个低精度解，再用该解对内点法进行热启动。内点法以难以热启动而著称 [7, 8, 9, 16, 17], ，但这种名声形成于不同的背景。早期工作试图利用一个线性规划的最优解来加速求解另一个相似的线性规划。从同一问题的近似解出发这一任务尚未受到太多关注，大概是因为像PDHG这样能够快速产生近最优解的方法此前并不具有竞争力。

当然，还有其他方法可以提高 PDHG 解的精度。最显而易见的方法是收紧 PDHG 收敛准则，让它有更多时间提升解的精度。另一种方法是将 PDHG 解输入交叉 [13]过程，通过执行类单纯形过程来计算精度高得多的基本解。每种方法都有各自的权衡。其中之一当然是运行时间与精度之间的权衡；另一个

则是每种方法对多核或 GPU 并行的适用程度。本文将探讨几种此类替代方案。

我们发现，一种从低精度 PDHG 解启动内点算法的低成本策略既快速又可靠。与冷启动方法相比，迭代次数平均减少约 3 倍，热启动版本在超过一半的情况下以少于 10 次迭代求解。所得内点解的违反量比其起始的 PDHG 解小 3 到 4 个数量级。

2 背景

A l线性规划问题可以以其原始形式和对偶形式表述 a s:

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned} \qquad\qquad \begin{aligned} \max \quad & b^\top y \\ \text{s.t.} \quad & A^\top y + z = c \\ & z \geq 0 \end{aligned}$$

约束矩阵 A 有 m 行和 n 列，目标向量 c 的长度为 n，右端项向量 b 的长度为 m。向量 x和 y 分别是原始解和对偶解，向量 z 提供约化成本 （z_j 给出目标函数相对于 x_j 单位变化的变化率）。

实际的线性规划模型通常比这种标准形式更为复杂，往往包含不等式以及对变量非平凡的下界和上界。这些都可以轻松映射到上述形式（并且通常在线性规划求解器实现中被隐式处理）。

任意一组三个向量（x、 y、 z）都对应着原始和对偶残差向量：

$$\begin{aligned} r_P &= b - Ax \\ r_D &= c - A^\top y - z \end{aligned}$$

当 (i) 原始和对偶残差向量均为零，(ii) x和 z 均非负，且 (iii) 目标间隙为零时，解即为最优解。最后一个条件可直接表示为 $c^\top x = b^\top y$，也可用不那么直接的形式表示为 $x^\top z = 0$ （称为互补性）。

PDHG 算法会维护当前迭代点，并持续迭代，直到所有三个最优性条件都满足目标容差（通常以相对形式表示）：

$$\begin{aligned} \|r_P\|_2 &\leq \varepsilon_{\text{rel}}(1 + \|b\|_2) \\ \|r_D\|_2 &\leq \varepsilon_{\text{rel}}(1 + \|c\|_2) \\ |c^\top x - b^\top y| &\leq \varepsilon_{\text{rel}}(1 + |c^\top x| + |b^\top y|) \end{aligned}$$

评估这些方法性能时最常用的收敛容差是 $\varepsilon_{\text{rel}} = 10^{-4}$。

PDHG 是一种一阶方法，这在实际应用中意味着收敛速度可能相当慢。关于这种方法，最有力的结论是：当问题具备某些特性时，收敛速度呈线性。

当问题具备某些特性时，收敛速度呈线性。更严格的收敛容差通常会显著增加运行时间。

内点法也会维护当前迭代点，而该迭代点未必满足可行性或互补性目标。内点法具有局部二次收敛的特性，这意味着一旦迭代点“足够接近”最优解，它们就会迅速收敛。收敛理论对达到收敛所需的迭代次数给出了严格的界限 [16]。这些界限基于相对容差，但实际实现通常会继续迭代，远超收敛理论所建议的容差要求。

单纯形法和交叉方法的容差通常以绝对值表示（$\|r_P\|_\infty \le \varepsilon_{\text{abs}}$ 和 $\|r_D\|_\infty \le \varepsilon_{\text{abs}}$），传统的目标值为 $\varepsilon_{\text{abs}} = 10^{-6}$。（原始）单纯形算法会维持（原始）可行性和零互补性（在数值上可行的范围内），因此其迭代只专注于满足这三个最优性条件之一。与 PDHG 相比，由于绝对目标值更小且采用无穷范数，单纯形法的终止条件严格得多。

2.1 预求解与缩放

在实际的线性规划求解器中，线性规划算法所求解的模型通常与用户提供的模型大不相同。两者之间的主要差异源于预求解 [1] 和缩放 [6, 15]。

预求解是一个识别变量和约束在任何解中都成立的性质，并利用这些性质创建规模更小、更易求解的等价模型的过程。例如，如果可以证明在最优解中 $x_1 = 1$ 或 $x_1 = x_2$，那么 x_1 就可以在模型传递给求解器之前从中移除。预求解是任何现代线性规划求解器的重要组成部分，通常能大幅降低求解模型的成本。

缩放是将约束矩阵中的每一行和每一列乘以一个常数，以改善数值条件性的过程。求解线性规划通常涉及对大型矩阵进行线性代数运算；更好的数值条件性可减少这些计算中的误差，通常也能提升速度和稳健性。缩放对内点法很有帮助，对单纯形法很重要，对 PDHG 则必不可少。

我们提及这些问题变换，是为了指出，线性规划求解器找到的解中的约束违反，与返回给用户的解中的约束违反并不相同。理论上，这些变换可能会使误差变小或变大，但遗憾的是，它们几乎总会使误差变大，而且往往大幅增加。因此，求解器需要生成精确解，不完全是由用户的需求所驱动；解转换回用户模型时精度也可能大幅下降，这同样是原因之一。

3 内点法热启动

我们希望通过混合 PDHG 与内点法，发挥两者各自的优势：让 PDHG 快速计算出一个低精度解，并

以此构造一个接近最优的起始点，供内点求解器使用，使其能够在接近最优解时发挥二次收敛的优势。

表面上，这种方法的步骤很直接：对模型执行 PDHG 迭代，直至达到（较宽松的）目标容差；从该解出发启动内点求解器，然后运行内点求解器直至收敛。

这里的难点在于，内点求解器不能从任意点进行热启动。收敛在很大程度上取决于起始点居中得有多好。实际上，内点求解器会求解一系列优化问题，每个问题的目标都是实现原始可行性、对偶可行性，以及 $x_j \cdot z_j = \mu$ 对所有 j 均成立（对于某个μ）[16]。算法在推进过程中不断减小 $\mu \to 0$ ，使互补项$x^\top z$ 趋近于 0，从而推动解趋近于最优点（假设原始可行性和对偶可行性也已实现）。在实际实现中，每次方法迭代后都会减小 μ ，因此与固定 μ相关的问题永远不会被完全求解。不过，内点法的收敛性证明依赖于步长保证，而只有当迭代点始终足够接近满足 $x_j \cdot z_j = \mu$ 对所有 j 均成立的点时，这些保证才成立（这类点的集合称为中心路径）。

遗憾的是，任意起始点不太可能同时满足原始可行性、对偶可行性以及这一中心化条件。例如，起始点可能是某个相关模型的最优解，而该模型经过了修改（例如，在求解混合整数规划问题时添加了变量界或割平面 [8]），因此该解不再可行。或者，解中的变量可能恰好处于其界上，从而满足 $x_j \cdot z_j = 0$ （这在 PDHG 解中很常见）。即使这两种情况都不成立，对于某个 μ而言，具有 $x_j \cdot z_j \approx \mu$ 也是一个相当深奥的条件，任意起始向量不太可能满足该条件。Yildirim 和 Wright [17]阐述了起始向量产生有利结果所需满足的条件，而这些条件相当严格。

尽管存在这些挑战，在实际实现中，谨慎构造合适的起始点已经显示出一定潜力。Gondzio [8] 在一个割平面应用中实现了个位数的迭代次数。John 和 Yildirim [9]发现迭代次数有所减少，但由于计算有效起始点的成本，这些改进通常并未转化为运行时间的缩短。Engau、Anjos 和 Vannelli 在相当广泛的测试集上将迭代次数减少了 50%。不过，据我们所知，没有广泛使用的内点线性规划求解器实现热启动，因此迄今开发的技术并未对求解线性规划问题的实践产生重大影响。

3.1 我们的热启动方法

尽管文献中介绍了多种精巧（有时成本高昂）的方法来计算有效的热启动向量，但正如我们先前所指出的，我们认为，试图解决的问题比以往研究的问题更简单：以往的方法专门针对底层问题发生显著变化时的重新优化，而我们则是从未修改的问题的近似最优解出发。

我们尝试了几种不同的方案，用于根据 PDHG 解构造热启动向量，最终确定了一种简单且成本低廉的方法，如算法 1 所述。我们测试了输入参数的多个取值，发现将最小互补性目标 $\mu_{\min}$ 设为 10^{-6}、最小变量值 $\alpha_{\min}$

算法 1 居中起始点

输入： 缩放线性规划的 PDHG 解 (x, y, z)，$\mu_{\min}$, $\delta_{\max}$, $\alpha_{\min}$
$\mu_{\text{target}} \leftarrow$ 最大值$(x^\top z/n, \mu_{\min})$
对所有 $j \in \{1..n\}$ **执行**
    $x'_j \leftarrow \max(x_j, \ \alpha_{\min})$, $z'_j \leftarrow \max(z_j, \ \alpha_{\min})$
    如果 $x'_j < z'_j$ **then**
        $x'_j \leftarrow \min(\max(\mu_{\text{target}}/z'_j, \ x'_j - \delta_{\max}) \ \ x'_j + \delta_{\max})$
        $z'_j \leftarrow \min(\max(\mu_{\text{target}}/x'_j, \ z'_j - \delta_{\max}) \ \ z'_j + \delta_{\max})$
    else
        $z'_j \leftarrow \min(\max(\mu_{\text{target}}/x'_j, \ \ z'_j - \delta_{\max}) \ \ z'_j + \delta_{\max})$
        $x'_j \leftarrow \min(\max(\mu_{\text{target}}/z'_j, \ x'_j - \delta_{\max}) \ \ x'_j + \delta_{\max})$
    结束 if
结束 for
从 $(x', \ y, \ z')$ 热启动内点求解器。
当内点步长非常小时 **执行**
    $\alpha_{\min} \leftarrow 10 \cdot \ \alpha_{\min}$
    对所有 $j \in \{1..n\}$ 执行
        $x'_j \leftarrow \max(x'_j, \ \alpha_{\min})$, $z'_j \leftarrow \max(z'_j, \ \alpha_{\min})$
    结束循环
当.....时结束循环

到 10^{-6}，且 t 最大变量扰动 $\delta_{\max}$ 设为 10^{-4} 时，效果最佳。

这种方法的细节确实很重要。我们发现，其中最重要的一点是使用缩放模型。缩放的一个目标是构造一个约束矩阵A ，使每一行和每一列的范数都大致为 1。这一性质的一个结果是，对变量施加恒定扰动，会使原始和对偶不可行度（按无穷范数计）恒定增加。不可行度大幅增加会使快速终止变得困难。

另一个重要细节是，需要采用一种策略，避免在第一次内点迭代时陷入停滞。我们希望起始点尽可能接近 PDHG 解，以加快收敛；但又不能过于接近边界，否则求解器将无法迈出较大的步子。我们发现，逐渐增大 $\alpha_{\min}$ ，直到内点求解器能够迈出一个非平凡的步子，在这两个目标之间取得了很好的平衡。在我们的测试中，将最小步长设为 10^{-6} 效果不错。

我们尝试对这一中心化方案进行的一项修改起初似乎很有前景，但最终未能成功。扰动结构变量后，我们尝试重新计算原始松弛量和对偶松弛量。随后对这些派生值进行扰动，以改善中心化。虽然这种方法确实能使残差更小，但收敛情况却明显变差。

4 实验结果

我们有多种求解线性规划问题的方法，每种方法在运行时间和精度之间都有各自的权衡。下面将展示其中几种方法的结果。

在此之前，我们先介绍测试环境，包括所用软件、机器以及线性规划测试集的详细信息。

4.1 测试环境

我们所有的测试均使用 Gurobi 13.0 版进行，并在此基础上扩展实现了本文所述的方法。该版本 Gurobi 中的 PDHG 实现与 cuPDLP+ [11]采用的方法非常接近。Gurobi 还包含先进的内点算法和交叉算法实现，这些算法经过多年测试与调优，适用于各种实际线性规划模型。

我们的大多数测试是在 Nvidia Grace Hopper GH200 系统上进行的，该系统由一颗包含 72 个核心的 Nvidia Grace CPU 和一块 Hopper H100 GPU 组成。在我们的测试中，PDHG 在图形处理器上运行，其他所有任务均在中央处理器上运行。该系统配备 512GB 共享内存，图形处理器则拥有独立的 96GB HBM 内存。对我们而言，这是一种独特的计算资源，也限制了我们在广泛的大型模型上开展大规模图形处理器测试的能力。

为部分应对这一限制，我们的一些测试是在由基于 AMD EPYC 4364P 的系统组成的大型集群上进行的，每台系统配备一颗 8 核处理器和 128 GB 内存。作为内存带宽有限的纯中央处理器系统，这些机器运行 PDHG 的性能相当差。我们在此使用它们，仅仅是因为大型集群让我们能够在规模大得多的测试集上运行测试。

4.2 测试模型

本文使用的主要测试集是 Mittelmann LP集 [14], ，其中包含 43 个公开可用、规模和难度适中的模型。这些模型的平均求解时间仅为几秒。该测试集上的所有测试均在我们的图形处理器系统上运行（PDHG 在图形处理器上运行；其他所有任务均在中央处理器上运行）。

我们还考虑了一个规模更大的测试集，其中包含 2298 个线性规划模型，涵盖了公开可用模型和客户模型。该测试集仅在我们基于中央处理器的大型集群上运行。

所有测试的时间限制均为 10,000 秒。所有运行时间结果均按从将用户模型提交给求解器到返回最优解所经过的时间计算。在展示运行时间和解的精度数据时，我们有时会提供测试集中每个模型的单独数值，有时则会对整个测试集进行汇总。汇总采用几何平均数。

4.3 违反度指标

我们在测试中采用的另一种汇总方式，是将原始可行性、对偶可行性和互补性的违反程度合并为一个指标。具体来说，我们使用的三个量分别是最大原始不可行度（以 $\|r_P\|_\infty$ 衡量）、最大对偶不可行度（以 $\|r_D\|_\infty$ 衡量），以及目标间隙（以 $x^\top z/(1+|c^\top x|)$ 衡量）。这些量都是在原始模型上计算的，并且已消除预求解和缩放的影响。我们取这三个量中的最大值，并将其称为该解相对于该模型的最大违反 。

我们为什么选择这种违反度量？无穷范数是线性规划中衡量原始和对偶不可行度的传统指标。这也符合我们的经验：用户通常会关注解对任一约束的违反程度。至于目标间隙，我们的经验是，用户通常会关注结果中有多少位数字一致。

还应补充一点：我们使用其他度量指标进行了相同的比较。虽然违反量的绝对值有所变化，但各方法之间的相对结果在定性上相似。

4.4 简单变体的性能-精度权衡

我们先比较三种易于获取的方案的运行时间和最大违反量，以改善基线 PDHG 运行所提供的解的精度（其中$\varepsilon_{\text{rel}} = 10^{-4}$）。前两种方案只需收紧容差；在我们的案例中，考虑的是$\varepsilon_{\text{rel}} = 10^{-6}$ 和 $\varepsilon_{\text{rel}} = 10^{-8}$。第三种方案则是不使用交叉，改用内点法。

图1展示了在 Mittelmann 线性规划测试集上进行这项测试的结果（测试在我们的 Grace Hopper 系统上运行）。每个点代表某种方法在某个模型上的一次运行结果。横轴表示该模型的运行时间，相对于四种方法在同一模型上达到的最佳运行时间。纵轴表示该模型的最大约束违反量（定义如前所述）。

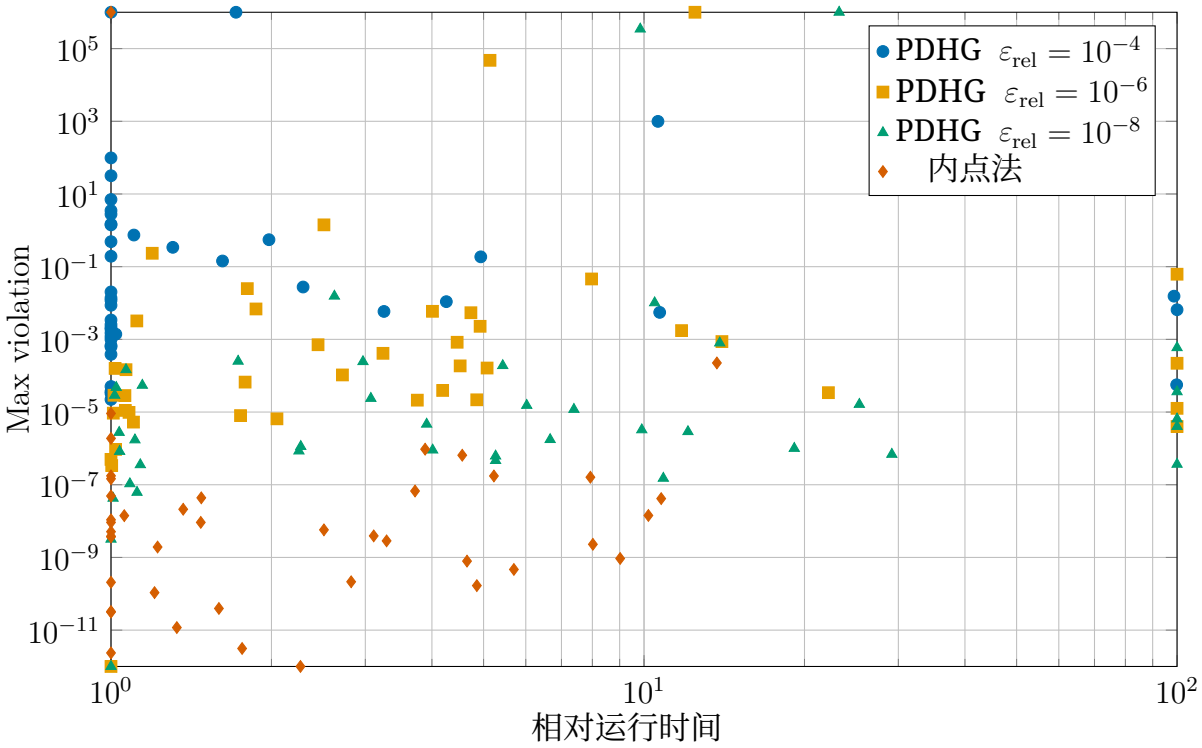

图1：四种求解方法处理 Mittelmann 测试集中的模型时的运行时间和最大违反量。大于 100 的运行时间比记为 100。同样，小于 10^{-12} 或大于 10^{6} 的最大违反量分别记为这些值。

从图中的结果可以清楚地看出，收紧 PDHG 收敛容差的效果有限；虽然约束违反量有所降低，但代价相当高。即使容差非常严格，约束违反量仍远高于内点法求解器的结果。请注意，内点法局部二次收敛，因此迭代点一旦接近最优解，之后的每次迭代都会显著提高解的精度。

表1展示了这项相同测试的汇总结果。第一行显示了

	PDHG $\varepsilon_{\text{rel}} = 10^{-4}$	**PDHG** $\varepsilon_{\text{rel}} = 10^{-6}$	**PDHG** $\varepsilon_{\text{rel}} = 10^{-8}$	**内点法**
求解的模型数	43	42	42	42
相对运行时间	1.00	2.29	3.56	1.16
平均最大违反量	$5 \cdot 10^{-2}$	$\mathbf{3} \cdot 10^{-4}$	$\mathbf{1} \cdot 10^{-5}$	$\mathbf{1} \cdot 10^{-8}$

表1：运行时间和最大违反的比较，Mittelmann测试：采用不同（相对）收敛容差的PDHG，以及不进行交叉的内点法（使用Mittelmann LP集中的43个模型的几何平均数计算）。

每种方法能够求解的模型。只有采用 $\varepsilon_{\text{rel}} = 10^{-4}$ 的PDHG能够以所需容差求解全部43个模型。其他三个选项无法在一个数值上具有挑战性的模型（*ns1688926*）上达到目标容差。

下一行显示总运行时间的几何平均数，并以基准选项（采用 $\varepsilon_{\text{rel}} = 10^{-4}$的PDHG）的运行时间为基准。第二行显示每种方法的最大约束违反的几何平均数。

从这些结果中可以得出几个明确的结论：(i) 随着容差收紧，PDHG 的性能会显著下降；(ii) 内点求解器产生的违约量比 PDHG 小 3 到 6 个数量级。虽然很难回答究竟需要多高的精度，但如此巨大的差异表明，其中一些方法对某些应用而言无法提供足够的精度。

4.5 混合方法的结果

现在来考察混合方法的性能。为了将结果置于更广泛的背景下，我们考虑几种额外方案，以获得比基准 PDHG（使用 $\varepsilon_{\text{rel}} = 10^{-4}$）更精确的解：

- 收紧 PDHG 容差；改用 $\varepsilon_{\text{相对}} = 10^{-8}$ 。上一节的比较中也考察了这一方案。

- 从 PDHG 解开始执行交叉步骤。交叉的运行时间可能在很大程度上取决于初始解的精度；我们发现，$\varepsilon_{\text{相对}} = 10^{-6}$ 能带来总体上最快的运行时间。

- 采用我们刚刚描述的策略，从 PDHG 解热启动内点求解器。我们发现，从满足 $\varepsilon_{\text{相对}} = 10^{-4}$ 的 PDHG 解出发可获得最佳结果。

	PDHG $\varepsilon_{\mathbf{rel}} = 10^{-8}$	**PDHG** $\varepsilon_{\mathrm{rel}} = 10^{-6}$ **+ 交叉**	**PDHG** $\varepsilon_{\mathrm{rel}} = 10^{-4}$ **+IP 热启动 内点法**	
求解的模型数	42	42	35	42
相对运行时间	3.56	2.84	2.17	1.16
平均最大违反量	$1\cdot\ 10^{-5}$	$\mathbf{1}\cdot 10^{-10}$	$\mathbf{2}\cdot 10^{-8}$	$\mathbf{1}\cdot 10^{-8}$

表 2：运行时间与最大违反的比较：采用更严格（相对）收敛容差的 PDHG、采用交叉的 PDHG、我们的混合方法，以及内点法。所有数据均为 Mittelmann 线性规划测试集中 43 个模型的几何平均值。

这些选项在我们的 Grace Hopper 系统上针对 Mittelmann 线性规划测试集运行的结果汇总于表 2。我们还纳入了冷启动内点求解器的结果。第一行显示测试集 43 个模型中，用每种方法求解到所要求容差的模型数量。如前所述，大多数方法在求解模型 *ns1688926* 时遇到困难。我们提出的混合方法遇到的困难略多一些，求解了该集合中 43 个模型中的 35 个。

如前所述，我们的热启动方法主要的失败模式是陷入停滞，无法从当前迭代点迈出实质性的一步。我们调整起始点的方法虽有很大帮助，但并未彻底解决这一问题。

下一行显示了各种方法相对于基线 PDHG 运行（未显示）的几何平均运行时间，仅针对各方法能够求解的模型。基于 PDHG 的方法比基线方法耗时得多。有趣的是，尽管交叉步骤是一项高度串行的操作，仅在中央处理器的一个核心上运行，但执行交叉步骤仍比收紧 PDHG 容差更快。我们提出的混合方法是所有替代方法中最快的，不过其稳健性不如其他方法。

最后一行显示了最大违反量的几何平均数。这是我们首次纳入交叉的测试，精度结果明显优于我们考虑过的其他所有选项。这是意料之中的——众所周知，单纯形法和交叉生成的解极其精确。还请注意，我们的 PDHG/内点混合方法也显著提高了精度，这同样符合预期。

从图表中可以明显看出，冷启动内点求解器在这组测试中总体表现最佳。我们稍后会再讨论这个话题。

我们想再次强调，不同应用对精度的要求各不相同，精度更高并不一定更好。我们只是想指出，不同方法所能达到的解质量可能存在巨大差异，并可能对后续产生重大影响。

图 2 展示了相同的数据，但细化到了单个模型的层面。同样，每个点代表一种方法在一个模型上的一次运行。横轴表示相对于所有选项中最佳运行时间的运行时间（基线 PDHG 运行也纳入了比较，但未显示在图中）。纵轴表示该模型的最大违反量。

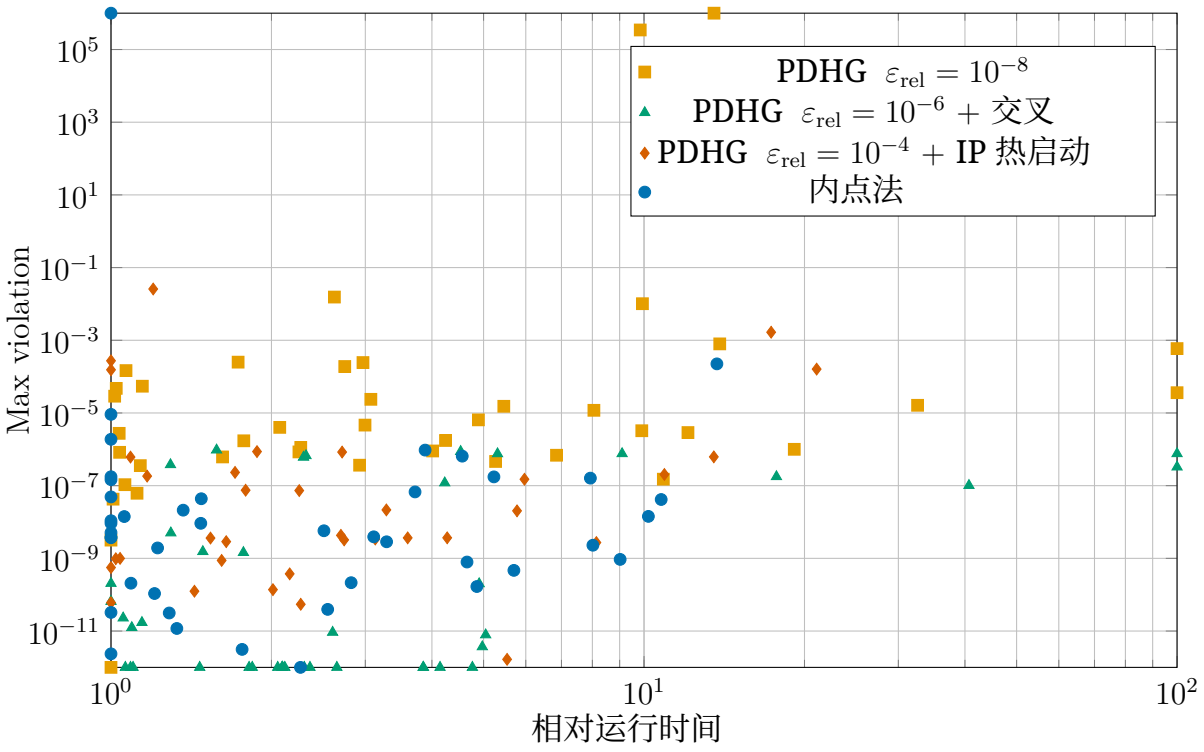

图 2：四种求解方法下，Mittelmann 测试集中各模型的运行时间与最大违反量。大于 100 的运行时间比记为 100。同样，小于 10^{-12} 或大于 10^{6} 的最大违反量记为相应数值。

	尝试的模型数	求解的模型数	成功率	IP 迭代次数比率
$\varepsilon_{\text{rel}} = 10^{-4}$	43	35	81%	0.51
$\varepsilon_{\text{rel}} = 10^{-6}$	42	37	88%	0.30

表 3：在 Mittelmann 线性规划测试集上，不同 PDHG 起始点的 IP 热启动稳健性（在 Grace Hopper 系统上运行）。

	尝试的模型数	求解的模型数	成功率	IP 迭代次数比率
$\varepsilon_{\text{rel}} = 10^{-4}$	2066	1725	83%	0.55
$\varepsilon_{\text{rel}} = 10^{-6}$	1984	1726	87%	0.35

表 4：在更广泛的测试集上，从不同 PDHG 起始点进行 IP 热启动的稳健性（在 AMD CPU 集群上运行）。

4.6 对迭代次数和稳健性的影响

现在，我们扩大测试集，以便更好地了解 PDHG/内点混合方法的稳健性，以及与冷启动相比，在更广泛的场景下它预计能在多大程度上降低内点求解的成本。

表 3 和表 4 汇总了我们两个测试集的结果。我们使用 PDHG 计算一个达到第一列所示容差的初始点；该解用于热启动内点求解器。第二列显示测试集中调用内点求解器的实例数量（如果预求解已解决问题，或 PDHG 无法在时间限制内找到满足指定容差的解，则不会调用该求解器）。下一列显示在这些模型中，热启动的内点求解器能够找到满足其收敛准则的解的数量。第三列显示前两列数值的比率。最后一列显示混合方法的内点迭代次数与冷启动内点求解的迭代次数之比的几何平均值，该值基于两种方法都成功求解的模型计算得出。

从这些表格可以看出，更广泛测试集上的结果实际上与较小的 Mittelmann 集上的结果非常相似。混合方法中热启动的内点求解器能够求解约 85% 提交给它的问题。使用 $\varepsilon_{\text{rel}} = 10^{-4}$ 时，内点迭代次数几乎减少到原来的一半；使用 $\varepsilon_{\text{rel}} = 10^{-6}$ 时，几乎减少到原来的三分之一。

表 5 为这些结果增添了一些细节。前两行重复了前面表格中的数据：它们显示了提交给内点求解器的模型数量，以及求解器成功求解的模型数量。第三行是新增的：它显示了内点求解器能够在 10 次或更少迭代内达到目标容差的模型数量。我们认为，这一门槛有力地表明，初始点对该模型的收敛有相当大的帮助。热启动的内点求解器在约一半成功求解的模型上，能在 10 次或更少迭代内收敛；相比之下，冷启动求解器的这一比例不到 17%。

尽管这里展示的结果有望证明我们提出的混合方法前景可期，但它们也引出了一个更广泛的问题：如何公平地评估 PDHG 的性

	内点法 **冷启动**	**PDHG** $\varepsilon_{\text{rel}} = 10^{-4}$ **+IP 热启动**	**PDHG** $\varepsilon_{\text{rel}} = 10^{-6}$ **+IP 热启动**
尝试的模型数	2179	2066	1984
求解的模型数	2090	1725	1726
在 ≤ 10 次迭代内求解的数量	318	793	1029

表5：不同起始点的内点迭代次数。

能。我们的方法在表 2 所考察的、用于提高 PDHG 解精度的各种方法之间的性能—精度权衡中，占据了有效前沿上的一个位置；不过，冷启动内点法取得了比这些方法都更好的结果。我们的经验是，尽管偶尔能取得显著胜利，但在广泛的测试集上进行评估时，即使收敛容差非常宽松，PDHG 相较于现有替代方案也并未表现出显著优势。

如果PDHG目前的优势只是个别案例，那么我们或许也应为我们提出的方法寻找个别案例上的优势。在Mittelmann 测试集上将冷启动内点求解器与我们的PDHG混合方法进行比较时，尽管后者在整个测试集上明显落后，但我们发现后者在6个案例中快了10%以上（而前者在16个案例中快了10%以上）。

5 讨论

本文提出的混合方法并非将PDHG与内点法相结合的唯一选择。我们还考虑使用简略的内点求解来为PDHG计算起始点。具体而言，我们使用迭代线性求解器在内点求解器的早期迭代中计算方向向量（这既快速又对图形处理器友好），并在迭代求解器的迭代总次数达到固定值后终止（我们的测试中为1,000或10,000次）。该方法为PDHG提供了一个残差更小、中心化程度优于通常所用平凡起始点的起始点。应当指出，尽管中心化对内点法至关重要，但据我们所知，它在PDHG的收敛论证中并不起作用。指望获得实际收益或许只是一厢情愿，而我们的实验也确实没有发现一致的优势。

在我们的测试中，我们只将图形处理器用于 PDHG。近期研究 [12]也探讨了将图形处理器用于内点迭代。虽然目前图形处理器相较于中央处理器的优势似乎有限，但图形处理器的发展日新月异，因此未来的结果可能会大不相同。目前，我们只需指出，我们的混合方法能够让你使用几乎完全在图形处理器上运行的方法，获得精度很高的解。

6 结论

本文提出了一种混合方法，将用于计算低精度解的 PDHG 步骤与用于显著提高该解精度的热启动内点步骤相结合。尽管热启动内点法是一个出了名的难题

问题，我们发现在此背景下，一种相对简单的方法实现了快速且可靠的收敛。在一个广泛的测试集中，内点求解器对超过一半的模型只需不到 10 次迭代即可收敛。

一个一直不太容易回答的问题是，如何评估我们的方法适用范围有多广。要让某个特定模型受益，以下条件必须成立：(i) PDHG 相较于其他方法具有显著的性能优势；(ii) 使用该解的应用需要比 PDHG 在可用时间内所能提供的精度更高；(iii) 执行几次内点迭代的代价并不高。我们提供的数据表明，在这类情形下，所提出的方法能带来显著优势。至于这些情形出现的频率，目前仍是一个悬而未决的问题。

致谢

我们要感谢 Haihao Lu 和 David Torres Sanchez 提供的宝贵意见。

参考文献

[1] Erling D. Andersen 和 Knud D. Andersen。线性规划中的预处理。*Mathematical Programming*，71:221–245，1996。[2]David Applegate、Mateo D´ıaz、Oliver Hinder、Haihao Lu、Miles Lubin、Brendan O'Donoghue 和 Warren Schudy。使用原始-对偶混合梯度求解实际大规模线性规划。神经信息处理系统进展，34 (2021):20243—-20257，2022。[3] Antonin Chambolle 和 Thomas Pock。一种用于凸问题的一阶原始-对偶算法及其在成像中的应用。*Journal of Mathematical Imaging and Vision*，40:120–145，2011。[4] Kaihuang Chen、Defeng Sun、Yancheng Yuan、Guojun Zhang 和 Xinyuan Zhao。HPR-LP：一种用于求解线性规划的 HPR 方法实现。*arXiv*预印本 *arXiv:2408.12179*，2024。URL `https://arxiv.org/abs/2408.12179`。[5] George B. Dantzig。单纯形法在运输问题中的应用。载于*Activity Analysis of Production and Allocation*，第359–373页，1951。[6] Joseph M. Elble 和 Nikolaos V. Sahinidis。在应用单纯形法之前对线性优化问题进行缩放。*Computation Optimization and Applications*，52 (2):345–371，2012。[7] Alexander Engau、Miguel F. Anjos 和 Anthony Vannelli。关于线性优化与组合优化的内点法热启动。*SIAM Journal on Optimization*，20(4):1828–1856，2010。

[8] Jacek Gondzio。在割平面方案中应用原始-对偶方法的热启动。*Mathematical Programming*，83:125–143，1998。[9] Elizabeth John 和 E. Alper Yildirim。固定维数线性规划中内点法热启动策略的实现。*Computationa Opti-mization and Applications*，41(2):151–183，2008。[10] Haihao Lu 和 Jinwen Yang。cuPDLP.jl：在 Julia 中实现用于线性规划的重启式原始-对偶混合梯度的 GPU 版本。*arXiv*预印本 *arXiv:2311.12180*，2023。URL https://arxiv.org/abs/2311.12180。[11] Haihao Lu、Zedong Peng 和 Jinwen Yang。cuPDLPx：一种进一步增强的基于GPU的一阶线性规划求解器。*arXiv*预印本 *arXiv:2507.14051*，2025。URLhttps://arxiv.org/abs/2507.14051。[12]Christopher Maes。使用 NVIDIA cuOpt 中由 GPU 加速的障碍法求解线性规划。
https://developer.nvidia.com/blog/
solve-linear-programs-using-the-gpu-accelerated-barrier-method-in-nvidia-cuopt,
2025。[13] Nimrod Megiddo。关于寻找原始最优基与对偶最优基。*ORSA Journal on Com-puting*，3(1):63–65，1991。[14] Hans Mittelmann。优化软件决策树。http://plato.asu.edu/guide.html，2025。[15] John A. Tomlin。关于线性规划问题的缩放。*Computational Practice inMathematical Programing, Mathematical Programming Studies*，4:146–166，1975。[16] Stephen Wright。原始-对偶内点法。SIAM，1997。[17] E. Alper Yildirim 和 Stephen Wright。线性规划内点法的热启动策略。*SIAM Journal on Optimization*，12(3):782–810，2002。