Solving Linear Programs with Fast Online Learning Algorithms

Wenzhi Gao [1] **Dongdong Ge** [1] **Chunlin Sun** [2] **Yinyu Ye** [3]

Abstract

This paper presents fast first-order methods for solving linear programs (LPs) approximately. We adapt online linear programming algorithms to offline LPs and obtain algorithms that avoid any matrix multiplication. We also introduce a variable-duplication technique that copies each variable K times and reduces the optimality gap and constraint violation by a factor of $\sqrt{K}$. Furthermore, we show how online algorithms can be effectively integrated into sifting, a column generation scheme for large-scale LPs. Numerical experiments demonstrate that our methods can serve as either an approximate direct solver, or an initialization subroutine for exact LP solving.

1. Introduction

First-order methods for large-scale linear programs are receiving increasing attention as problem sizes grow beyond the capacity of traditional simplex (Dantzig, 2016) and interior point methods (Ye, 2011). When a low or moderate-accuracy solution is required, first-order methods often outperform simplex and interior point solvers as they only involve matrix-vector multiplication and are free of matrix decomposition. A recent practice (Applegate et al., 2021) demonstrates that fine-tuned first-order methods are also capable of solving LPs of millions of variables to high accuracy. However, as (Nesterov, 2012) suggests, when the problem size further reaches trillions of variables or more, even storing the problem data in memory becomes prohibitive, not to mention performing matrix-vector multiplications. As a result, most first-order methods either resort to distributed architecture (Basu et al., 2020; Fountoulakis and Gondzio, 2016) or rely on sparsity (Nesterov, 2012).

In contrast to the LPs discussed above, which we call "offline" LPs, there is another type of LPs called "online" LPs (Agrawal et al., 2014; Li and Ye, 2022). They come from revenue management (Talluri et al., 2004) and model the situation of making sequential decisions on the customer orders that are described by LP columns. Here an LP column denotes a column in the constraint matrix together with its corresponding objective value. Due to online nature of the problem, generally an algorithm for online LPs **1)**. only accesses one LP column at a time; **2)**. makes decisions instantly. Through making decisions, the behavior of online algorithms gets refined and improved, which is why they are also known as "online learning".

In the line of online LP literature, state-of-the-art online LP algorithms build on offline LPs as a subroutine (Jiang et al., 2022; Li and Ye, 2022; Ma et al., 2022), and it seems that *online LP unilaterally* ***benefits from*** *offline LP*. It is therefore natural to ask:

Does online LP benefit offline LP?

In this paper, we answer this question positively by exploiting a recent research direction on fast online algorithms (Li et al., 2020; Balseiro et al., 2022). Unlike the offline LP-based online algorithms, these methods build on stochastic first-order oracles, make decisions and refinements in linear time, and simultaneously estimate primal and dual solutions in a single pass through the problem data. Also, they achieve provable optimality guarantees under minimal assumptions on the problem data. All these motivate us to study if these time-efficient methods help offline LP solving. In particular, we investigate two popular online algorithms: explicit subgradient (Li et al., 2020) and implicit proximal point (Kulis and Bartlett, 2010).

First-order Methods for Linear Programming First-order methods for LPs are recently developed and presented as optimization software. Some successful attempts include `ABIP` (Lin et al., 2021; Deng et al., 2022), `COSMO` (Garstka et al., 2021), `ECLIPSE` (Basu et al., 2020), `PDLP` (Applegate et al., 2021) and `SCS` (O'donoghue et al., 2016). The listed algorithms target large-scale LPs, and the major computational bottleneck lies in matrix-vector multiplication (sometimes one-time matrix decomposition). Com-

[1]School of Information Management and Engineering, Shanghai University of Finance and Economics [2]Institute for Computational and Mathematical Engineering, Stanford University [3]Management Science & Engineering, Stanford University. Correspondence to: Wenzhi Gao <gwz@163.shufe.edu.cn>, Dongdong Ge <ge.dongdong@mail.shufe.edu.cn>, Chunlin Sun <chunlin@stanford.edu>, Yinyu Ye <yyye@stanford.edu>.

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

pared to the aforementioned works, our work is more in line with (block) subgradient-based or coordinate descent-based approaches (Nesterov, 2012; 2014). What differentiates our method is that they (Nesterov, 2012; 2014) have to update primal and dual variables sequentially, while our proposed method simultaneously updates both solutions.

Simple and Fast Online LP A simple and fast online LP algorithm is recently proposed and analyzed by (Li et al., 2020). It exploits the finite-sum structure of the LP dual problem and runs stochastic subgradient method in the dual space, while simultaneously estimating the primal solution by optimality condition. Under mild assumptions, given an LP of m constraints and n variables, the authors show that their algorithm achieves $\mathcal{O}\left(\sqrt{n}\log n + m\sqrt{n}\right)$ constraint violation and an $\mathcal{O}(m\sqrt{n})$ optimality gap (also known as regret in online learning literature) under the random permutation setting that we exploit in this paper. Concurrent works (Balseiro et al., 2020; 2022) further generalize the results to nonlinear objective using a mirror descent framework and achieve $\mathcal{O}(m + \sqrt{mn})$ gap with no constraint violation under a weaker setting. We also note that (Balseiro et al., 2022) covers an adversarial setting that is more general than random permutation. However, the optimality gap is too pessimistic to be applied for offline LPs, and therefore we exploit the setting of (Li et al., 2020).

Contributions In this paper, we show that online LP algorithms can benefit offline LP both theoretically and practically. Specifically,

- We investigate how two online learning algorithms perform when they are used to solve offline LPs. These two methods, known as explicit subgradient and implicit proximal point in the online learning literature, are free of any matrix-vector multiplication through sequential online access to LP columns. We provide theoretical analyses of the optimality gap and constraint violation leveraging the analysis from online LP literature. For both algorithms, we obtain an $\mathcal{O}\left(m\log n + \sqrt{n}\log n + \sqrt{mn}\right)$ optimality gap and $\mathcal{O}\left(m + \sqrt{mn}\right)$ constraint violation, which improves the results from (Li et al., 2020). Moreover, to enhance the practical performance of the online algorithms and to make them usable for offline LPs, we take advantage of the offline LP setting and propose a variable duplication scheme which further reduces the gap and violation by a factor of $\sqrt{K}$, where K is the number of copies of each variable.
- For exact LP solving, we identify that our algorithms can perfectly fit into sifting (Bixby et al., 1992), an LP column generation framework targeting problems with far more columns than constraints. We show online algorithms provide both an initial basis guess and an approximate dual solution. Numerical experiments show that our online algorithms can accelerate sifting by providing a good initialization and by stabilizing the dual solution.

Other Related Work Disregarding the complexity of solving subproblems, offline LP-based algorithms achieve state-of-the-art theoretical guarantees for online LPs. For example, (Jiang et al., 2022; Ma et al., 2022; Bray, 2019) all achieve $\mathcal{O}(\log n)$ optimality gap (up to a $\log\log$ term) under different LP non-degeneracy assumptions.

Going beyond the online LP problem, which is classified as online convex optimization with constraint (`OCOwC`), there is substantial research on general online convex optimization (Hazan et al., 2016). While most online algorithms are subgradient-based, there is also another line of research based on the implicit proximal point update (Kulis and Bartlett, 2010; Chen et al., 2022). In this paper, we choose the implicit update as one of the fast algorithms.

Outline of the Paper This paper is organized as follows. In **Section 2**, we introduce the basic problem setup and assumptions; In **Section 3**, we provide theoretical analyses of the performance of our algorithms; **Section 4** briefly discusses the sifting procedure for large-scale LPs and how online algorithms can be potentially combined with this framework. Last we conduct numerical experiments in **Section 5** to verify our theory and to show the practical efficiency of our proposed methods. In the appendix, we further elaborate on some practical aspects of our methods.

2. Problem Setup and Assumptions

Notations Throughout the paper, we use $\|\cdot\|$ to denote the Euclidean norm and $\langle\cdot,\cdot\rangle$ to denote the Euclidean inner product. Bold letters $\mathbf{A}, \mathbf{a}$ are used to denote matrices and vectors; the subdifferential of a convex function f is denoted by $\partial f(x)$ and $f'(x) \in \partial f(x)$ is called a subgradient; we use $[\cdot]_+ := \max\{\cdot, 0\}$ to denote the element-wise positive part function; $\mathbb{I}\{\cdot\}$ denotes the 0-1 indicator function and δ_S is the indicator function of set S. Unless specified, we use $\mathbb{E}[\cdot]$ to denote expectation taken over a certain permutation of LP columns. We use $\mathbf{a}_{[i:j]}$ to denote indexing of vector $\mathbf{a}$ from the i-th to j-th coordinate, or indexing of a matrix from the i-th to j-th column.

2.1. Linear Programming and Duality

Given a linear program of m constraints and n variables

$$\max_{\mathbf{x}\in\mathcal{F}_p} \quad \langle \mathbf{c}, \mathbf{x}\rangle,$$

its dual problem is given by

$$\min_{(\mathbf{y},\mathbf{s})\in\mathcal{F}_d} \quad \langle \mathbf{b}, \mathbf{y}\rangle + \langle \mathbf{1}, \mathbf{s}\rangle$$

Table 1. Previous optimality gap and constraint violation bounds. Ex: explicit update; Im: implicit proximal point update; Duplicate: **Algorithm 2**; ρ: optimality gap; v: constraint violation. We note that the Duplicate setting is only available for offline LPs.

Work	Algorithm	Stochastic input (v & ρ)	Random permutation (v & ρ)
(Li et al., 2020)	Ex. Subgrad	$\mathcal{O}(m + m\sqrt{n})$ & $\mathcal{O}(m\sqrt{n})$	$\mathcal{O}(m\sqrt{n})$ & $\mathcal{O}(\sqrt{n}\log n + m\sqrt{n})$
(Balseiro et al., 2022)	Ex. Mirror	$\mathcal{O}(m + \sqrt{mn})$ & 0	—
This work	Ex. Subgrad	$\mathcal{O}(m + \sqrt{mn})$ & $\mathcal{O}(\sqrt{mn})$	$\mathcal{O}(m + \sqrt{mn})$ & $\mathcal{O}(m\log n + \sqrt{n}\log n + \sqrt{mn})$
	Im. Proximal	$\mathcal{O}(m + \sqrt{mn})$ & $\mathcal{O}(\sqrt{mn})$	$\mathcal{O}(m + \sqrt{mn})$ & $\mathcal{O}(m\log n + \sqrt{n}\log n + \sqrt{mn})$
	Ex/Im. Duplicate	$\mathcal{O}(\frac{m}{K} + \sqrt{\frac{mn}{K}})$ & $\mathcal{O}(\sqrt{\frac{mn}{K}})$	$\mathcal{O}(\frac{m}{K} + \sqrt{\frac{mn}{K}})$ & $\mathcal{O}(\frac{m\log n}{K} + \sqrt{\frac{n}{K}}\log n + \sqrt{\frac{mn}{K}})$

where the primal and dual feasible sets are denoted by

$$\mathcal{F}_p := \{\mathbf{x} : \mathbf{A}\mathbf{x} \le \mathbf{b}, \mathbf{0} \le \mathbf{x} \le \mathbf{1}\}$$
$$\mathcal{F}_d := \left\{(\mathbf{y}, \mathbf{s}) \ge \mathbf{0} : \mathbf{A}^\top \mathbf{y} + \mathbf{s} \ge \mathbf{c}, \mathbf{s} \ge \mathbf{0}\right\},$$

with $\mathbf{A} \in \mathbb{R}^{m\times n}, \mathbf{b} \in \mathbb{R}^m, \mathbf{c} \in \mathbb{R}^n$ and $\mathbf{1} = (1, \dots, 1)^\top$ denotes the column vector of all ones. By the same argument as in (Li et al., 2020), we remove the dual variable $\mathbf{s}$ and rewrite the dual problem with a finite-sum formulation.

$$\min_{\mathbf{y}\ge\mathbf{0}} \frac{1}{n}\sum_{j=1}^{n} \langle \mathbf{d}, \mathbf{y}\rangle + [c_j - \langle \mathbf{a}_j, \mathbf{y}\rangle]_+ =: \frac{1}{n}\sum_{j=1}^{n} f(\mathbf{y}, j) \quad (1)$$

where $\mathbf{d} = \mathbf{b}/n$ and $\mathbf{a}_j$ denotes the j-th column of $\mathbf{A}$. Here, with a slight abuse of notation, we denote $f(\mathbf{y}, j) = \mathbf{d}, \mathbf{y}\rangle + [c_j - \langle \mathbf{a}_j, \mathbf{y}\rangle]_+$ as the stochastic function associated with the LP column $(c_j, \mathbf{a}_j)$ for all j without the explicit mention of c and $\mathbf{a}$ in the function arguments. Given the optimal dual solution $\mathbf{y}^*$, the optimality conditions tell us

$$x_j^* \in \begin{cases} \{0\}, & c_j < \langle \mathbf{a}_j, \mathbf{y}^*\rangle \\ [0,1], & c_j = \langle \mathbf{a}_j, \mathbf{y}^*\rangle \\ \{1\}, & c_j > \langle \mathbf{a}_j, \mathbf{y}^*\rangle \end{cases} \quad (2)$$

Taking into account this finite-sum structure of the dual problem, together with the close relevance between primal and dual solutions, it is appealing to apply first-order stochastic algorithms to the dual problem and simultaneously estimate primal solutions using the relation (2). This is indeed what "simple and fast online algorithms" does.

2.2. Simple and Fast Online Algorithms

Now, we are ready to introduce online algorithms of our interest in **Algorithm 1**.

In each iteration, let $\mathbf{y}^k$ be the current estimation of the dual solution. We choose $i_k \in [n]$, approxiamte $f(\mathbf{y}, i_k)$ by some function $f_{\mathbf{y}^k}(\mathbf{y}, i_k)$, which will be defined in **Section 3**, and perform stochastic proximal updates on the dual variable $\mathbf{y}$. At the end of each iteration, we use the current dual information, combined with (2) to estimate the primal solution $\hat{x}^{i_k}$. Since in each iteration, only one LP column participates in the update, the cost of each iteration is very low, thereby making the methods "simple and fast".

Algorithm 1 Fast online algorithms for LP

Input: $\mathbf{y}^0, \{f(\mathbf{y}, j)\}$ from LP data $\mathbf{A}, \mathbf{b}, \mathbf{c}$
for k = 1 to n **do**
  Choose $i_k \in [n]$

$$\mathbf{y}^{k+1} = \arg\min_{\mathbf{y}\ge\mathbf{0}} \{f_{\mathbf{y}^k}(\mathbf{y}, k) + \tfrac{1}{2\gamma_k}\|\mathbf{y} - \mathbf{y}^k\|^2\} \quad (3)$$

  Estimate $\hat{x}^{i_k}$ based on (3).
end
Output: $\hat{\mathbf{x}}$

2.3. Assumptions

We make the following assumptions across the paper.

- **A1:** $\max_i d_i = \bar{d} \ge \min_i d_i = \underline{d} > 0 \quad (\mathbf{d} = \mathbf{b}/n)$
- **A2:** $\|\mathbf{a}_j\|_\infty \le \bar{a}$ and $|c_j| \le \bar{c}$ for any $j \in [n]$
- **A3:** For any dual solution $\mathbf{y} \ge \mathbf{0}$, no more than m columns satisfy $c_j = \langle \mathbf{a}_j, \mathbf{y}\rangle$

Remark 1. The above assumptions are standard in the literature of online LP (Li et al., 2020; Balseiro et al., 2022; Li and Ye, 2022), and can also be easily satisfied by most offline LPs. **A1** is satisfied by a wide range of LPs such as multi-knapsack, set-covering, and online resource allocation. Even if $b_i = 0$, we can perturb it to make the assumption hold; Bounds in **A2** are only used for analysis and $\bar{a}, \bar{c}$ can be computed by a single pass through data. **A3** could be satisfied by an arbitrarily small perturbation of $\mathbf{b}$ (Megiddo and Chandrasekaran, 1989; Agrawal et al., 2014).

2.4. Performance Measure

We use optimality gap and constraint violation to measure the quality of a given primal solution $\hat{\mathbf{x}}$.

$$\rho(\hat{\mathbf{x}}) := \max_{\mathbf{x}\in\mathcal{F}_p} \langle \mathbf{c}, \mathbf{x}\rangle - \langle \mathbf{c}, \hat{\mathbf{x}}\rangle \quad (4)$$

$$v(\hat{\mathbf{x}}) := \|[\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}]_+\| \quad (5)$$

With all the tools in hand, we are ready to present several realizations of **Algorithm 1** and to analyze their performance for offline LPs.

3. Fast Online Algorithms for Offline LPs

In this section, we provide two realizations of **Algorithm 1** and analyze their expected optimality gap and constraint violation. We start by specifying the choice of i_k in **Algorithm 1**. Although stochastic input, namely sampling i_k from $[n]$ randomly with replacement is a feasible option, a problem is that, with this method, some LP columns $(c_j, \mathbf{a}_j)$ might not be traversed within $\mathcal{O}(n)$ iterations, and therefore we cannot get a solution estimate for those columns. Therefore, random permutation, or choosing i_k by sampling without replacement from $[n]$, is a better choice for offline LP's context. In other words, we can safely stop after n iterations and ensure that all the columns are associated with their estimated primal values.

To avoid the heavy notations from permutation, without loss of generality we assume instead of sampling i_k from $[n]$, we permute the columns of the offline LP, so that we can simply let $i_k = k$ and get the same theoretical results.

Remark 2. As is shown in (Li et al., 2020), once we prove the result for the random permutation setting, the analysis can be directly applied to the stochastic input setting with a slightly better bound for the optimality gap.

3.1. Online Explicit Update

In this section, we analyze the performance of the online explicit subgradient update in the offline setting. The (Sub)gadient-based update approximates $f(\mathbf{y}, k)$ by a linear function

$$\begin{aligned} f_{\mathbf{y}^k}(\mathbf{y}, k) &= \langle f'(\mathbf{y}^k, k), \mathbf{y} - \mathbf{y}^k \rangle \\ &= \langle \mathbf{d} - \mathbf{a}_k \mathbb{I}\{c_k > \langle \mathbf{a}_k, \mathbf{y}^k \rangle\}, \mathbf{y} - \mathbf{y}^k \rangle, \end{aligned} \tag{6}$$

where we take $\mathbf{d} - \mathbf{a}_k \mathbb{I}\{c_k > \langle \mathbf{a}_k, \mathbf{y}^k \rangle\} \in \partial f(\mathbf{y}^k, k)$ and estimate the primal solution by $x^k = \mathbb{I}\{c_k > \langle \mathbf{a}_k, \mathbf{y}^k \rangle\}$. Specifically, the dual update (3) is given in the closed form

$$\mathbf{y}^{k+1} = [\mathbf{y}^k - \gamma(\mathbf{d} - \mathbf{a}_k x^k)]_+.$$

With this dual updating rule, our **Algorithm 1** becomes the method in (Li et al., 2020). Compared with (Li et al., 2020), we provide a sharper analysis to achieve a better trade-off between the optimality gap and constraint violation.

Lemma 1. *Under assumptions* **A1** *to* **A3**, *if we take* $\gamma_k \equiv \gamma$, *then solution* $\hat{\mathbf{x}}$ *output by* ***Algorithm 1*** *using explicit subgradient update* (6) *satisfies*

$$\begin{aligned} \mathbb{E}[\rho(\hat{\mathbf{x}})] &\le \mathcal{O}(m \log n + \sqrt{n} \log n) + \tfrac{m(\bar{a}+\bar{d})^2 n}{2} \cdot \gamma \\ \mathbb{E}[v(\hat{\mathbf{x}})] &\le \tfrac{m(\bar{a}+\bar{d})^2}{\underline{d}} + \sqrt{m}(\bar{a} + \bar{d}) + \tfrac{\bar{c}}{\underline{d}} \cdot \gamma^{-1}, \end{aligned}$$

where $\mathbb{E}[\cdot]$ *is taken over the random permutation.*

Remark 3. **Lemma 1** implies a trade-off between ρ and v with respect to the stepsize of the subgradient update. Since we know $\bar{a}, \bar{c}, \underline{d}, \bar{d}$ in the offline case, it is, therefore, possible to find an optimal γ to balance the two criteria.

Theorem 1. *Under the same conditions as* ***Lemma 1***, *if we take* $\gamma = \sqrt{\frac{2\bar{c}}{\underline{d}(\bar{a}+\bar{d})^2 mn}}$, *then* ***Algorithm 1*** *using explicit subgradient update* (6) *outputs a solution* $\hat{\mathbf{x}}$ *satisfying*

$$\begin{aligned} \mathbb{E}[\rho(\hat{\mathbf{x}})] &\le \mathcal{O}(m \log n + \sqrt{n} \log n) + \left(\tfrac{(\bar{a}+\bar{d})^2 \bar{c}}{2\underline{d}}\right)^{1/2} \sqrt{mn} \\ \mathbb{E}[v(\hat{\mathbf{x}})] &\le \tfrac{m(\bar{a}+\bar{d})^2}{\underline{d}} + \sqrt{m}(\bar{a} + \bar{d}) + \left(\tfrac{(\bar{a}+\bar{d})^2 \bar{c}}{2\underline{d}}\right)^{1/2} \sqrt{mn}, \end{aligned}$$

where $\mathbb{E}[\cdot]$ *is taken over the random permutation.*

Remark 4. We see the online algorithm gives $\mathcal{O}(m \log n + \sqrt{n} \log n + \sqrt{mn})$ gap and $\mathcal{O}(m + \sqrt{mn})$ violation even if we take γ to be a suboptimal value of $\mathcal{O}(1/\sqrt{mn})$. Compared with the $\mathcal{O}(\sqrt{n} \log n + m\sqrt{n})$ gap and $\mathcal{O}(m\sqrt{n})$ violation of (Li et al., 2020), the result is improved with respect to m. The main reason why (Li et al., 2020) gets a suboptimal result is that they choose $\gamma = 1/\sqrt{n}$, which gives an unbalanced trade-off between gap and violation.

Remark 5. The online explicit updates can be implemented in $\mathcal{O}(\text{nnz}(\mathbf{A}))$ time and is free of any matrix-vector operations. See **Section B.1** for more details.

3.2. Online Implicit Update

In this section, we analyze the performance of the online implicit update applied to offline LPs. Instead of approximating $f(\mathbf{y}, k)$, we keep all the information using

$$f_{\mathbf{y}^k}(\mathbf{y}, k) = f(\mathbf{y}, k) = \langle \mathbf{d}, \mathbf{y} \rangle + [c_k - \langle \mathbf{a}_k, \mathbf{y} \rangle]_+ \tag{7}$$

and the primal solution is estimated through proximal point

$$\begin{aligned} \min_{\mathbf{y}, s} \quad & \langle \mathbf{d}, \mathbf{y} \rangle + s + \tfrac{1}{2\gamma_k} \|\mathbf{y} - \mathbf{y}^k\|^2 \\ \text{subject to} \quad & \mathbf{y} \ge \mathbf{0}, s \ge c_k - \langle \mathbf{a}_k, \mathbf{y} \rangle, s \ge 0 \end{aligned}$$

and we let $x^k = \lambda(s \ge c_k - \langle \mathbf{a}_k, \mathbf{y} \rangle)$ be the Lagrangian multiplier of $s \ge c_k - \langle \mathbf{a}_k, \mathbf{y} \rangle$ in the optimal solution.

Remark 6. In the literature of online algorithms, the proximal point update is known as implicit update, where "implicit" comes from the optimality condition

$$\mathbf{0} \in \partial f(\mathbf{y}^{k+1}, k) + \gamma_k^{-1}(\mathbf{y}^{k+1} - \mathbf{y}^k) + \mathcal{N}_{\mathbb{R}^n_+}(\mathbf{y}^{k+1})$$

and $\mathbf{y}^{k+1}$ can be expressed implicitly as

$$\mathbf{y}^{k+1} \in \mathbf{y}^k - \gamma_k(\partial f(\mathbf{y}^{k+1}, k) + \mathcal{N}_{\mathbb{R}^n_+}(\mathbf{y}^{k+1})),$$

where $\mathcal{N}_{\mathbb{R}^n_+}(\mathbf{y}^{k+1})$ is the normal cone of the nonnegative orthant at $\mathbf{y}^{k+1}$. Unlike explicit update which linearizes $f(\mathbf{y}, k)$, implicit update preserves all the information and it is shown to be more robust to stepsize selection than subgradient (Asi and Duchi, 2019; Deng and Gao, 2021).

The analysis of implicit update is similar, and we still have a trade-off between gap and violation

Lemma 2. *Under assumptions* **A1** *to* **A3***, if we take* $\gamma_k \equiv \gamma$*, then solution* $\hat{\mathbf{x}}$ *output by* ***Algorithm 1*** *using implicit proximal point update* (7) *satisfies*

$$\mathbb{E}[\rho(\hat{\mathbf{x}})] \le \mathcal{O}(m\log n + \sqrt{n}\log n) + \tfrac{5m(\bar{a}+\bar{d})^2 n}{2}\cdot\gamma$$
$$\mathbb{E}[v(\hat{\mathbf{x}})] \le \tfrac{3m(\bar{a}+\bar{d})^2}{\underline{d}} + \sqrt{m}(\bar{a}+\bar{d}) + \tfrac{\bar{c}}{\underline{d}}\cdot\gamma^{-1},$$

where $\mathbb{E}[\cdot]$ *is taken over the random permutation.*

Choosing γ properly, we get bounds on gap and violation.

Theorem 2. *Under the same condition as* ***Lemma 2****, if we take* $\gamma = \sqrt{\frac{2\bar{c}}{5\underline{d}(\bar{a}+\bar{d})^2 mn}}$*, then* ***Algorithm 1*** *using implicit proximal point update* (7) *outputs a solution* $\hat{\mathbf{x}}$ *satisfying*

$$\mathbb{E}[\rho(\hat{\mathbf{x}})] \le \mathcal{O}(m\log n + \sqrt{n}\log n) + \left(\tfrac{5(\bar{a}+\bar{d})^2\bar{c}}{2\underline{d}}\right)^{1/2}\sqrt{mn}$$
$$\mathbb{E}[v(\hat{\mathbf{x}})] \le \tfrac{3m(\bar{a}+\bar{d})^2}{\underline{d}} + \sqrt{m}(\bar{a}+\bar{d}) + \left(\tfrac{5(\bar{a}+\bar{d})^2\bar{c}}{2\underline{d}}\right)^{1/2}\sqrt{mn},$$

where $\mathbb{E}[\cdot]$ *is taken over the random permutation.*

Remark 7. Although we do not see an improvement of bounds using implicit update, as will be shown by our numerical experiments, the implicit update often behaves better empirically. One intuitive explanation is that, unlike the explicit update which only outputs binary values, the implicit update is capable of dealing with fractional values. To illustrate this consider the following LP

$$\max_{0\le x_1,x_2\le 1} \; x_1 + x_2 \;\text{ subject to }\; x_1 + x_2 \le 0.5$$

whose optimal value is 0.5. If we do not allow constraint violation greater than 0.1, then subgradient update will never take $x = 1$ from arbitrary start. However, for the implicit update, we recover the optimal solution if $\gamma < 0.01$.

3.3. Improvement by Variable Duplication

Despite the efficiency of online algorithms, the current optimality gap and constraint violation guarantees are far from enough to solve offline LPs, even approximately. To address this issue, we take advantage of the offline setting and propose to make a trade-off between time and accuracy, more specifically, by **1)**. duplicating each column K times **2)**. running online algorithm on the augmented problem with nK variables **3)**. taking the average of primal estimates. This scheme ends up giving an intuitive $\sqrt{K}$ reduction in the final bound.

Theorem 3. *Under assumptions* **A1** *to* **A3***, if we apply* ***Algorithm 2*** *with* K *duplications and take* $\gamma = \mathcal{O}(1/\sqrt{Kmn})$*, then both explicit and implicit updates output solutions* $\hat{\mathbf{x}}$ *satisfying*

$$\mathbb{E}[\rho(\hat{\mathbf{x}})] \le \mathcal{O}\big(\tfrac{m\log n}{K} + \sqrt{\tfrac{n}{K}}\log n + \sqrt{\tfrac{mn}{K}}\big)$$
$$\mathbb{E}[v(\hat{\mathbf{x}})] \le \mathcal{O}\big(\tfrac{m}{K} + \sqrt{\tfrac{mn}{K}}\big).$$

Algorithm 2 Online algorithm with duplication

Input: $\mathbf{y}^0, K, \{f(\mathbf{y}, j)\}$ from LP data $\mathbf{A}, \mathbf{b}, \mathbf{c}$

- Duplicate each of $f(\mathbf{y}, j)$ K times and generate permutation for nK columns (including $\mathbf{c}$) $\{\underbrace{\mathbf{a}_{1,1},\ldots,\mathbf{a}_{1,n}}_{\text{Duplication 1}}, \underbrace{\mathbf{a}_{2,1},\ldots,\mathbf{a}_{2,n}}_{\text{Duplication 2}}, \ldots, \underbrace{\mathbf{a}_{K,1},\ldots,\mathbf{a}_{K,n}}_{\text{Duplication } K}\}$
- Run **Algorithm 1** and get $\hat{\mathbf{x}}_{\text{Dup}} = (\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_K)$
- Aggregate solution $\hat{\mathbf{x}} = \frac{1}{K}\sum_{k=1}^{K}\hat{\mathbf{x}}_k$.

Output: $\hat{\mathbf{x}}$

Remark 8. There is an extra $\mathcal{O}(\frac{\log K}{\sqrt{K}})$ term in the bound for ρ, but note that we generally take $K = \mathcal{O}(n)$ and we drop this term when presenting our results.

Remark 9. We provide an intuitive explanation of the improvement. Taking the explicit update as an example: when each variable is duplicated $K > 1$ times, the final output $\hat{x}^k$ would be allowed to take i/K for $i \le K$, while $K = 1$ only allows $\hat{x}^k \in \{0, 1\}$. In other words, larger K offers higher granularity to approximate fractional solutions.

Till now we have presented the theoretical results of the online algorithms applied to offline LPs. In the following sections, we will focus on how the online algorithms help exact LP solving through sifting.

4. Application: Sifting for Linear Programs

In the previous sections, we have discussed the use of online algorithms to approximately solve offline LPs. However, unlike the simplex method, first-order methods rarely give accurate basis status and generally cannot be applied for exact LP solving. In this section, we try to alleviate this issue by identifying the use of our method in sifting, a "column generation" framework for linear programming. For ease of exposition, we temporarily switch to standard-form LPs

$$\max_{\mathbf{x}} \; \langle \mathbf{c}, \mathbf{x}\rangle \quad \text{subject to } \; \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \ge \mathbf{0}.$$

Sifting for LP Sifting initially appeared in (Forrest, 1989) and was formally presented in (Bixby et al., 1992) to solve LPs with $n \gg m$. Mature LP solvers often use sifting as a candidate solver (Pedroso, 2011; Manual, 1987; Ge et al., 2022). Using the idea that the set of basic columns $\mathcal{B} = \{j : x_j^* > 0\}$ is small relative to $[n]$, sifting solves a

sequence of working problems which restrict the problem to a more tractable subset of columns $\mathcal{W} \subseteq [n]$, $|\mathcal{W}| \ll n$

$$\max_{\mathbf{x}_{\mathcal{W}}} \ \langle \mathbf{c}, \mathbf{x}_{\mathcal{W}} \rangle \quad \text{subject to} \ \ \mathbf{A}\mathbf{x}_{\mathcal{W}} = \mathbf{b}, \mathbf{x}_{\mathcal{W}} \geq \mathbf{0}.$$

Let $\mathbf{y}^*_{\mathcal{W}}$ denote the optimal dual solution to the working problem. If $\mathbf{y}^*_{\mathcal{W}}$ is dual feasible for the original LP, say, $\mathbf{A}^\top \mathbf{y}^*_{\mathcal{W}} \geq \mathbf{c}$, then $\mathcal{B} \subseteq \mathcal{W}$ (assuming the optimal solution is unique) and the original problem is solved. Otherwise we price out the dual infeasible columns $\mathcal{I} = \{j : \langle \mathbf{a}_j, \mathbf{y}^*_{\mathcal{W}} \rangle < c_j\}$ and add them to $\mathcal{W}$.

Algorithm 3 Sifting procedure for LPs

Input: Initial working set $\mathcal{W}$
while $\mathcal{I}$ is nonempty **do**
  Solve working problem
  $\max_{\mathbf{A}_{\mathcal{W}}\mathbf{x}_{\mathcal{W}}=\mathbf{b}, \mathbf{x}_{\mathcal{W}} \geq \mathbf{0}} \langle \mathbf{c}_{\mathcal{W}}, \mathbf{x}_{\mathcal{W}} \rangle$ and get $\mathbf{y}^*_{\mathcal{W}}$
  Update $\mathcal{I} = \{j : \langle \mathbf{a}_j, \mathbf{y}^*_{\mathcal{W}} \rangle < c_j\}$
  $\mathcal{W} = \mathcal{W} \cup \mathcal{I}$
end
Output: Optimal solution to LP

As a special case of column generation, sifting faces challenges that are similar to column generation (Lübbecke, 2010)

- (heading-in) a good initialization of $\mathcal{W}$ is often hard
- (dual-oscillation) solution $\mathbf{y}^*_{\mathcal{W}}$ is unstable at the end

Dual Stabilization Among the techniques for sifting, dual stabilization (Du Merle et al., 1999; Amor et al., 2004; Pessoa et al., 2018) has been one of the most successful attempts. In a word, most of the dual stabilization techniques work by finding some anchor point $\hat{\mathbf{y}}$ that lies at the center of dual feasible region (Lee and Park, 2011; Gondzio et al., 2013) and then by stablilizing the dual iterations by taking convex combination

$$\hat{\mathbf{y}}_{\mathcal{W}} = \alpha \mathbf{y}_{\mathcal{W}} + (1-\alpha)\hat{\mathbf{y}}. \tag{8}$$

However, computing an interior point or some center of the dual feasible region is often too costly for huge LPs.

Accelerate Sifting by Online Algorithm Generally speaking, to accelerate sifting one needs

- a good estimate of $\{j : x^*_j > 0\}$.
- some approximate $\hat{\mathbf{y}} \approx \mathbf{y}^*$
- an inexpensive algorithm that obtains them

To this point it's not hard to see that our online algorithm is a perfect candidate to fulfill all three requirements above. First, we can use $\hat{\mathbf{x}}$ as a score function to build initial $\mathcal{W}$. For example, we can choose some threshold value τ and initialize $\mathcal{W} = \{j : \hat{x}^j \geq \tau\}$. During the sifting procedure, we can use the approximate dual solution from online algorithm to stabilize the dual iterations. Most importantly, online algorithm is effcient and its running time is negligible compared to the whole sifting procedure.

In practice, we can efficiently embed sifting into the LP pre-solver by going through the problem data several times before sifting starts. As our experiments suggest, online algorithm manages to identify the basis status for a wide range of real-life LPs.

Remark 10. Though the online algorithms work for problems with inequality constraints and upper-bounded variables, as the experiments suggest, it often suffices to use the online algorithm as a heuristic in practice even if these constraints are not necessarily satisfied.

Due to space limit we leave a more detailed discussion of sifting to **Section B.4**.

5. Experiments

In this section, we conduct numerical experiments to validate the efficiency of our proposed methods. The experiment is divided into two parts. In the first part we verify our theoretical results on multi-knapsack benchmark dataset and its variants. In the second part, we turn to exact LP solving and test large-scale LP benchmark datasets to see how online algorithms benefit sifting solvers.

5.1. Approximate Solver

In this part we test online algorithms in approximate LP solving. We also compare the performance of explicit and implicit updates in practice.

Data Description We use synthetic data from multi-knapsack benchmark. More detailedly, we generate benchmark multi-knapsack problems $\max_{\mathbf{A}\mathbf{x}\leq\mathbf{b}, \mathbf{0}\leq\mathbf{x}\leq\mathbf{1}} \langle \mathbf{c}, \mathbf{x} \rangle$ as discussed in (Chu and Beasley, 1998): we generate each element of a_{ij} uniformly from $\{1, \ldots, 1000\}$. After generating $\mathbf{A}$, we zero out each element of $\mathbf{A}$ with probability $(1-\sigma)$, where $\sigma \in (0, 1]$ controls the sparsity of $\mathbf{A}$. $\mathbf{b}$ is generated by $b_i = \frac{\tau}{n}\sum_j a_{ij}$, where τ is called tightness coefficient. Each element of $\mathbf{c}$ is generated by $c_i = \frac{1}{m}\sum_i a_{ij} + \delta_i$, where δ_i is sampled uniformly from $\{1, \ldots, 500\}$.

Performance Metric In the first part of our experiment, given a feasible approximate solution $\hat{\mathbf{x}}$, we use relative optimality to measure its quality

$$r(\hat{\mathbf{x}}) := \left| \frac{\langle \mathbf{c}, \hat{\mathbf{x}} \rangle}{\langle \mathbf{c}, \mathbf{x}^* \rangle} \right|.$$

Testing Configuration and Setup

1). **Dataset**. We test $(m, n) \in \{(5, 10^2), (8, 10^3), (16, 2 \times 10^3), (32, 4 \times 10^3)\}, \tau \in [10^{-2}, 1], \sigma = 1$.
2). **Initial Point**. We let online algorithms start from $\mathbf{0}$.
3). **Feasibility**. We force the algorithms to respect constraints (take $\hat{x}^k = 1$ only if $\mathbf{b} - \sum_{j=1}^{k} \mathbf{a}_j x_j \geq \mathbf{0}$).
4). **Duplication**. We allow $K \in \{1, 2, 4, 8, 16, 32\}$
5). **Stepsize**. We take $\gamma = 1/\sqrt{Kmn}$.
6). **Subproblem**. We discuss the way to efficiently compute implicit update in the appendix **Section B.2**.

First we compare the performance of the two algorithms through their relative optimality under different data settings. For LPs of the same size, we fix (m, n, K) and test $\tau \in [10^{-2}, 1]$ for ten values evenly distributed on the log-scale. Figure 1 illustrates the performance of both explicit and implicit updates. It can be clearly seen that implicit update, in most cases, outperforms the explicit update. Especially when the constraint is tight (τ close to 0) and $K = 1$, we see that the performance of implicit updates are dominant. Besides, we observe that with K increasing, explicit update starts to catch up.

Next we examine the efficiency of the variable duplication scheme by fixing (m, n, τ) and increasing $K \in \{1, 2, 4, 8, 16, 32\}$. Figure 2 illustrates how variable duplication improves performance of the online algorithms and shows its potential in approximate LP solving. It can be seen that as K increases, relative optimality is gradually improved and we can achieve higher than 90% relative optimality given moderate K. Therefore, it suffices to adopt the online algorithm with variable duplication in the applications where an approximately optimal solution is acceptable. In the appendix **Section B.3** we further investigate the performance of online algorithms for direct LP solving.

Another observation from Figure 2 is that when τ is close to 0, as we just mentioned, explicit update is dominated by the implicit update due to the restriction of constraint violation. However, as we increase K, this gap is soon filled. This suggests in practice we can alternatively combine variable duplication with explicit update to achieve comparable performance to the implicit update. Especially when the constraint matrix is sparse, an $\mathcal{O}(\text{nnz}(\mathbf{A}))$ implementation would be fairly competitive.

5.2. Sifting and Large LPs

In this part, we turn to sifting in exact LP solving and see how much online algorithms can help.

Data Description We test both synthetic data and real-life large-scale LPs. For the synthetic data, we generate LP instances in the same way as in **Section 5.1.** Besides, we collect 13 instances from (Mittelmann, 2022).

Table 2. Datasets collected from large-scale LP benchmark

Dataset	Row	Column	Dataset	Row	Column
`rail507`	507	6.4e+04	`rail516`	516	4.8e+04
`rail582`	582	5.6e+04	`rail2586`	2586	9.2e+05
`rail4284`	4284	1.0e+06	`scpm1`	5000	5.0e+05
`scpn2`	5000	1.0e+06	`scpl4`	2000	2.0e+05
`scpj4scip`	1000	1.0e+05	`scpk4`	2000	1.0e+05
`s82`	87878	1.7e+06	`s100`	14733	3.6e+05
`s250r10`	10962	2.7e+05	-	-	-

Performance Metric We mention in **Section 4** that online algorithm helps sifting in **1)**. working problem initialization **2)**. dual stabilization, and we test these two aspects using different criteria. For working problem initialization, assume that we get an initial working problem estimate $\mathcal{W}$ from the online algorithm and denote by $\mathcal{B}$ the set of basic columns, then we use

$$\text{acc}(\hat{\mathbf{x}}) := \frac{|\mathcal{B} \cap \mathcal{W}|}{|\mathcal{B}|} \quad \text{and} \quad \text{rdc}(\hat{\mathbf{x}}) := \frac{|\mathcal{W}|}{n}$$

to respectively evaluate **1)**. how many basic columns are found out. **2)**. size of the initialized working problem relative to the original LP. On the other hand, we also need to evaluate how much, overall, online algorithm can accelerate a sifting solver. To this end we directly use CPU seconds $T(\hat{\mathbf{x}})$ as the metric.

Testing Configuration and Setup

1). **Dataset**. We use $(m, n) \in \{(10^2, 10^5), (10^2, 10^6)\}$, $\tau \in \{(0.05, 0.1)\}$ and $\sigma \in \{(0.01, 0.1, 0.15, 0.2)\}$
2). **Algorithm Selection**. Since the problems are all sparse, only explicit update is used.
3). **Initial Point**. We let online algorithm start from $\mathbf{1}$.
4). **Duplication**. We take $K = 2$ for all the datasets.
5). **Stepsize**. We take $\gamma = 1/\sqrt{Kmn}$.
6). **Basis Prediction**. Given the output of online algorithm $\hat{\mathbf{x}}$, we use $\mathcal{W} = \{j : \hat{x}^j \geq 1/K\}$ to initialize the working problem.
7). **Dual Stabilization**. We implement a basic dual stabilization procedure which takes $\alpha = 0.4$ in (8).
8). **Sifting Solver**. We adopt the sifting solver in `CPLEX 12.10` as our benchmark solver.

Table 3 describes the practical performance of the the online algorithm applied to practical large-scale LPs. We can see that on the collected benchmark datasets, online algorithm successfully identifies more than 90% of the basic

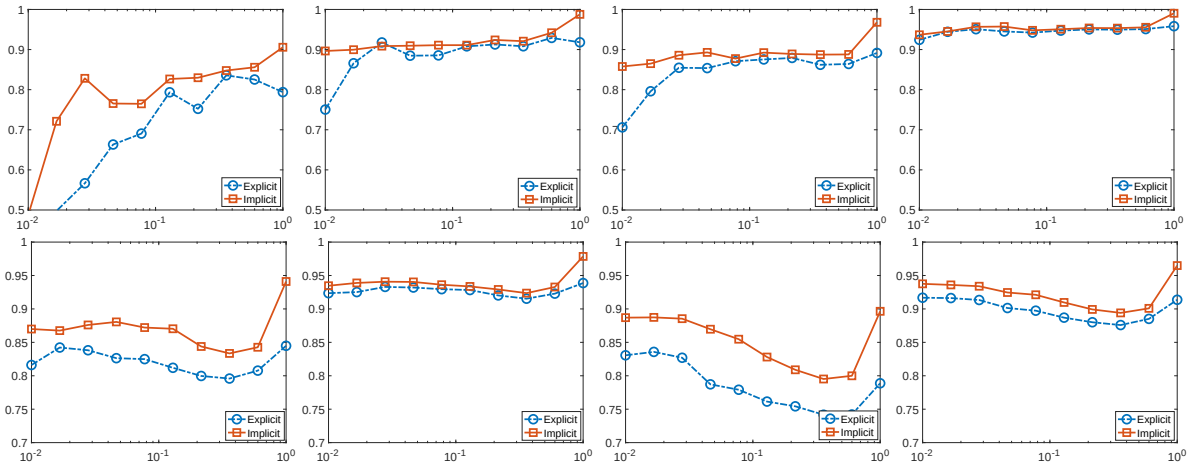

Figure 1. First row from left to right $(m, n, K) \in \{(5, 10^2, 1), (5, 10^2, 8), (8, 10^3, 1), (8, 10^3, 8)\}$. Second row from left to right $(m, n, K) \in \{(16, 2 \times 10^3, 1), (16, 2 \times 10^3, 8), (32, 4 \times 10^3, 1), (32, 4 \times 10^3, 8)\}$. The x-axis represents τ parameter ranging from 10^{-2} to 1; The y-axis represents the relative optimality.

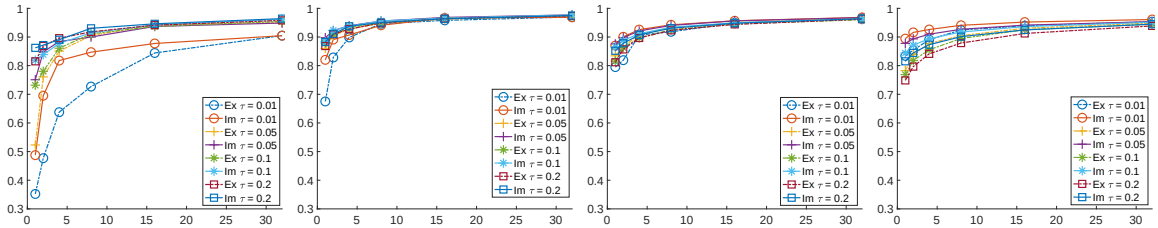

Figure 2. From left to right $(m, n) \in \{(5, 10^2), (8, 10^3), (16, 2 \times 10^3), (32, 4 \times 10^3)\}$. The x-axis represents K parameter ranging in $\{1, 2, 4, 8, 16, 32\}$. The y-axis represents the relative optimality.

columns in most cases while restricting the size of initialization less than 20% of the original LP size. Especially for the `scp` instances, online algorithm identifies all the basic columns with fewer than 5% of columns. In this case, the original LP is solved after the first sifting iteration and we only need to solve a much smaller LP. For the overall LP solving time, we observe a clear speedup on 6 out of the 13 instances, neutral performance on 3 instances and slow down on 4 of the instances. We interpret this slow-down as the effect of our preliminary implementation of the sifting solver compared to `CPLEX`.

Finally, we experiment on synthetic datasets that satisfy the assumptions **A1** to **A3**. As **Table 4** shows, sifting solver, combined with our online algorithm, often outperforms the commercial sifting solvers by more than 50%. This further illustrates the practical efficiency of our proposed method.

6. Conclusions

We adapt two fast online algorithms for offline LPs and obtain algorithms that are free of any matrix multiplication or access to the full LP constraint matrix. We theoretically analyze the optimality gap and constraint violation of the two algorithms and propose a variable-duplication scheme to improve their practical performance. In addition, we identify the potential of online algorithms in exact LP solving when combined with sifting, an LP column generation procedure. Our numerical experiments demonstrate the efficiency of online algorithms, both as an approximate direct solver and as an auxiliary routine in sifting. We believe that it is an interesting direction to introduce online algorithms to the context of offline LP solving.

7. Acknowledgement

We thank Yuyang Ye, Xiaocheng Li, and the seminar participants at Shanghai University of Finance and Economics for their fruitful discussions and comments. Especially, we thank Dr. Qi Huangfu for proposing an $\mathcal{O}(\text{nnz}(\mathbf{A}))$ implementation of the explicit update method and testing it in the state-of-the-art commercial solver `COPT` (Ge et al., 2022).

Finally, we sincerely appreciate the efforts the Area Chairs and all the reviewers put into the review process.

Table 3. Performance of initializing working problem and CPU solution time of the large-scale LPs

Dataset	acc($\hat{\mathbf{x}}$)	rdc($\hat{\mathbf{x}}$)	T_{CPLEX}	$T(\hat{\mathbf{x}})$
rail507	271/301	11862/62171	0.50	0.72
rail516	121/138	8572/46978	0.38	0.73
rail582	325/347	12465/54315	0.60	1.15
rail2586	1536/1672	145373/909940	5.44	10.65
rail4284	1951/2042	348135/1090526	14.13	22.64
scpm1	2754/2754	10352/500000	19.08	8.74
scpn2	3411/3411	20860/1000000	51.38	12.99
scpl4	1149/1149	5718/200000	0.94	0.78
scpj4scip	552/552	3635/99947	0.47	0.39
scpk4	930/930	4077/100000	0.60	0.58
s82	1992/3020	52383/1687859	>3600	>3600
s100	150/487	835/364203	151.65	38.30
s250r100	415/747	3080/270323	23.07	21.55

Table 4. CPU time of sifting on synthetic datasets

$(m,n)=(10^2,10^5)$				$(m,n)=(10^2,10^6)$			
τ	σ	T_{CPLEX}	$T(\hat{\mathbf{x}})$	τ	σ	T_{CPLEX}	$T(\hat{\mathbf{x}})$
0.05	0.1%	8.08	3.43	0.05	0.1%	82.79	38.20
0.05	0.5%	7.73	4.45	0.05	0.5%	117.02	73.39
0.05	1%	7.86	2.81	0.05	1%	61.52	35.94
0.05	5%	7.76	3.88	0.05	5%	82.77	35.84
0.05	10%	7.82	3.52	0.05	10%	74.89	37.67
0.05	15%	7.96	6.30	0.05	15%	65.03	40.75
0.05	20%	7.57	3.89	0.05	20%	70.21	37.13
0.10	0.1%	9.19	5.57	0.10	0.1%	79.35	67.30
0.10	0.5%	7.57	2.71	0.10	0.5%	58.34	47.50
0.10	1%	7.63	4.48	0.10	1%	54.69	42.73
0.10	5%	9.08	6.51	0.10	5%	56.13	52.58
0.10	10%	8.30	5.51	0.10	10%	56.96	47.71
0.10	15%	8.93	4.33	0.10	15%	52.40	53.79
0.10	20%	7.82	5.18	0.10	20%	57.77	42.21

8. Disclosure of Funding

This research is partially supported by the National Natural Science Foundation of China (NSFC) [Grant NSFC-72150001, 72225009, 11831002].

References

S. Agrawal, Z. Wang, and Y. Ye. A dynamic near-optimal algorithm for online linear programming. *Operations Research*, 62(4):876–890, 2014.

H. B. Amor, J. Desrosiers, and A. Frangioni. *Stabilization in column generation*. Groupe d'études et de recherche en analyse des décisions, 2004.

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

H. Asi and J. C. Duchi. Stochastic (approximate) proximal point methods: Convergence, optimality, and adaptivity. *SIAM Journal on Optimization*, 29(3):2257–2290, 2019.

S. Balseiro, H. Lu, and V. Mirrokni. Dual mirror descent for online allocation problems. In *International Conference on Machine Learning*, pages 613–628. PMLR, 2020.

S. R. Balseiro, H. Lu, and V. Mirrokni. The best of many worlds: Dual mirror descent for online allocation problems. *Operations Research*, 2022.

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

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

R. L. Bray. Logarithmic regret in multisecretary and online linear programming problems with continuous valuations. *arXiv e-prints*, pages arXiv–1912, 2019.

K. Chen, A. Cutkosky, and F. Orabona. Implicit parameter-free online learning with truncated linear models. In *International Conference on Algorithmic Learning Theory*, pages 148–175. PMLR, 2022.

P. C. Chu and J. E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics*, 4(1):63–86, 1998.

G. Dantzig. Linear programming and extensions. In *Linear programming and extensions*. Princeton university press, 2016.

Q. Deng and W. Gao. Minibatch and momentum model-based methods for stochastic weakly convex optimization. *Advances in Neural Information Processing Systems*, 34:23115–23127, 2021.

Q. Deng, Q. Feng, W. Gao, D. Ge, B. Jiang, Y. Jiang, J. Liu, T. Liu, C. Xue, Y. Ye, et al. New developments of admm-based interior point methods for linear programming and conic programming. *arXiv preprint arXiv:2209.01793*, 2022.

O. Du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discrete Mathematics*, 194(1-3):229–237, 1999.

J. Forrest. Mathematical programming with a library of optimization subroutines. In *ORSA/TIMS Joint National Meeting, New York*, 1989.

K. Fountoulakis and J. Gondzio. Performance of first-and second-order methods for l1-regularized least squares problems. *Computational Optimization and Applications*, 65(3):605–635, 2016.

M. Garstka, M. Cannon, and P. Goulart. Cosmo: A conic operator splitting method for convex conic problems. *Journal of Optimization Theory and Applications*, 190 (3):779–810, 2021.

D. Ge, Q. Huangfu, Z. Wang, J. Wu, and Y. Ye. Cardinal optimizer (copt) user guide. *arXiv preprint arXiv:2208.14314*, 2022.

J. Gondzio, P. González-Brevis, and P. Munari. New developments in the primal–dual column generation technique. *European Journal of Operational Research*, 224 (1):41–51, 2013.

E. Hazan et al. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016.

J. Jiang, W. Ma, and J. Zhang. Degeneracy is OK: Logarithmic Regret for Network Revenue Management with Indiscrete Distributions. *arXiv*, 2022. doi: 10.48550/arxiv.2210.07996.

B. Kulis and P. L. Bartlett. Implicit online learning. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 575–582, 2010.

C. Lee and S. Park. Chebyshev center based column generation. *Discrete Applied Mathematics*, 159(18):2251–2265, 2011.

X. Li and Y. Ye. Online linear programming: Dual convergence, new algorithms, and regret bounds. *Operations Research*, 70(5):2948–2966, 2022.

X. Li, C. Sun, and Y. Ye. Simple and fast algorithm for binary integer and online linear programming. *Advances in Neural Information Processing Systems*, 33:9412–9421, 2020.

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.

M. E. Lübbecke. Column generation. *Wiley encyclopedia of operations research and management science*, 2010.

W. Ma, Y. Cao, D. H. K. Tsang, and D. Xia. Optimal Regularized Online Convex Allocation by Adaptive Re-Solving. *arXiv*, 2022. doi: 10.48550/arxiv.2209.00399.

C. U. Manual. Ibm ilog cplex optimization studio. *Version*, 12(1987-2018):1, 1987.

N. Megiddo and R. Chandrasekaran. On the ε-perturbation method for avoiding degeneracy. *Operations Research Letters*, 8(6):305–308, 1989.

H. Mittelmann. Benchmark of barrier lp solvers, 2022.

Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

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

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, 2016.

J. P. Pedroso. Optimization with gurobi and python. *INESC Porto and Universidade do Porto,, Porto, Portugal*, 1, 2011.

A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck. Automation and combination of linear-programming based stabilization techniques in column generation. *INFORMS Journal on Computing*, 30(2):339–360, 2018.

K. T. Talluri, G. Van Ryzin, and G. Van Ryzin. *The theory and practice of revenue management*, volume 1. Springer, 2004.

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

Appendix

Table of Contents

A Theoretical Aspects of Online Linear Programming **13**
A.1 Auxiliary Results . . . 13
A.2 Proof of Results in Section 3.1 . . . 14
A.3 Proof of Results in Section 3.2 . . . 15
A.4 Proof of Results in Section 3.3 . . . 19
A.5 Proof of Theorem 3 . . . 19
A.6 Additional Experiment on Violated Assumption . . . 20

B Practical Aspects of Online Linear Programming **21**
B.1 Fast $\mathcal{O}(\text{nnz}(\mathbf{A}))$ Implementation For the Explicit Update . . . 21
B.2 Cases of Simple Implicit Update . . . 21
B.3 Direct Approximate LP Solving . . . 22
B.4 Sifting and Column Generation . . . 23
B.5 Integer Programming . . . 26

Structure of the Appendix

The appendix is organized as follows. **Section A** introduces auxiliary results and proves our main results from **Section 3**. As a complement to our discussions in **Section 5**, **Section B** gives a more comprehensive treatment of online linear programming applied to **1)**. Direct LP solving. **2)**. LP sifting and column generation. **3)**. Integer programming. Particularly we discuss the practical implementation of both explicit and implicit update in **Section B.1** and **Section B.2**.

A. Theoretical Aspects of Online Linear Programming

A.1. Auxiliary Results

In this section, we present some auxiliary results from (Li et al., 2020) that will help in the proof. Recall that given an index set $\mathcal{S} \subseteq [n]$, we use $\mathbf{A}_{\mathcal{S}}$ to denote the sub-matrix indexed from columns of $\mathbf{A}$ and use $\mathbf{c}_{\mathcal{S}}$ to denote a sub-vector indexed from $\mathbf{c}$. Then we introduce two auxiliary LPs as follows.

Auxiliary LPs from Proposition 1 of (Li et al., 2020)

$$\begin{aligned} F_s^* := \max_{\mathbf{x}_{[1:s]}} \quad & \langle \mathbf{c}_{[1:s]}, \mathbf{x}_{[1:s]} \rangle \\ \text{subject to} \quad & \mathbf{A}_{[1:s]}\mathbf{x}_{[1:s]} \leq \tfrac{s\mathbf{b}}{n} \\ & \mathbf{0} \leq \mathbf{x}_{[1:s]} \leq \mathbf{1} \end{aligned}$$

$$\begin{aligned} \hat{F}_k^* := \max_{\mathbf{x}_{[k:n]}} \quad & \langle \mathbf{c}_{[k:n]}, \mathbf{x}_{[k:n]} \rangle \\ \text{subject to} \quad & \mathbf{A}_{[k:n]}\mathbf{x}_{[k:n]} \leq (1 - \tfrac{k-1}{n})\mathbf{b} \\ & \mathbf{0} \leq \mathbf{x}_{[k:n]} \leq \mathbf{1}, \end{aligned}$$

The optimal values of of the two LPs are denoted by F_s^* and $\hat{F}_k^*$ respectively. The following lemma provides a lens to deal with random permutation.

Lemma 3 (Optimality gap (Li et al., 2020))**.** *We have the following bound on the optimality gap of the online algorithm*

$$\begin{aligned} F_n^* - \sum_{k=1}^n \mathbb{E}[c_k x^k] &\leq m\bar{c} + \frac{\bar{c}\sqrt{n}\log n}{\underline{d}} + m\bar{c}\log n + \frac{\alpha\bar{c}}{n} + \sum_{k=1}^n \mathbb{E}\Big[\tfrac{\hat{F}_{n-k+1}^*}{n-k+1} - c_k x^k\Big] \\ &= \mathcal{O}(m\log n + \sqrt{n}\log n) + \sum_{k=1}^n \mathbb{E}\Big[\tfrac{\hat{F}_{n-k+1}^*}{n-k+1} - c_k x^k\Big], \end{aligned}$$

where $\alpha = \max\{e, e^{16\bar{a}^2}, 16\bar{a}^2\}$ and the expectation is taken over the random permutation.

Remark 11. Given **Lemma 3**, it remains to analyze the quantity $\sum_{k=1}^n \mathbb{E}\Big[\frac{\hat{F}_{n-k+1}^*}{n-k+1} - c_k x^k\Big]$ to bound the optimality gap, and in our proof we analyze it under explicit and implicit updates respectively.

We also need a well-known three-point lemma in the analysis of implicit updates.

Lemma 4 (Three-point lemma (Beck, 2017))**.** *Let f be a convex function and*

$$\mathbf{y}^+ = \arg\min_{\mathbf{y}} \Big\{ f(\mathbf{y}) + \frac{1}{2\gamma}\|\mathbf{y} - \mathbf{z}\|^2 \Big\},$$

where $\gamma > 0$. Then we have

$$f(\mathbf{y}^+) + \frac{1}{2\gamma}\|\mathbf{y}^+ - \mathbf{z}\|^2 \leq f(\mathbf{y}) + \frac{1}{2\gamma}\|\mathbf{y} - \mathbf{z}\|^2 - \frac{1}{2\gamma}\|\mathbf{y} - \mathbf{y}^+\|^2.$$

Clue of proof With the tools in hand, now we get down to the analysis of different algorithms. Our proof basically follows three steps as follows:

- Tracking the dual iteration $\{\mathbf{y}^k\}$
- Tracking constraint violation and optimality gap
- Taking trade-off between optimality gap and constraint violation by properly choosing γ.

A.2. Proof of Results in Section 3.1

In this section, we present the proof for subgradient-based explicit online algorithm.

The following lemma tracks the dual iterations of **Algorithm 1** with explicit update.

Lemma 5 (Tracking the dual iteration)**.** *Under* **A1** *and* **A2***, if we let* $\{\mathbf{y}^k\}$ *be the sequence of dual iterates generated by* ***Algorithm 1*** *with explicit update and* $\gamma_k \equiv \gamma$*, then*

$$\|\mathbf{y}^k\| \leq \frac{m(\bar{a}+\bar{d})^2\gamma}{\underline{d}} + \sqrt{m}(\bar{a}+\bar{d})\gamma + \frac{\bar{c}}{\underline{d}}.$$

Boundedness of dual iterations turns out to be important for limiting the constraint violation. Now we state the detailed proof of the theoretical results.

A.2.1. Proof of Lemma 5

The proof is adapted from (Li et al., 2020) and is improved via a sharper analysis. First recall that we update the dual iterations by

$$\mathbf{y}^{k+1} = [\mathbf{y}^k + \gamma(\mathbf{a}_k \mathbb{I}\{c_k > \langle \mathbf{a}_k, \mathbf{y}^k\rangle\} - \mathbf{d})]_+ = [\mathbf{y}^k + \gamma(\mathbf{a}_k x^k - \mathbf{d})]_+$$

and we successively deduce that

$$\begin{aligned}
&\|\mathbf{y}^{k+1}\|^2 - \|\mathbf{y}^k\|^2 \\
&\leq \|\mathbf{y}^k + \gamma(\mathbf{a}_k x^k - \mathbf{d})\|^2 - \|\mathbf{y}^k\|^2 && (9)\\
&\leq -2\gamma\langle \mathbf{d} - \mathbf{a}_k x^k, \mathbf{y}^k\rangle + m(\bar{a}+\bar{d})^2\gamma^2 && (10)\\
&\leq m(\bar{a}+\bar{d})^2\gamma^2 + 2\gamma\bar{c} - 2\gamma\underline{d}\|\mathbf{y}^k\|, && (11)
\end{aligned}$$

where (9) is due to $\|[\mathbf{x}]_+\| \leq \|\mathbf{x}\|$; (10) uses **A1**, **A2** to get $\gamma^2\|\mathbf{a}_k x^k - \mathbf{d}\|^2 \leq m(\bar{a}+\bar{d})^2\gamma^2$ since $\|\mathbf{a}_k\|_\infty \leq \bar{a}, \mathbf{0} < \mathbf{d} \leq \bar{d}\cdot\mathbf{1}$ and $|x^k| \leq 1$; (11) uses the relation

$$-2\gamma\langle \mathbf{d}, \mathbf{y}^k\rangle \leq -2\gamma\underline{d}\|\mathbf{y}^k\|_1 \leq -2\gamma\underline{d}\|\mathbf{y}^k\|_2$$

and $\langle \mathbf{a}_k x^k, \mathbf{y}^k\rangle \leq c_k \leq \bar{c}$.

On the other hand, we have, by triangle inequality that

$$\|\mathbf{y}^{k+1}\| \leq \|\mathbf{y}^k + \gamma(\mathbf{a}_k x^k - \mathbf{d})\| \leq \|\mathbf{y}^k\| + \gamma\|\mathbf{a}_k x^k - \mathbf{d}\| \leq \|\mathbf{y}^k\| + \gamma\sqrt{m}(\bar{a}+\bar{d})$$

and if $\|\mathbf{y}^k\| \geq \frac{m(\bar{a}+\bar{d})^2\gamma+2\bar{c}}{2\underline{d}}$, by (11) we know that

$$\|\mathbf{y}^{k+1}\|^2 - \|\mathbf{y}^k\|^2 \leq m(\bar{a}+\bar{d})^2\gamma^2 + 2\gamma\bar{c} - 2\gamma\underline{d}\|\mathbf{y}^k\| \leq 0.$$

Therefore, if $\|\mathbf{y}^1\| \leq \frac{m(\bar{a}+\bar{d})^2\gamma+2\bar{c}}{2\underline{d}}$, then $\|\mathbf{y}^k\|$ never exceeds $\frac{m(\bar{a}+\bar{d})^2\gamma+2\bar{c}}{2\underline{d}} + \gamma\sqrt{m}(\bar{a}+\bar{d})$ and this completes the proof.

A.2.2. Proof of Lemma 1

By the updating formula of subgradient,

$$\mathbf{y}^{k+1} = [\mathbf{y}^k + \gamma(\mathbf{a}x^k - \mathbf{d})]_+ \geq \mathbf{y}^k + \gamma(\mathbf{a}_k x^k - \mathbf{d})$$

and telescoping over $k = 1, \ldots, n$ gives

$$\mathbf{A}\hat{\mathbf{x}} = \sum_{k=1}^n \mathbf{a}_k x^k \leq \mathbf{b} + \gamma^{-1}\sum_{k=1}^n(\mathbf{y}^{k+1} - \mathbf{y}^k) \leq \mathbf{b} + \gamma^{-1}\mathbf{y}^{n+1}.$$

Re-arranging the term, taking positive part and taking expectation with respect to the random permutation, we have

$$\begin{aligned}\mathbb{E}[\|[\mathbf{A}\hat{\mathbf{x}}-\mathbf{b}]_+\|] &\le \gamma^{-1}\mathbb{E}[\|\mathbf{y}^{n+1}\|] \\ &\le \gamma^{-1}\left[\frac{m(\bar{a}+\bar{d})^2\gamma}{\underline{d}}+\gamma\sqrt{m}(\bar{a}+\bar{d})+\frac{\bar{c}}{\underline{d}}\right] \qquad (12)\\ &= \frac{m(\bar{a}+\bar{d})^2}{\underline{d}}+\sqrt{m}(\bar{a}+\bar{d})+\frac{\bar{c}}{\gamma\underline{d}},\end{aligned}$$

where (12) invokes **Lemma 5** to bound the constraint violation.

Next we bound $\sum_{k=1}^n \mathbb{E}\left[\frac{\hat{F}^*_{n-k+1}}{n-k+1}-c_k x^k\right]$ and deduce that

$$\begin{aligned}&\|\mathbf{y}^{k+1}\|^2-\|\mathbf{y}^k\|^2\\ =&\|[\mathbf{y}^k+\gamma(\mathbf{a}_k x^k-\mathbf{d})]_+\|^2-\|\mathbf{y}^k\|^2\\ \le&\|\mathbf{y}^k+\gamma(\mathbf{a}_k x^k-\mathbf{d})\|^2-\|\mathbf{y}^k\|^2\\ =&-2\gamma\langle\mathbf{d}-\mathbf{a}_k x^k,\mathbf{y}^k\rangle+\gamma^2\|\mathbf{a}_k x^k-\mathbf{d}\|^2\\ \le&-2\gamma\langle\mathbf{d}-\mathbf{a}_k x^k,\mathbf{y}^k\rangle+m(\bar{a}+\bar{d})^2\gamma^2,\end{aligned}$$

where the last inequality again uses the relation $\gamma^2\|\mathbf{a}_k x^k-\mathbf{d}\|^2\le m(\bar{a}+\bar{d})^2\gamma^2$. Then we take expectation and telescope over $k=1,\ldots,n$ to obtain

$$\begin{aligned}\sum_{k=1}^n\mathbb{E}[\|\mathbf{y}^{k+1}\|^2-\|\mathbf{y}^k\|^2] &= \mathbb{E}[\|\mathbf{y}^{n+1}\|^2]-\|\mathbf{y}^1\|^2\\ &\le \sum_{k=1}^n\mathbb{E}[-2\gamma\langle\mathbf{d}-\mathbf{a}_k x^k,\mathbf{y}^k\rangle+m(\bar{a}+\bar{d})^2\gamma^2].\end{aligned}$$

Re-arranging the terms, we have

$$\mathbb{E}\Big[\sum_{k=1}^n 2\gamma\langle\mathbf{d}-\mathbf{a}_k x^k,\mathbf{y}^k\rangle\Big]\le\sum_{k=1}^n m(\bar{a}+\bar{d})^2\gamma^2-\mathbb{E}[\|\mathbf{y}^{n+1}\|^2]\le mn(\bar{a}+\bar{d})^2\gamma^2,$$

Last we observe that (Li et al., 2020)

$$\sum_{k=1}^n\mathbb{E}\left[\frac{\hat{F}^*_{n-k+1}}{n-k+1}-c_k x^k\right]\le\sum_{k=1}^n\mathbb{E}[\langle\mathbf{d}-\mathbf{a}_k x^k,\mathbf{y}^k\rangle]\le\frac{m(\bar{a}+\bar{d})^2\gamma n}{2}$$

and plugging the bound back completes the proof.

A.2.3. Proof of Theorem 1

We have, for some Δ independent of γ, that

$$\begin{aligned}\mathbb{E}[\rho(\hat{\mathbf{x}})+v(\hat{\mathbf{x}})] &\le \Delta+\frac{m(\bar{a}+\bar{d})^2\gamma n}{2}+\frac{\bar{c}}{\gamma\underline{d}}\\ (\text{Taking }\gamma=\sqrt{\tfrac{2\bar{c}}{\underline{d}(\bar{a}+\bar{d})^2 mn}}) &= \Delta+2\left(\frac{(\bar{a}+\bar{d})^2\bar{c}}{2\underline{d}}\right)^{1/2}\sqrt{mn},\end{aligned}$$

where $\frac{m(\bar{a}+\bar{d})^2\gamma n}{2}=\frac{\bar{c}}{\gamma\underline{d}}$ minimizes the right hand side and this completes the proof.

A.3. Proof of Results in Section 3.2

In this section, we prove the results for implicit update. First we recall the update of **Algorithm 1** under the implicit update.

Update of Online Implicit Update

$$\begin{aligned}\mathbf{y}^{k+1} &= \arg\min_{\mathbf{y}\geq\mathbf{0}} \left\{\langle \mathbf{d}, \mathbf{y}\rangle + [c_k - \langle \mathbf{a}_k, \mathbf{y}\rangle]_+ + \frac{1}{2\gamma}\|\mathbf{y}-\mathbf{y}^k\|^2\right\} \\ &= \arg\min_{\mathbf{y}\geq\mathbf{0}} \left\{\langle \mathbf{d}, \mathbf{y}\rangle + s + \frac{1}{2\gamma}\|\mathbf{y}-\mathbf{y}^k\|^2 : s \geq c_k - \langle \mathbf{a}_k, \mathbf{y}\rangle, s\geq 0\right\} \\ x^k &= \lambda(s \geq c_k - \langle \mathbf{a}_k, \mathbf{y}\rangle),\end{aligned}$$

where $\lambda(s \geq c_k - \langle \mathbf{a}_k, \mathbf{y}\rangle)$ denotes the Lagrangian multiplier of the constraint $s \geq c_k - \langle \mathbf{a}_k, \mathbf{y}\rangle$. As in the explicit case, we track the dual solution.

Lemma 6 (Tracking the dual iteration)**.** *Under* **A1** *and* **A2***, letting* $\{\mathbf{y}^k\}$ *be the sequence of dual iterates generated by* ***Algorithm 1*** *with implicit update and* $\gamma_k \equiv \gamma$*, then*

$$\begin{aligned}\|\mathbf{y}^{k+1}-\mathbf{y}^k\| &\leq \sqrt{m}(\bar{a}+\bar{d})\gamma \\ \|\mathbf{y}^k\| &\leq \frac{m(\bar{a}+\bar{d})^2\gamma}{\underline{d}} + \sqrt{m}(\bar{a}+\bar{d})\gamma + \frac{\bar{c}}{\underline{d}}.\end{aligned}$$

After bounding the dual iterations, we can move on to tracking constraint violation and establish a bound of optimality gap, thus giving **Lemma 2** and **Theorem 2**.

A.3.1. Proof of Lemma 6

First we prove that $\|\mathbf{y}^{k+1}-\mathbf{y}^k\| \leq \gamma\sqrt{m}(\bar{a}+\bar{d})$. Note that

$$g(\mathbf{y}) = \langle \mathbf{d}, \mathbf{y}\rangle + [c_k - \langle \mathbf{a}_k, \mathbf{y}\rangle]_+ + \delta_{\mathbf{y}\geq\mathbf{0}}$$

is convex, where $\delta_{\mathbf{y}\geq\mathbf{0}}$ denotes the indicator function of $\mathbb{R}^m_+$. Then we invoke three-point lemma and

$$\langle \mathbf{d}, \mathbf{y}^{k+1}\rangle + [c_k - \langle \mathbf{a}_k, \mathbf{y}^{k+1}\rangle]_+ + \frac{1}{2\gamma}\|\mathbf{y}^{k+1}-\mathbf{y}^k\|^2 \leq \langle \mathbf{d}, \mathbf{y}^k\rangle + [c_k - \langle \mathbf{a}_k, \mathbf{y}^k\rangle]_+ - \frac{1}{2\gamma}\|\mathbf{y}^{k+1}-\mathbf{y}^k\|^2.$$

Re-arranging the terms, we successively deduce that

$$\begin{aligned}\gamma^{-1}\|\mathbf{y}^{k+1}-\mathbf{y}^k\|^2 &\leq \langle \mathbf{d}, \mathbf{y}^k - \mathbf{y}^{k+1}\rangle + [c_k - \langle \mathbf{a}_k, \mathbf{y}^k\rangle]_+ - [c_k - \langle \mathbf{a}_k, \mathbf{y}^{k+1}\rangle]_+ \\ &\leq \|\mathbf{d}\|\cdot\|\mathbf{y}^k-\mathbf{y}^{k+1}\| + \left|\langle \mathbf{a}_k, \mathbf{y}^k-\mathbf{y}^{k+1}\rangle\right| \qquad (13) \\ &\leq \sqrt{m}\bar{d}\|\mathbf{y}^k-\mathbf{y}^{k+1}\| + \sqrt{m}\bar{a}\|\mathbf{y}^k-\mathbf{y}^{k+1}\| \qquad (14) \\ &= \sqrt{m}(\bar{a}+\bar{d})\|\mathbf{y}^k-\mathbf{y}^{k+1}\|,\end{aligned}$$

where (13) uses Cauchy's inequality $\langle \mathbf{a}, \mathbf{b}\rangle \leq \|\mathbf{a}\|\cdot\|\mathbf{b}\|$ and the relation $[x]_+ - [y]_+ \leq |x-y|_+$; (14) again applies Cauchy's inequality together with **A1**, **A2**. Dividing both sides of the inequality by $\|\mathbf{y}^k-\mathbf{y}^{k+1}\|$ shows $\|\mathbf{y}^{k+1}-\mathbf{y}^k\| \leq \gamma\sqrt{m}(\bar{a}+\bar{d})$.

Next we bound $\|\mathbf{y}^k\|$. Due to the complication of implicit update, we resort to a constrained smooth formulation of the proximal subproblem

$$\begin{array}{lcl} \min_{\mathbf{y}} & \langle \mathbf{d}, \mathbf{y}\rangle + s + \frac{1}{2\gamma}\|\mathbf{y}-\mathbf{y}^k\|^2 & \text{Dual} \\ \text{subject to} & s \geq c_k - \langle \mathbf{a}_k, \mathbf{y}\rangle & x \\ & s\geq 0, \mathbf{y}\geq\mathbf{0} & v, \mathbf{w}, \end{array}$$

which is a convex quadratic programming problem. Now we check the Lagrangian function

$$L(\mathbf{y}, x, \mathbf{s}, x, v) = \langle \mathbf{d}, \mathbf{y}\rangle + s + \frac{1}{2\gamma}\|\mathbf{y}-\mathbf{y}^k\|^2 + x(c_k - \langle \mathbf{a}_k, \mathbf{y}\rangle - s) - vs - \langle \mathbf{y}, \mathbf{w}\rangle,$$

where $\mathbf{w}$ is the multiplier of $\mathbf{y}$ and v is the multiplier of s. Writing the KKT conditions, we have

$$\begin{aligned} s &\geq c_k - \langle \mathbf{a}_k, \mathbf{y}^{k+1} \rangle \\ s &\geq 0 \\ \mathbf{d} + \gamma^{-1}(\mathbf{y}^{k+1} - \mathbf{y}^k) - \mathbf{a}_k x^k - \mathbf{w} &= \mathbf{0} \qquad (15) \\ v + x^k &= 1 \\ \langle \mathbf{y}^{k+1}, \mathbf{w} \rangle &= 0 \\ x^k(c_k - \langle \mathbf{a}_k, \mathbf{y}^{k+1} \rangle - s) &= 0 \qquad (16) \\ vs &= 0 \\ (x^k, v, \mathbf{y}^{k+1}, \mathbf{w}) &\geq \mathbf{0} \end{aligned}$$

and from (15) we know that

$$\mathbf{y}^{k+1} = \mathbf{y}^k - \gamma(\mathbf{d} - \mathbf{a}_k x^k) + \gamma \mathbf{w} \qquad (17)$$

for some $\mathbf{w} \geq \mathbf{0}$. Also we notice that since $\langle \mathbf{y}^{k+1}, \mathbf{w} \rangle = 0$, $w_i = 0$ whenever $y_i^{k+1} > 0$, which implies

$$\begin{aligned} \|\mathbf{y}^{k+1}\|^2 &= \|\mathbf{y}^k - \gamma(\mathbf{d} - \mathbf{a}_k x^k) + \gamma \mathbf{w}\|^2 \\ &\leq \|\mathbf{y}^k - \gamma(\mathbf{d} - \mathbf{a}_k x^k)\|^2 \\ &= \|\mathbf{y}^k\|^2 - 2\gamma \langle \mathbf{d} - \mathbf{a}_k x^k, \mathbf{y}^k \rangle + \gamma^2 \|\mathbf{d} - \mathbf{a}_k x^k\|^2 \qquad (18) \\ &\leq \|\mathbf{y}^k\|^2 + 3m(\bar{a} + \bar{d})^2 \gamma^2 + 2\gamma \bar{c} - 2\gamma \underline{d} \|\mathbf{y}^k\|. \qquad (19) \end{aligned}$$

where (19) is again by Cauchy's inequality and **A1**, **A2**, $-2\gamma \langle \mathbf{d}, \mathbf{y}^k \rangle \leq 2\gamma \underline{d} \|\mathbf{y}^k\|_1 \leq -2\gamma \underline{d} \|\mathbf{y}^k\|$, and that

$$\begin{aligned} \langle \mathbf{a}_k x^k, \mathbf{y}^k \rangle &= \langle \mathbf{a}_k x^k, \mathbf{y}^{k+1} \rangle - \langle \mathbf{a}_k x^k, \mathbf{y}^{k+1} - \mathbf{y}^k \rangle \\ &\leq \langle \mathbf{a}_k x^k, \mathbf{y}^{k+1} \rangle + \|\mathbf{a}_k\| \cdot \|\mathbf{y}^{k+1} - \mathbf{y}^k\| \cdot |x^k| \\ &\leq \bar{c} + \gamma m \bar{a}(\bar{a} + \bar{d}) \qquad (20) \\ &\leq \bar{c} + \gamma m(\bar{a} + \bar{d})^2, \end{aligned}$$

where (20) uses $\langle \mathbf{a}_k x^k, \mathbf{y}^{k+1} \rangle \leq \bar{c}$ from (16) and we invoke the bound $\|\mathbf{y}^{k+1} - \mathbf{y}^k\| \leq \gamma \sqrt{m}(\bar{a} + \bar{d})$.

On the other hand, we know that

$$\begin{aligned} \|\mathbf{y}^{k+1}\| &= \|\mathbf{y}^{k+1} - \mathbf{y}^k + \mathbf{y}^k\| \\ &\leq \|\mathbf{y}^{k+1} - \mathbf{y}^k\| + \|\mathbf{y}^k\| \\ &\leq \|\mathbf{y}^k\| + \gamma \sqrt{m}(\bar{a} + \bar{d}), \qquad (21) \end{aligned}$$

where (21) again uses $\|\mathbf{y}^{k+1} - \mathbf{y}^k\| \leq \gamma \sqrt{m}(\bar{a} + \bar{d})$. By exactly the same argument as in **Lemma 5**, we know that

$$\|\mathbf{y}^k\| \leq \frac{3m(\bar{a} + \bar{d})^2 \gamma}{\underline{d}} + \sqrt{m}(\bar{a} + \bar{d})\gamma + \frac{\bar{c}}{\underline{d}}$$

and this completes the proof.

A.3.2. Proof of Lemma 2

First we consider constraint violation and recall that we run implicit update (17)

$$\mathbf{y}^{k+1} = \mathbf{y}^k - \gamma(\mathbf{d} - \mathbf{a}_k x^k) + \gamma \mathbf{w} \geq \mathbf{y}^k - \gamma(\mathbf{d} - \mathbf{a}_k x^k).$$

Hence telescoping gives

$$\mathbb{E}[\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}\|_+] \leq \gamma^{-1} \mathbb{E}[\|\mathbf{y}^{k+1}\|] \leq \frac{3m(\bar{a} + \bar{d})^2}{\underline{d}} + \sqrt{m}(\bar{a} + \bar{d}) + \frac{\bar{c}}{\gamma \underline{d}}.$$

As for the optimality gap, we have, similar to (Li et al., 2020), that

$$\sum_{k=1}^{n}\mathbb{E}\Big[\frac{\hat{F}^*_{n-k+1}}{n-k+1}-c_kx^k\Big]\le\sum_{k=1}^{n}\mathbb{E}[\langle\mathbf{d},\mathbf{y}^k\rangle+[c_k-\langle\mathbf{a}_k,\mathbf{y}^k\rangle]_+-c_kx^k]$$

Now we look again into the KKT conditions, which, after simplification, gives

$$s\ge[c_k-\langle\mathbf{a}_k,\mathbf{y}^{k+1}\rangle]_+ \tag{22}$$

$$x^k(c_k-\langle\mathbf{a}_k,\mathbf{y}^{k+1}\rangle-s)=0 \tag{23}$$

$$(1-x^k)s=0. \tag{24}$$

Combining the above relations, we successively deduce that

$$c_kx^k=(\langle\mathbf{a}_k,\mathbf{y}^{k+1}\rangle+s)x^k \tag{25}$$

$$=\langle\mathbf{a}_k,\mathbf{y}^{k+1}\rangle x^k+sx^k \tag{26}$$

$$=\langle\mathbf{a}_k,\mathbf{y}^{k+1}\rangle x^k+s \tag{27}$$

$$\ge\langle\mathbf{a}_k,\mathbf{y}^{k+1}\rangle x^k+[c_k-\langle\mathbf{a}_k,\mathbf{y}^{k+1}\rangle]_+,$$

where (25) re-arranges (23) and (27) uses (24). Plugging it back, we can derive the following bound.

$$\begin{aligned}&\sum_{k=1}^{n}\mathbb{E}[\langle\mathbf{d},\mathbf{y}^k\rangle+[c_k-\langle\mathbf{a}_k,\mathbf{y}^k\rangle]_+-c_kx^k]\\ \le&\sum_{k=1}^{n}\mathbb{E}[\langle\mathbf{d},\mathbf{y}^k\rangle+[c_k-\langle\mathbf{a}_k,\mathbf{y}^k\rangle]_+-\langle\mathbf{a}_k,\mathbf{y}^{k+1}\rangle x^k-[c_k-\langle\mathbf{a}_k,\mathbf{y}^{k+1}\rangle]_+]\\ =&\sum_{k=1}^{n}\mathbb{E}[\langle\mathbf{d}-\mathbf{a}_kx^k,\mathbf{y}^k\rangle]+\sum_{k=1}^{n}\mathbb{E}[\langle\mathbf{a}_kx^k,\mathbf{y}^k-\mathbf{y}^{k+1}\rangle]+\sum_{k=1}^{n}\mathbb{E}\{[c_k-\langle\mathbf{a}_k,\mathbf{y}^k\rangle]_+-[c_k-\langle\mathbf{a}_k,\mathbf{y}^{k+1}\rangle]_+\}\end{aligned}$$

Next we bound the last two summations by

$$\sum_{k=1}^{n}\mathbb{E}[\langle\mathbf{a}_kx^k,\mathbf{y}^k-\mathbf{y}^{k+1}\rangle]\le\sum_{k=1}^{n}\mathbb{E}[\|\mathbf{a}_k\|\cdot\|\mathbf{y}^k-\mathbf{y}^{k+1}\|]\le mn\gamma\bar{a}(\bar{a}+\bar{d})$$

$$\begin{aligned}\sum_{k=1}^{n}\mathbb{E}\{[c_k-\langle\mathbf{a}_k,\mathbf{y}^k\rangle]_+-[c_k-\langle\mathbf{a}_k,\mathbf{y}^{k+1}\rangle]_+\}&\le\sum_{k=1}^{n}\mathbb{E}[|\langle\mathbf{a}_k,\mathbf{y}^k-\mathbf{y}^{k+1}\rangle|]\\ &\le\sum_{k=1}^{n}\mathbb{E}[\|\mathbf{a}_k\|\cdot\|\mathbf{y}^k-\mathbf{y}^{k+1}\|]\\ &\le mn\gamma\bar{a}(\bar{a}+\bar{d})\end{aligned}$$

with Cauchy's inequality, **Lemma 6** and **A2**. Then we re-arrange (18) and obtain

$$\begin{aligned}&2\gamma\langle\mathbf{d}-\mathbf{a}_kx^k,\mathbf{y}^k\rangle\\ \le&\|\mathbf{y}^k\|^2-\|\mathbf{y}^{k+1}\|^2+\gamma^2\|\mathbf{d}-\mathbf{a}_kx^k\|^2\\ \le&\|\mathbf{y}^k\|^2-\|\mathbf{y}^{k+1}\|^2+\gamma^2m(\bar{a}+\bar{d})^2\end{aligned} \tag{28}$$

and (28) uses **A1** and **A2** and the fact that $|x^k|\le 1$. Finally, we telescope

$$2\gamma\langle\mathbf{d}-\mathbf{a}_kx^k,\mathbf{y}^k\rangle\le\|\mathbf{y}^k\|^2-\|\mathbf{y}^{k+1}\|^2+\gamma^2m(\bar{a}+\bar{d})^2,$$

as in the previous analysis to get

$$\sum_{k=1}^{n}\mathbb{E}[\langle\mathbf{d}-\mathbf{a}_kx^k,\mathbf{y}^k\rangle]\le\frac{m(\bar{a}+\bar{d})^2n\gamma}{2}.$$

Putting all the bounds together, we have

$$\mathbb{E}[\rho(\hat{\mathbf{x}})] \leq \frac{mn(\bar{a}+\bar{d})^2}{2}\gamma + 2mn\gamma\bar{a}(\bar{a}+\bar{d}) \leq \frac{5m(\bar{a}+\bar{d})^2 n\gamma}{2}$$

and this completes the proof.

A.3.3. Proof of Theorem 2

The proof works exactly in the same way as **Theorem 1** by observing that

$$\mathbb{E}[\rho(\hat{\mathbf{x}}) + v(\hat{\mathbf{x}})] \leq \Delta + \frac{5mn(\bar{a}+\bar{d})^2}{2}\gamma + \frac{\bar{c}}{\gamma\underline{d}}$$

and taking optimal $\gamma^* = \sqrt{\frac{2\bar{c}}{5\underline{d}(\bar{a}+\bar{d})^2 mn}}$ to minimize the right hand side.

A.4. Proof of Results in Section 3.3

A.5. Proof of Theorem 3

In this section we consider the variable duplication scheme. Given an LP

$$\begin{aligned} \max_{\mathbf{x}} \quad & \langle \mathbf{c}, \mathbf{x} \rangle \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{0} \leq \mathbf{x} \leq \mathbf{1} \end{aligned}$$

and its duplicated version

$$\begin{aligned} \max_{\{\mathbf{x}_j\}} \quad & \textstyle\sum_{j=1}^K \langle \mathbf{c}, \mathbf{x}_j \rangle \\ \text{subject to} \quad & \textstyle\sum_{j=1}^K \mathbf{A}\mathbf{x}_j \leq K\mathbf{b} \\ & \mathbf{0} \leq \mathbf{x}_j \leq \mathbf{1}, \end{aligned}$$

It's clear that the duplicated LP also satisfies **A1**, **A2**. Also we know that **A3** can be satisfied by an arbitrarily small perturbation of the objective coefficients. Then we immediately have $\sum_{j=1}^K \langle \mathbf{c}, \mathbf{x}_j^* \rangle = K\langle \mathbf{c}, \mathbf{x}^* \rangle$ up to some arbitrarily small perturbation. Suppose that we apply **Theorem 1** or **Theorem 2** to get $\{\mathbf{x}_j'\}$ such that

$$\mathbb{E}[K\langle \mathbf{c}, \mathbf{x}^* \rangle - \sum_{j=1}^K \langle \mathbf{c}, \mathbf{x}_j' \rangle] = \mathcal{O}(m\log n + \sqrt{nK}\log n + \sqrt{mnK} + \sqrt{nK}\log K)$$

$$\mathbb{E}\Big[\Big\|\Big[\sum_{j=1}^K \mathbf{A}\mathbf{x}_j' - K\mathbf{b}\Big]_+\Big\|\Big] = \mathcal{O}(m + \sqrt{mnK}).$$

Then in view of $\hat{\mathbf{x}} = \frac{1}{K}\sum_{j=1}^K \mathbf{x}_j'$ we have

$$\mathbb{E}[\rho(\hat{\mathbf{x}})] = \mathbb{E}\Big[\langle \mathbf{c}, \mathbf{x}^* \rangle - \langle \mathbf{c}, \frac{1}{K}\sum_{k=1}^K \mathbf{x}_k' \rangle\Big] = \mathcal{O}\left(\frac{m\log n}{K} + \sqrt{\frac{n}{K}}\log n + \sqrt{\frac{mn}{K}} + \sqrt{\frac{n}{K}}\log K\right)$$

$$\mathbb{E}[v(\hat{\mathbf{x}})] = \mathbb{E}\Big[\Big\|\Big[\mathbf{A}\Big(\frac{1}{K}\sum_{j=1}^K \mathbf{x}_j'\Big) - \mathbf{b}\Big]_+\Big\|\Big] = \mathcal{O}\left(\frac{m}{K} + \sqrt{\frac{mn}{K}}\right)$$

assuming that $K = \mathcal{O}(n)$, and this completes the proof.

A.6. Additional Experiment on Violated Assumption

In this section, we carry out additional experiment to see what happens when some of our assumptions are (nearly) violated. Specifically we consider the MKP problem with $\mathbf{b} = \mathbf{1}$.

$$\begin{aligned} \max_{\mathbf{x}} \quad & \langle \mathbf{c}, \mathbf{x} \rangle \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{1} \\ & \mathbf{0} \leq \mathbf{x} \leq \mathbf{1}, \end{aligned}$$

Note that when n is large, **A1** becomes asymptotically violated since $\underline{d} = \frac{1}{n}$ becomes closer to 0.

Testing Configuration and Setup We configure the algorithm as follows (**Section 5** gives a more detailed description).

1). Dataset. We take $(m, n) \in \{(5, 100), (8, 1000), (16, 2000), (32, 4000)\}, \sigma = 1$.

2). Initial Point. We let online algorithms start from $\mathbf{0}$.

3). Feasibility. We force the algorithm to respect constraint violation.

4). Duplication. We allow $K \in \{1, 2, 4, 8, 16, 32, 64, 128\}$

5). Stepsize. We take $\gamma = (Kmn)^{-1/2}$.

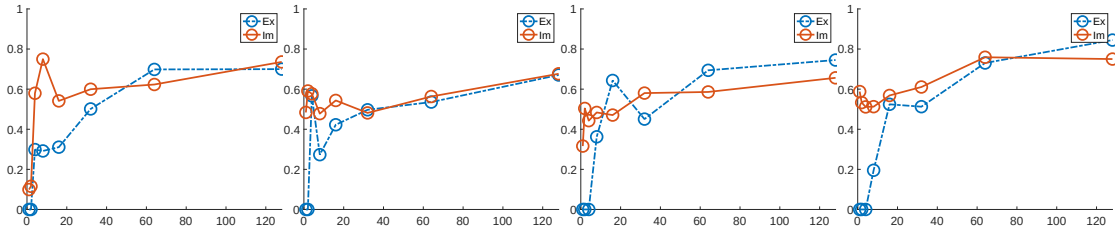

Figure 3. From left to right: $(m, n) \in \{(5, 100), (8, 1000), (16, 2000), (32, 4000)\}$ The axis represents K parameter ranging from 1 to 128. y-axis, relative optimality gap

We see that as suggested by our theory, as $\underline{d} = \frac{1}{n}$ gets smaller, online algorithms perform poorly when $K = 1$, but when we increase K, both explicit and implicit update gradually retrieve better solutions.

B. Practical Aspects of Online Linear Programming

In this section, we discuss the practical aspects of online linear programming in more details. In **Section B.1** and **Section B.2**, we show when and how online updates can be carried out efficiently; In **Section B.2**, we apply online explicit update to more LP problem types and compare its performance with other LP solvers; In **Section B.4**, we formalize the contents of LP sifting and present more implementation details. Finally we show in **Section B.5** that online LP is applicable to speedup mixed integer programming.

B.1. Fast $\mathcal{O}(\mathbf{nnz}(\mathbf{A}))$ Implementation For the Explicit Update

In this section, we discuss the practical aspects of implementing the explicit update. The computation of explicit update comes from the following two steps

$$\begin{aligned} x^k &\leftarrow \mathbb{I}\{c_k > \langle \mathbf{a}_k, \mathbf{y}\rangle\} \\ \mathbf{y}^{k+1} &\leftarrow [\mathbf{y}^k + \mathbf{a}_k x^k - \mathbf{d}]_+, \end{aligned}$$

where we take $\gamma = 1, d_i \equiv d$ without loss of generality. Due to the `axpy` operation $\mathbf{y} + \mathbf{a}_k x^k - \mathbf{d}$ and the full density of $\mathbf{d}$, a direct implementation of `axpy` and $[\mathbf{x}]_+$ will immediately result in $\mathcal{O}(mn)$ flops and is undesirable. To address this, we define $p_j(k) \coloneqq \max_l\{l < k : a_{l,j} \neq 0\}$, the latest iteration l where $a_{l,j}$ is nonzero. Then we observe that

$$\begin{aligned} y_j^{k+1} &= [y_j^k - a_{k,j}x^k - d]_+ \\ &= [[y_j^{k-1} - a_{k-1,j}x^{k-1} - d]_+ - a_{k,j}x^k - d]_+ \\ &= \cdots \\ &= [[y_j^{p(k)} - a_{p(k),j}x^{p_j(k)} - d]_+ - (k - p(k))d]_+, \end{aligned}$$

where we use the relation $[[a-b]_+ - b]_+ = [a - 2b]_+, b \geq 0$ recursively. Further we observe that

$$a_j^{k+1} y_j^{k+1} = a_j^{k+1}[[y_j^{p(k)} - a_{p(k),j}x^{p_j(k)} - d]_+ - (k - p(k))d]_+,$$

which implies that we only need to evaluate $[[y_j^{p(k)} - a_{p(k),j}x^{p_j(k)} - d]_+ - (k - p(k))d]_+$ if $a_j^{k+1} \neq 0$ and the operation takes $\mathcal{O}(1)$. Hence we have overall complexity of $\mathcal{O}(\text{nnz}(\mathbf{A}))$ to compute all the primal estimates and $\mathcal{O}(m)$ to recover $\mathbf{y}^{n+1}$. In practice we can maintain an $\mathbb{N}^m$ array `lastUpdate[m]` to record and to update $p_j(k)$ for each $j \in [m]$.

B.2. Cases of Simple Implicit Update

In this section we focus on the practical implementation of the online implicit update and discuss the case where the implicit is easier to compute. Here we consider

$$\begin{aligned} \min_{\mathbf{y}} \quad & \langle \mathbf{d}, \mathbf{y}\rangle + [c - \langle \mathbf{a}, \mathbf{y}\rangle]_+ + \tfrac{1}{2\gamma}\|\mathbf{y} - \mathbf{z}\|^2 \\ \text{subject to} \quad & \mathbf{y} \geq \mathbf{0}. \end{aligned}$$

where $\mathbf{a} \geq \mathbf{0}$. Let $\mathbf{y}^+$ be the optimal solution to the problem and we do case analysis.

Case 1. $c - \langle \mathbf{a}, \mathbf{y}^+\rangle > 0$. Then $\mathbf{y}^+ = \mathbf{z} - \gamma^{-1}(\mathbf{a} - \mathbf{d})$.

Case 2. $c - \langle \mathbf{a}, \mathbf{y}^+\rangle < 0$. Then $\mathbf{y}^+ = \mathbf{z} - \gamma^{-1}\mathbf{d}$.

Case 3. $c - \langle \mathbf{a}, \mathbf{y}^+\rangle = 0$. Then we have, equivalently, that $\mathbf{y}^+$ is the optimal solution to the following problem.

$$\begin{aligned} \min_{\mathbf{y}} \quad & \langle \mathbf{d}, \mathbf{y}\rangle + \tfrac{1}{2\gamma}\|\mathbf{y} - \mathbf{z}\|^2 \\ \text{subject to} \quad & \langle \mathbf{a}, \mathbf{y}\rangle = c \\ & \mathbf{y} \geq \mathbf{0}. \end{aligned}$$

Since $\mathbf{a} \geq \mathbf{0}$, letting $(A, \bar{A})$ be the partition of $[m]$ such that $\mathbf{a}_A > \mathbf{0}$ and we can separate the problem into

$$\begin{aligned} \min_{\mathbf{y}_{\bar{A}}} \quad & \langle \mathbf{d}_{\bar{A}}, \mathbf{y}_{\bar{A}}\rangle + \tfrac{1}{2\gamma}\|\mathbf{y}_{\bar{A}} - \mathbf{z}_{\bar{A}}\|^2 \\ \text{subject to} \quad & \mathbf{y}_{\bar{A}} \geq \mathbf{0} \end{aligned}$$

and

$$\begin{aligned} \min_{\mathbf{y}_A} \quad & \langle \mathbf{d}_A, \mathbf{y}_A \rangle + \tfrac{1}{2\gamma}\|\mathbf{y}_A - \mathbf{z}_A\|^2 \\ \text{subject to} \quad & \langle \mathbf{a}_A, \mathbf{y}_A \rangle = c \\ & \mathbf{y}_A \geq \mathbf{0}, \end{aligned}$$

where $\mathbf{y}_{\bar{A}}$ can be efficiently updated and $\mathbf{y}_A$ can be written as

$$\mathbf{y}_A = \text{Proj}_{\Delta_{\mathbf{a}_A/c}}(\mathbf{z}_A - \gamma^{-1}\mathbf{d}_A),$$

where $\text{Proj}_{\Delta_{\mathbf{a}_A}}$ denotes orthogonal projection onto the weighted simplex

$$\Delta_{c^{-1}\mathbf{a}_A} := \{\mathbf{y}_A : \langle c^{-1}\mathbf{a}_A, \mathbf{y}_A \rangle = 1, \mathbf{y}_A \geq \mathbf{0}\}$$

and can be efficiently solved using sorting-based method proposed in (Perez et al., 2022).

Remark 12. In practice, we can try the solutions from the first two cases and verify if they satisfy the conditions from case analysis. If in both cases the condition is violated, we invoke the above subroutine and compute the projection.

B.3. Direct Approximate LP Solving

In this section, we evaluate the performance of online LP solver for direct LP solving. The experiments in this section consist of two parts. First we focus on the CPU running time of our proposed method, and then we turn to more LP instances as an extension of the experiments in **Section 5**.

CPU Time Evaluation First we evaluate the CPU running time of our methods. As we already discussed, we can implement the online LP algorithm in $\mathcal{O}(\text{nnz}(\mathbf{A}))$ time, which implies our method is highly scalable. The following table summarizes the total CPU time of **Algorithm 2** with $K = 100$ under different settings of m, n and $\text{nnz}(\mathbf{A})$. Now that explicit update is independent of specific numerical values of the matrices, we use the same way of data generation as in **Section 5**.

Table 5. CPU Time evaluation of explicit update. Time given in CPU seconds.

m	n	nnz	Time	m	n	nnz	Time	m	n	nnz	Time
10^2	10^2	10^3	0.00	10^3	10^2	10^4	0.00	10^4	10^2	10^4	0.01
10^2	10^3	10^4	0.00	10^3	10^3	10^4	0.00	10^4	10^3	10^5	0.04
10^2	10^4	10^4	0.02	10^3	10^4	10^5	0.05	10^4	10^4	10^6	0.38
10^2	10^5	10^5	0.26	10^3	10^5	10^6	0.51	10^4	10^5	10^5	0.28
10^2	10^6	10^6	2.61	10^3	10^6	10^5	0.72	10^4	10^6	10^6	2.77

m	n	nnz	Time	m	n	nnz	Time
10^5	10^2	10^5	0.09	10^6	10^2	10^6	1.54
10^5	10^3	10^6	0.57	10^6	10^3	10^5	0.57
10^5	10^4	10^5	0.11	10^6	10^4	10^6	1.61
10^5	10^5	10^6	0.79	10^6	10^5	10^7	11.01
10^5	10^6	10^7	8.08	10^6	10^6	10^8	120.00

Table 5 suggests, for the same n, CPU time our method increases almost linearly with respect to $\text{nnz}(\mathbf{A})$, which verifies that the implementation from **Section B.1** is highly scalable to huge linear programs.

Real and synthetic LPs Now we switch to a more practical setting where we employ online algorithms to solve real-life instances and compare performance of online algorithms with LP solvers. Our setup is given as follows.

Testing configuration and setup

1). Dataset. Our dataset comes from three sources. **1)**. 7 instances that are LP relaxations from `MIPLIB`. **2)**. 4 MKP instances generated according to the statistics in **Table 5**. **3)**. Modified `Netlib` instances.

2). Initial point. We let online algorithm start from $\mathbf{0}$

3). Feasibility. We do not enforce feasibility of the constraints

4). Duplication. We allow a maximum of $K = 5000$ variable duplications

5). Stopping criterion. We let the algorithm stop if

$$\max\left\{\frac{\|[\mathbf{A}\mathbf{x}-\mathbf{b}]_+\|}{\|\mathbf{b}\|_1+1}, \frac{\langle \mathbf{b}^\top \mathbf{y}\rangle + \langle \mathbf{u}, [\mathbf{c}-\mathbf{A}^\top \mathbf{y}]_+\rangle - \langle \mathbf{c}, \mathbf{x}\rangle}{|\langle \mathbf{b}^\top \mathbf{y}\rangle + \langle \mathbf{u}, [\mathbf{c}-\mathbf{A}^\top \mathbf{y}]_+)| + |\langle \mathbf{c}, \mathbf{x}\rangle| + 1}\right\} \le \varepsilon = 5\times 10^{-3}$$

6). Modification of `Netlib` instances. Some `Netlib` instances do not meet assumptions for the online algorithm and thus prohibits direct solving. We modify `Netlib` instances by **1)**. Taking $\hat{b}_i = \max\{b_i, 10^{-3}\}$. **2)**. Enforcing upperbound $u_i = \min\{u_i, 10^2\}$. **3)**. Changing constraint senses into $\le$.

Table 6. Time of solving `MIPLIB` instances to an `1e-03` relative accuracy solution and comparison with `Gurobi v9.5`. Synthetic MKP instance `mkp-{i}-{j}` stands for MKP instance of 10^i rows and 10^j columns. GTime: `Gurobi` solution time.

Instance	pInf	Gap	Time	GTime	Instance	pInf	Gap	Time	GTime
`2club200v15p5scn`	1.7e-04	5.5e-02	3.44	0.32	`mkp-2-5`	4.4e-03	4.2e-03	0.03	0.44
`cdc7-4-3-2`	5.6e-05	5.0e-03	1.27	24.00	`mkp-2-6`	5.3e-05	4.9e-03	0.17	0.22
`cod105`	1.0e-06	9.6e-03	1.29	0.99	`mkp-2-7`	5.3e-05	5.0e-03	0.62	5.70
`p6b`	1.0e-04	1.2e-05	0.09	0.16	`mkp-3-5`	1.8e-03	4.9e-03	0.06	0.11
`m100n500k4r1`	9.7e-05	3.4e-03	0.12	0.08	`mkp-3-6`	1.8e-03	4.9e-03	0.29	0.54
`manna81`	1.0e-04	4.9e-03	0.62	0.08	`mkp-3-7`	4.4e-03	2.7e-03	1.38	44.70
`queens-30`	4.0e-05	4.8e-03	0.22	0.63					

Remark 13. Since our modification of `Netlib` instances destroys the original bound and coefficients, our experiment on `Netlib` is presented only for reference.

B.4. Sifting and Column Generation

This section discusses online LP in relation to sifting and column generation. We first explain the basic idea and history of the sifting algorithm. Then we identify some common challenges and solutions for sifting solvers. Finally, we show that our methods fit well with sifting and how we can make use of outputs of the online algorithms to speed up sifting.

B.4.1. Sifting: Column Generation for Linear Programming

Assume we are solving a standard-form LP

$$\begin{aligned} \max_{\mathbf{x}} \quad & \langle \mathbf{c}, \mathbf{x}\rangle \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \ge \mathbf{0} \end{aligned}$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$ and suppose $n \gg m$. By the theory of LP, we know that given an optimal basis $\mathcal{B}^*$ and the corresponding solution $\mathbf{x}^*$, $|\{i | x_i^* > 0\}| \le m \ll n$. This property tells us one important fact that most columns are redundant for achieving the optimal solution. One illustration of this property is that if we somehow manage to find a subset $\mathcal{B}^* \subseteq \mathcal{W} \subseteq [n]$ and solve a smaller LP

$$\begin{aligned} \max_{\mathbf{x}_{\mathcal{W}}} \quad & \langle \mathbf{c}_{\mathcal{W}}, \mathbf{x}_{\mathcal{W}}\rangle \\ \text{subject to} \quad & \mathbf{A}_{\mathcal{W}}\mathbf{x}_{\mathcal{W}} = \mathbf{b} \\ & \mathbf{x}_{\mathcal{W}} \ge \mathbf{0}, \end{aligned}$$

then $\langle \mathbf{c}_{\mathcal{W}}, \mathbf{x}^*_{\mathcal{W}} \rangle = \langle \mathbf{c}_{\mathcal{W}}, \mathbf{x}^*_{\mathcal{W}} \rangle + \langle \mathbf{c}_{\bar{\mathcal{W}}}, \mathbf{x}^*_{\bar{\mathcal{W}}} \rangle = \langle \mathbf{c}, \mathbf{x}^* \rangle$ and we solve the original LP at lower cost by concatenating $\mathbf{x}^* = (\mathbf{x}^*_{\mathcal{W}}, \mathbf{x}^*_{\bar{\mathcal{W}}} = \mathbf{0})$. The very intuition behind sifting is to iteratively update $\mathcal{W}$ till $\mathcal{B}^* \subseteq \mathcal{W}$, while in hope of $|\mathcal{W}| \ll [n]$.

Although we solve for the optimal primal solution $\mathbf{x}^*$, one of the most important components of sifting instead lies in the dual solution $\mathbf{y}^*$, as dual solutions tell us how to update $\mathcal{W}$ if $\mathcal{B}^* \nsubseteq \mathcal{W}$. Given optimal $(\mathbf{x}^*_{\mathcal{W}}, \mathbf{y}^*_{\mathcal{W}})$ to the aforementioned LP but $\mathcal{W}$ does not contain any optimal bases, we know that $\mathbf{c} - \mathbf{A}^\top \mathbf{y}^*_{\mathcal{W}} \nleq \mathbf{0}$, and otherwise $\mathbf{x}^* = (\mathbf{x}^*_{\mathcal{W}}; \mathbf{x}^*_{\bar{\mathcal{W}}} = \mathbf{0})$ is certificated as optimal. In other words, we know some dual infeasibility hides in $\bar{\mathcal{W}}$:

$$c_j - \langle \mathbf{a}_j, \mathbf{y}^*_{\mathcal{W}} \rangle > 0, \quad \text{for some } j \in \mathcal{I} \subseteq \bar{\mathcal{W}}.$$

and this information guides us to eliminate such dual infeasibility by updating $\mathcal{W} \leftarrow \mathcal{W} \cup \mathcal{I}$. Now that $(\mathbf{x}^*_{\mathcal{W}}; \mathbf{x}^*_{\bar{\mathcal{W}}})$ is feasible for the updated LP problem, sifting subproblems can be efficiently warm-started from the previous iterations.

Now we are ready to formalize the LP sifting procedure. Given $\mathcal{W} \subseteq [n]$, the smaller LP is called a *working problem* and the procedure finding $\mathcal{I} \subseteq \bar{\mathcal{W}}$ is called *pricing*. Sifting iteratively updates the working problem by pricing out dual infeasible columns, till $\mathcal{I} = \varnothing$.

Development of LP Sifting The idea of sifting was initially proposed in (Forrest, 1989) and formalized in a case study (Bixby et al., 1992) to solve the LP relaxation of a huge airline crew scheduling problem. One advantage of sifting is the freedom to choose solvers for the working problems, such as simplex and the interior point method. Since then, sifting has been adopted as a common framework for huge LPs and applied in different fields (Kasirzadeh et al., 2017; Moharrami et al., 2015; Yanover et al., 2006). State-of-the-art mathematical programming softwares (Berthold et al., 2018; Pedroso, 2011; Ge et al., 2022; Manual, 1987) these days implement their own sifting solvers and trigger it when the ratio n/m is large. In a word, sifting has evolved into a mature engineering technique for huge LPs.

Connection with Column Generation It's not hard to see sifting bears great resemblance to column generation (Lübbecke and Desrosiers, 2005) from early LP/MIP literature. To some extent sifting is a special case of column generation and one major difference lies in the treatment of the pricing problem. In sifting we can enumerate all the columns to find $\mathcal{I} = \{j \in \bar{\mathcal{W}} : c_j - \langle \mathbf{a}_j, \mathbf{y}^*_{\mathcal{W}} \rangle > 0\}$, while in the traditional setting of column generation (for example, cutting stock (Ben Amor and Valério de Carvalho, 2005)) we often resort to heuristics or combinatorial approaches to identify "the most infeasible column"

$$\max_{j \in \bar{\mathcal{W}}} \quad c_j - \langle \mathbf{a}_j, \mathbf{y}^*_{\mathcal{W}} \rangle.$$

Due to the deep connection between sifting and column generation, most techniques developed for column generation can be smoothly applied to sifting. In the next section, we discuss the difficulties of implementing a sifting solver and some widely known solutions from the column generation literature.

B.4.2. Difficulties and Solutions

In this section, we discuss the difficulties when implementing a sifting solver. There are three major difficulties (Lübbecke and Desrosiers, 2005) well-known as **1)**. heading-in **2)**. tailing-off and **3)**. dual oscillation. And we would like to further ascribe these difficulties to a lack of *prior knowledge*.

Lack of Prior Primal Knowledge We refer to prior primal knowledge as a measure of *likelihood* that each column participates in the optimal, or simply a feasible basis. We believe a lack of this knowledge is partially responsible for the heading-in and tailing-off effect.

Heading-in effect appears in the initialization of sifting, where we have to start from some initial working problem $\mathcal{W}$ and move on. However, if we are given no prior knowledge, how to pick $\mathcal{W}$ becomes a problem: it's unlikely that arbitrarily initialized $\mathcal{W}$ would produce a feasible, not to mention an approximately optimal solution to the original problem. Therefore most sifting implementations resort to big-M method, where the original problem is augmented by two blocks of artificial variables associated with big-M penalties.

$$\begin{aligned} \max_{\mathbf{x}_{\mathcal{W}}, \mathbf{s}_l, \mathbf{s}_u} \quad & \langle \mathbf{c}_{\mathcal{W}}, \mathbf{x}_{\mathcal{W}} \rangle - M \langle \mathbf{e}, \mathbf{s}_l \rangle - M \langle \mathbf{e}, \mathbf{s}_u \rangle \\ \text{subject to} \quad & \mathbf{A}_{\mathcal{W}} \mathbf{x}_{\mathcal{W}} - \mathbf{s}_l + \mathbf{s}_u = \mathbf{b} \\ & \mathbf{x}_{\mathcal{W}}, \mathbf{s}_l, \mathbf{s}_u \geq \mathbf{0} \end{aligned}$$

The augmented problem is equivalent to the original problem if M is sufficiently large, and it admits a trivial initial feasible solution. However, whenever $\mathbf{s}_l$ or $\mathbf{s}_u$ has an entry in the basis, $M\langle \mathbf{e}, \mathbf{s}_l\rangle + M\langle \mathbf{e}, \mathbf{s}_u\rangle$ would make the objective value from sifting far from the true approximate objective, and thus the initial sifting iterations provide little information about the original problem: we get little information until we kick $\mathbf{s}_l$ and $\mathbf{s}_u$ out of basis. This effect is known as heading-in and can be addressed by the prior knowledge about an approximate feasible primal solution.

Tailing-off refers to the phenomenon where consecutive sifting iterations bring little progress when sifting converges. In other words, at the end of sifting we keep pricing out "useless" columns that bring no actual improvement, and the true optimal basic columns stay in $\bar{\mathcal{W}}$. While several factors may contribute to tailing-off, prior knowledge about some approximate optimal solution can efficiently alleviate this effect. Namely given some approximate optimal solution $\hat{\mathbf{x}}$, we could either incorporate $\hat{\mathbf{x}}$ in the pricing rule, or simply keep all the basic columns from $\hat{\mathbf{x}}$ in the working problem.

Lack of Prior Dual Knowledge We refer to dual prior knowledge as an approximate dual optimal solution. A lack of this knowledge directly results in the notorious dual oscillation effect in sifting. For a more rigorous definition and analysis of dual oscillation, we refer the interested readers to (Lübbecke and Desrosiers, 2005; Desrosiers and Lübbecke, 2005), and in a word dual oscillation refers to the unstable behavior of the dual sequence $\{\mathbf{y}^*_{\mathcal{W}}\}$, which is also one important reason why tailing-off happens. There is vast literature attacking the issue of dual oscillation, and a systematic approach, known as dual stabilization, has been proposed and successfully applied to many applications. An important aspect of dual stabilization is to make $\{\mathbf{y}^*_{\mathcal{W}}\}$ go more smoothly by taking average

$$\mathbf{y}^*_{\mathcal{W}} \leftarrow \alpha \mathbf{y}^*_{\mathcal{W}} + (1-\alpha)\hat{\mathbf{y}}, \alpha \in (0,1]$$

where $\hat{\mathbf{y}}$ is an anchor point in the dual space obtained either before or during sifting. Some common choices of $\hat{\mathbf{y}}$ are geometric centers of the primal polytope $\{\mathbf{x} : \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$, such as analytic center (Luo and Sun, 1998) and Chebyshev center (Lee and Park, 2011). But these centers are generally too costly to be computed and less practical for really huge-scale problems.

So far we have discussed several issues sifting faces and their solutions. Overall we should find some approximate primal/dual optimal solution at low cost, and our online algorithms has a role to play here.

B.4.3. Accelerated Sifting via Online Algorithms

Finally, we are ready to present our accelerated sifting procedure using online algorithms. Recall that our method **1)**. outputs an approximate primal estimate $\hat{\mathbf{x}}$. **2)**. outputs a dual approximate solution. **3)**. runs in $\mathcal{O}(\text{nnz}(\mathbf{A}))$ time. Therefore our method can provide both primal and dual estimates at very low cost.

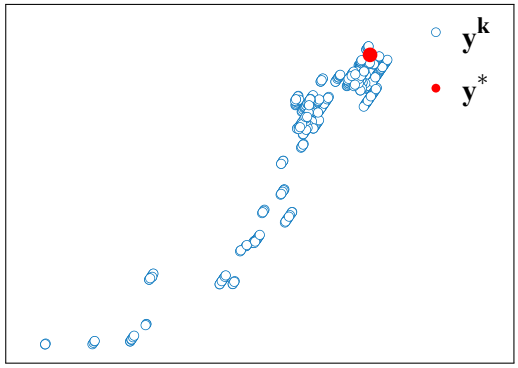

Figure 4. Convergence of the dual solution $\mathbf{y}^k$ to the optimal $\mathbf{y}^*$

Remark 14. Generally online algorithm does not address heading-in since the problem structure we target admits a trivial primal feasible solution $\mathbf{x} = \mathbf{0}$. But providing an initial good guess of optimal basis can still speed up sifting.

B.5. Integer Programming

Finally, we remark that our method can be naturally extended to binary (and integer) programming. Let $\mathbf{x}^*_{\text{Bin}}$ denote the optimal solution to the binary problem and let $\mathbf{x}^*_{\text{LP}}, \mathbf{y}^*_{\text{LP}}$ be the optimal solution to the LP relaxation. Also let $\hat{\mathbf{x}}_{\text{Bin}}$ be some integer feasible solution and $\hat{\mathbf{y}}_{\text{LP}}$ be some dual feasible solution, then the following chain of inequalities hold

$$\langle \mathbf{c}, \hat{\mathbf{x}}_{\text{Bin}} \rangle \leq \langle \mathbf{c}, \mathbf{x}^*_{\text{Bin}} \rangle \leq \langle \mathbf{c}, \mathbf{x}^*_{\text{LP}} \rangle \leq \langle \mathbf{b}, \mathbf{y}^*_{\text{LP}} \rangle + \langle \mathbf{1}, [\mathbf{c} - \mathbf{A}^\top \mathbf{y}^*_{\text{LP}}]_+ \rangle \leq \langle \mathbf{b}, \hat{\mathbf{y}}_{\text{LP}} \rangle + \langle \mathbf{1}, [\mathbf{c} - \mathbf{A}^\top \hat{\mathbf{y}}_{\text{LP}}]_+ \rangle$$

and

$$\langle \mathbf{c}, \mathbf{x}^*_{\text{LP}} \rangle - \langle \mathbf{c}, \hat{\mathbf{x}}_{\text{Bin}} \rangle \leq \langle \mathbf{b}, \hat{\mathbf{y}}_{\text{LP}} \rangle + \langle \mathbf{1}, [\mathbf{c} - \mathbf{A}^\top \hat{\mathbf{y}}_{\text{LP}}]_+ \rangle - \langle \mathbf{c}, \hat{\mathbf{x}}_{\text{Bin}} \rangle,$$

which implies $\hat{\mathbf{y}}$ provides a valid dual bound for the binary programming problem. Combined with $\hat{\mathbf{x}}$ obtained by rounding the solution from online algorithm $\hat{\mathbf{x}}_{\text{LP}}$, we can expect approximately solving a binary programming problem without resorting to branch and bound. As with general integer problems, we can split $x_j \in \{0, \ldots, U\}$ into $x_j = \sum_{k=1}^{U} x_{jk}$ and re-apply the algorithm for binary problems.

References in the Appendix

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

H. Ben Amor and J. Valério de Carvalho. Cutting stock problems. In *Column generation*, pages 131–161. Springer, 2005.

T. Berthold, J. Farmer, S. Heinz, and M. Perregaard. Parallelization of the fico xpress-optimizer. *Optimization Methods and Software*, 33(3):518–529, 2018.

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

J. Desrosiers and M. E. Lübbecke. A primer in column generation. In *Column generation*, pages 1–32. Springer, 2005.

J. Forrest. Mathematical programming with a library of optimization subroutines. In *ORSA/TIMS Joint National Meeting, New York*, 1989.

D. Ge, Q. Huangfu, Z. Wang, J. Wu, and Y. Ye. Cardinal optimizer (copt) user guide. *arXiv preprint arXiv:2208.14314*, 2022.

A. Kasirzadeh, M. Saddoune, and F. Soumis. Airline crew scheduling: models, algorithms, and data sets. *EURO Journal on Transportation and Logistics*, 6(2):111–137, 2017.

C. Lee and S. Park. Chebyshev center based column generation. *Discrete Applied Mathematics*, 159(18):2251–2265, 2011.

X. Li, C. Sun, and Y. Ye. Simple and fast algorithm for binary integer and online linear programming. *Advances in Neural Information Processing Systems*, 33:9412–9421, 2020.

M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations research*, 53(6):1007–1023, 2005.

Z.-Q. Luo and J. Sun. An analytic center based column generation algorithm for convex quadratic feasibility problems. *SIAM Journal on Optimization*, 9(1):217–235, 1998.

C. U. Manual. Ibm ilog cplex optimization studio. *Version*, 12(1987-2018):1, 1987.

H. Moharrami, M. Mahini, and G. Cocchetti. Elastoplastic analysis of plane stress/strain structures via restricted basis linear programming. *Computers & Structures*, 146:1–11, 2015.

J. P. Pedroso. Optimization with gurobi and python. *INESC Porto and Universidade do Porto,, Porto, Portugal*, 1, 2011.

G. Perez, S. Ament, C. Gomes, and M. Barlaud. Efficient projection algorithms onto the weighted l1 ball. *Artificial Intelligence*, 306:103683, 2022.

C. Yanover, T. Meltzer, Y. Weiss, K. P. Bennett, and E. Parrado-Hernández. Linear programming relaxations and belief propagation–an empirical study. *Journal of Machine Learning Research*, 7(9), 2006.

Table 7. Experiment on *modified* `Netlib` LP datasets. GTime: `Gurobi` solution time

Instance	pInf	Gap	Time	GTime	Instance	pInf	Gap	Time	GTime
25fv47	9.9e-05	4.94-03	0.45	0.02	osa-30	0.0e+00	0.00+00	0.00	0.13
80bau3b	6.4e-04	2.18-02	1.13	0.10	osa-60	0.0e+00	0.00+00	0.00	0.15
adlittle	2.8e-03	1.92-02	0.60	0.08	pds-02	0.0e+00	2.41-05	0.00	0.07
afiro	0.0e+00	4.92-03	0.01	0.07	pds-06	4.1e-05	1.10-05	0.00	0.13
agg	0.0e+00	0.00+00	0.00	0.07	pds-10	2.4e-05	5.88-05	0.00	0.14
agg2	0.0e+00	0.00+00	0.00	0.07	pds-20	1.7e-05	1.44-04	0.00	0.12
agg3	0.0e+00	0.00+00	0.00	0.07	perold	0.0e+00	1.49-02	0.69	0.08
bandm	1.6e-03	4.57-02	0.62	0.08	pilot.ja	9.2e-05	4.25-04	0.01	0.08
beaconfd	0.0e+00	0.00+00	0.00	0.07	pilot	4.9e-05	1.32-02	1.14	0.13
blend	2.8e-04	1.63-03	0.60	0.08	pilot.we	0.0e+00	3.56-05	0.00	0.07
bnl1	8.2e-04	1.28-01	0.67	0.08	pilot4	0.0e+00	1.07-02	0.66	0.08
bnl2	1.1e-04	8.20-02	0.77	0.08	pilot87	1.4e-05	6.69-03	1.55	0.14
boeing1	3.2e-04	5.32-02	0.64	0.08	pilotnov	0.0e+00	0.00+00	0.00	0.08
boeing2	6.1e-04	1.31-02	0.61	0.08	qap8	0.0e+00	0.00+00	0.00	0.07
bore3d	3.3e-05	4.98-03	0.03	0.08	qap12	0.0e+00	0.00+00	0.00	0.10
brandy	5.7e-05	1.07-03	0.00	0.08	qap15	0.0e+00	0.00+00	0.00	0.10
capri	0.0e+00	1.60-03	0.00	0.07	recipe	0.0e+00	1.23-03	0.00	0.07
cre-a	0.0e+00	0.00+00	0.00	0.07	sc50a	0.0e+00	4.92-03	0.02	0.08
cre-b	0.0e+00	0.00+00	0.00	0.11	sc50b	0.0e+00	4.69-03	0.01	0.07
cre-c	0.0e+00	0.00+00	0.00	0.07	sc105	0.0e+00	4.99-03	0.04	0.07
cre-d	0.0e+00	0.00+00	0.00	0.11	sc205	0.0e+00	4.99-03	0.34	0.07
cycle	6.0e-03	2.73-01	0.80	0.08	scagr7	2.1e-05	1.43-05	0.00	0.08
czprob	7.1e-05	4.99-03	0.11	0.10	scagr25	1.3e-05	1.91-05	0.00	0.09
d2q06c	1.3e-04	1.37-02	1.20	0.14	scfxm1	9.2e-05	2.79-03	0.01	0.08
d6cube	0.0e+00	0.00+00	0.00	0.11	scfxm2	9.8e-05	4.65-03	0.01	0.08
degen2	6.3e-04	9.83-03	0.65	0.09	scfxm3	8.0e-05	4.93-03	0.01	0.09
degen3	3.9e-04	3.62-03	1.00	0.09	scorpion	0.0e+00	0.00+00	0.00	0.07
dfl001	0.0e+00	2.83-03	0.09	0.11	scrs8	1.6e-03	1.88-01	0.65	0.08
e226	2.5e-02	1.00-02	0.63	0.09	scsd1	0.0e+00	0.00+00	0.00	0.07
etamacro	3.9e-04	5.43-03	0.63	0.08	scsd6	0.0e+00	0.00+00	0.00	0.07
fffff800	2.9e-06	4.46-03	0.54	0.09	scsd8	0.0e+00	0.00+00	0.00	0.07
finnis	0.0e+00	0.00+00	0.00	0.07	sctap1	0.0e+00	0.00+00	0.00	0.07
fit1d	8.3e-01	8.90-01	0.76	0.08	sctap2	0.0e+00	0.00+00	0.00	0.08
fit1p	0.0e+00	0.00+00	0.00	0.07	sctap3	0.0e+00	0.00+00	0.00	0.08
fit2d	4.2e-02	6.04-01	2.65	0.20	seba	0.0e+00	0.00+00	0.00	0.07
fit2p	0.0e+00	0.00+00	0.00	0.10	share1b	0.0e+00	3.22-02	0.61	0.08
forplan	8.6e-05	1.32-03	0.01	0.08	share2b	9.5e-04	3.47-01	0.60	0.08
ganges	0.0e+00	6.45-04	0.00	0.07	shell	0.0e+00	0.00+00	0.00	0.07
gfrd-pnc	0.0e+00	2.35-16	0.00	0.07	ship04l	5.8e-04	5.57-02	0.69	0.09
greenbea	1.2e-05	6.78-05	0.00	0.08	ship04s	2.5e-04	5.57-03	0.66	0.09
greenbeb	4.4e-05	8.66-05	0.00	0.09	ship08l	0.0e+00	0.00+00	0.00	0.07
grow7	0.0e+00	0.00+00	0.00	0.07	ship08s	2.4e-04	3.72-03	0.68	0.08
grow15	0.0e+00	0.00+00	0.00	0.07	ship12l	0.0e+00	0.00+00	0.00	0.08
grow22	0.0e+00	0.00+00	0.00	0.07	ship12s	1.0e-04	3.76-03	0.51	0.09
israel	2.0e-05	4.50-03	0.04	0.08	sierra	0.0e+00	1.87-01	0.72	0.09
kb2	0.0e+00	5.69-02	0.63	0.08	stair	0.0e+00	4.31-03	0.00	0.08
ken-07	2.4e-04	4.72-04	0.76	0.08	standata	0.0e+00	0.00+00	0.00	0.08
ken-11	7.8e-05	1.95-04	0.00	0.09	standgub	0.0e+00	0.00+00	0.00	0.07
ken-13	1.9e-05	6.17-05	0.00	0.14	stan.	0.0e+00	0.00+00	0.00	0.07
ken-18	1.6e-05	4.42-05	0.00	0.18	stocfor1	9.3e-03	3.47-03	0.61	0.07
lotfi	0.0e+00	3.45-02	0.62	0.08	stocfor2	9.4e-04	5.08-04	0.72	0.08
maros-r7	0.0e+00	0.00+00	0.00	0.11	stocfor3	2.2e-04	1.79-04	1.66	0.13
maros	6.3e-04	2.48-03	0.67	0.08	truss	0.0e+00	0.00+00	0.00	0.07
modszk1	0.0e+00	0.00+00	0.00	0.07	tuff	0.0e+00	0.00+00	0.00	0.08
nesm	0.0e+00	0.00+00	0.00	0.08	vtp.base	0.0e+00	0.00+00	0.00	0.07
osa-07	0.0e+00	0.00+00	0.00	0.11	wood1p	0.0e+00	0.00+00	0.00	0.10
osa-14	0.0e+00	0.00+00	0.00	0.11	woodw	0.0e+00	0.00+00	0.00	0.07