Static, Width-Independent LP Solvers via Black-box Regret Minimization

Ximing Li[1] and Liv d'Aliberti[2]

[1] *Operations Research and Financial Engineering, Princeton University*
[2] *Computer Science, Princeton University*

December 12, 2024

Abstract

This note provides a brief introduction to algorithms based on Packing-Covering Linear Programs (LPs) via Multiplicative Weight Updates (MWU), as described in [4]. To the best of our knowledge, Bhattacharya et al. introduce the first width-independent, near-linear-time LP solver whose analysis is derived from a black-box application of the regret bound for MWU. We begin by introducing and defining packing-covering dual linear programs, width-independent solvers, and the MWU protocol. Next, we restate the static, black-box Whack-a-Mole MWU algorithm for packing-covering LPs, provide concrete examples to help readers understand the MWU update, and outline a near-linear-time implementation. Following this, we reformulate the MWU algorithm as an adversarial bandit problem and establish theoretical guarantees for the Whack-a-Mole MWU algorithm. Finally, in the fourth section, we give a *width-independent* bound on running time. We conclude the note by briefly discussing the extension to the dynamic update implementation of the Whack-a-Mole MWU algorithm.

1 Introduction

Set packing and covering problems are fundamental to a diverse set of fields, including engineering, approximation theory, and economics. Moreover, approximation algorithms based on packing-covering principles are used to solve problems in facility location-allocation problems, vehicle routing, crew scheduling, switching circuit design, and capital investment decisions [3]. The covering problem aims to minimize a non-negative cost function subject to non-negative covering constraints. Conversely, the packing problem aims to maximize a non-negative profit function subject to non-negative packing constraints [2]. The problems can be mathematically represented as follows:

$$\begin{array}{ll} \text{Covering LP:} & \min_{x\in\mathbb{R}^n_{\geq 0}} \left\{c^\top x \mid Ax \geq b\right\}, \\ \text{Packing LP:} & \max_{y\in\mathbb{R}^m_{\geq 0}} \left\{b^\top y \mid A^\top y \leq c\right\}. \end{array} \tag{1}$$

Where matrix coefficients $A \in \mathbb{R}^{m\times n}_{\geq 0}, b \in \mathbb{R}^m_{\geq 0}, c \in \mathbb{R}^n_{\geq 0}$ are all non-negative. The Covering and Packing linear programs (LPs) are duals to each other with the primal-dual optimum denoted as OPT. Since precisely solving the LP is challenging, particularly for large and/or very wide matrices, we focus on computing an approximate solutions efficiently.

Definition 1 (Approximate solution)**.** *For any $\epsilon > 0$ and vector $x \in \mathbb{R}^n_{\geq 0}$, we call it as an ϵ-approximation for the Covering LP if $Ax \geq b$ and $c^\top x \leq (1+\epsilon)$ OPT. Similarly, any vector $y \in \mathbb{R}^m_{\geq 0}$ is called as an ϵ-approximation for the Packing LP if $A^\top y \leq c$ and $b^\top y \geq \mathrm{OPT}/(1+\epsilon)$.*

1.1 Multiplicative Weight Update Framework

The Multiplicative Weight Update (MWU) framework is a widely adopted optimization method with diverse applications, including as an iterative solver for Covering and Packing LPs. At a high level, MWU-based solvers for the Covering and Packing LPs follows two approaches:

1. **Dual-Update Approach**: The dual solution y (corresponding to the Packing LP) is updated additively, while the primal solution x (corresponding to the covering LP) is maintained multiplicatively as an indicator variable.

2. **Primal-Update Approach**: The primal solution x (corresponding to the covering LP) is updated multiplicatively.

Width-Independent. In MWU-type algorithms, an important parameter is the *width*, defined as λOPT, where λ represents the largest entry in the coefficient matrix. The running time of many existing MWU-based solvers depends heavily upon the width λ. For instance, [7] achieves a time complexity of approximately $\tilde{O}\left(N\left(\frac{\lambda\mathrm{OPT}}{\epsilon}\right)^2\right)$, while [1], improves this bound to $\tilde{O}\left(N\frac{\lambda\mathrm{OPT}}{\epsilon^2}\right)$. Furthermore, the latter work provides a theoretical guarantee for LP solvers through a black-box application of the regret bound for MWU.

However, in practice, a large width λ leads to a slow convergence rate and necessitates a heavy computational load. This limitation has spurred interest in the development of width-independent solvers [5, 8]. These fast algorithms enjoy a $\tilde{O}(N/\operatorname{poly}(\epsilon))$ running time that is independent (or only log-dependent) on λOPT. By removing the dependence upon width, fast width-independent solvers offer a significant improvement in efficiency and scalability.

Black-Box Manner. Although the algorithms described above all build on the same MWU framework, each requires a separate and fine-tuned analysis resembling a proof of the regret bound of MWU. But can we achieve a *width-independent* Packing-Covering LPs solver whose guarantee directly follows from the regret bound of MWU in a *blackbox* manner? [4] provides a positive answer to this question, which serves as the central focus of our note. We then extend beyond the theoretical analysis of [4] by reformulating the problem as an adversarial bandit learning scenario, and we use this formulation to carry out a black-box MWU regret bound.

1.2 Organization

This note examines the Packing-Covering LP framework introduced by [4]. In Section 2, we provide a detailed overview of the static whack-a-mole LP solver based on the MWU method, supplemented with two detailed examples to illustrate the core idea. Section 3 discusses the connection between the algorithm as the MWU policy update in an adversarial bandit learning scenario. Based on this formulation, we establish a theoretic guarantee by a black-box application of the MWU regret bound. Section 4 provides a width-independent running time and explains the underlying rationale. Section 5 summarizes the note and gives suggestions on future readings.

2 Algorithm Description

We begin by presenting the whack-a-mole MWU algorithm basic template in the static setting. Then, to help readers understand, we include two examples of the algorithm over small-scale packing-covering LP problems. We end the section with a discussion on how to implement the algorithm in near linear time.

2.1 Static Whack-a-Mole MWU

Algorithm 1, described below, gives the most basic description of a solution to the following problem statement for a static setting:

Problem Statement: *Given a matrix* $C \in [0, \lambda]^{m \times n}$ *where* $\lambda > 0$*, either return a vector* $x \in \mathbb{R}^n_{\geq 0}$ *with* $\mathbb{1}^T x \leq 1 + \Theta(\epsilon)$ *and* $Cx \geq (1 - \Theta(\epsilon)) \cdot \mathbb{1}$ *or return a vector* $y \in \mathbb{R}^m_{\geq 0}$ *with* $\mathbb{1}^T y \geq 1 - \Theta(\epsilon)$ *and* $C^T y \leq (1 + \Theta(\epsilon)) \cdot \mathbb{1}$.

Where we either return an approximately feasible solution to the **covering LP** with objective $\leq 1 + \Theta(\epsilon)$ or return an approximately feasible solution to the dual **packing LP** with objective $\geq 1 - \Theta(\epsilon)$. Note this problem is a special case of the Packing-Covering LPs in (1) with $A = C, b = \mathbb{1}^m$, and $c = \mathbb{1}^n$, and Appendix B shows that general Packing-Covering LPs can be reduced to this special case.

The algorithm maintains a vector $\hat{x} \in \mathbb{R}^n_{\geq 0}$, where $\hat{x}_j$ denotes the *weight* associated with the variable $j \in [n]$ from the covering LP and the normalized

Algorithm 1 Static Whack-a-Mole MWU Basic Template

Define $T \leftarrow \frac{\lambda \ln(n)}{\epsilon^2}$, and two vectors $\hat{x}, x^1 \in \mathbb{R}^n_{\geq 0}$, where $\hat{x}^1 \leftarrow \mathbb{1}$ and $x^1 \leftarrow \frac{\hat{x}^1}{\|\hat{x}\|_1}$
for $t = 1$ to T **do**
  if $\forall i \in [m], (C \cdot x^t)_i \geq 1 - \epsilon$ **then**
    Return (x^t, NULL).
  else
    $\hat{x}^{t+1} \leftarrow \text{WHACK}(i_t, \hat{x}^t)$.
    $x^{t+1} \leftarrow \frac{\hat{x}^{t+1}}{\|\hat{x}^t + 1\|_1}$.
    Let $y^t \in \Delta^m$ be the vector where $(y^t)_{i_t} = 1$ and $(y^t)_i = 0, \forall i \in [m] \backslash \{i_t\}$.
  end if
end for
$y \leftarrow (1/T) \cdot \sum_{t=1}^{T} y^t$.
Return (NULL, y).

Algorithm 2 WHACK$(i_t, \hat{x}^t)$

for $j \in [n]$ **do**
  $\hat{z}_j \leftarrow (1 + \epsilon \cdot \frac{C_{ij}}{\lambda}) \cdot \hat{x}_j$.
end for
Return $\hat{z}$

vector $x := \hat{x}/\|\hat{x}\|_1$. This ensures that $\mathbb{1}^T \cdot x = 1$. The algorithm will run in $T = \lambda \ln(n)/\epsilon^2$ iterations, where $0 < \epsilon < 1/2$. If we let $\hat{x}^t$ and x^t reflect the status of $\hat{x}$ and x at the start of iteration $t \in [T]$, then before a given iteration the algorithm will do one of the two following cases:

- Case (1): Observe that $Cx^t \geq (1 - \epsilon) \cdot \mathbb{1}$ and return (x^t, NULL). In this case, $x^t \in \mathbb{R}^n_{\geq 0}$ is an approximately feasible solution to the covering LP, with objective $\mathbb{1}^T x^t = 1$.
- Case (2): A covering constraint $i_t \in [m]$ with $(Cx^t)_{i_t}$ is identified. The constraint is then WHACK-ed into place by setting $\hat{x} \leftarrow (1 + \epsilon \cdot \frac{C_{i_t j}}{\lambda}) \cdot \hat{x}_j$ for all $j \in [n]$, returning an updated normalized vector x.

WHACK-ing a violated covering constraint makes progress towards making the solution x^t feasible for the covering LP. We let $y^t \in \Delta^m$ denote the *indicator* vector for the covering constraint $i_t \in [m]$ that gets whacked. After T iterations, the algorithm returns (NULL, y), where y is the average of vectors $y^1, \cdots, y^T$, and $\mathbb{1}^T y = 1$. Per the following informal lemma and theorem, y is the approximately feasible solution to the dual packing LP. We will discuss the theoretical analysis in details later in Section 3 and 4.

Lemma 1 *Suppose that Algorithm 1 returns $(NULL, y)$. Then $C^T y \leq (1+4\epsilon) \cdot \mathbb{1}$.*
Theorem 1 *Algorithm 1 either returns an $x^t \in \mathbb{R}^n_{\geq 0}$ with $\mathbb{1}^T x^t = 1$ and $Cx^t \geq$*

$(1-\epsilon)\cdot \mathbb{1}$ or it returns a $y \in \mathbb{R}^m_{\geq 0}$ with $\mathbb{1}^T y = 1$ and $C^T y \leq (1+4\epsilon)\cdot \mathbb{1}$.

2.2 Small Scale Examples

To gain some intuition into both **Theorem 1** and **Algorithm 1**, before we analyze in later sections, we consider the following **Example 1** of a very small toy **covering** problem, where given a matrix

$$C = \begin{bmatrix} 0.8 & 0.1 \end{bmatrix}, \text{ and let } \lambda = 0.8, \epsilon = 0.5 \tag{2}$$

We are hoping to find $\vec{x} \in \mathbb{R}^2_{\geq 0}$ that minimizes $\mathbb{1}^T \vec{x} = x_1 + x_2 \leq 1.5$ subject to:

$$Cx = \begin{bmatrix} 0.8 & 0.1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \geq 0.5 \tag{3}$$

The **packing** problem dual is then, to find $\vec{y} \in \mathbb{R}^1_{\geq 0}$ that maximizes $\mathbb{1}^t \vec{y} = y_1 \geq 0.5$ subject to:

$$C^T y = \begin{bmatrix} 0.8y_1 \\ 0.1y_1 \end{bmatrix} \leq 1.5 \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{4}$$

Now, to find our $\vec{x}$ and $\vec{y}$ vectors, we will employ our Whack-a-Mole MWU algorithm.

Initialization:

- We start with $\hat{x}^1 = \mathbb{1} = (1,1)$. Thus, $x^1 \to \frac{\hat{x}^1}{\|\hat{x}^1\|_1} = (1/2, 1/2)$.
- Now, we check feasibility for the covering constraints with x^1:

$$(Cx^1)_1 = 0.8 \cdot \frac{1}{2} + 0.1 \cdot \frac{1}{2} = 0.45 \tag{5}$$

- With our $\epsilon = 0.5$, clearly we have not satisfied the constraint

$$(Cx^1)_1 = \begin{bmatrix} 0.45 \end{bmatrix} \ngeq \begin{bmatrix} 0.5 \end{bmatrix} \tag{6}$$

  So, we are clearly not feasible.

- So, we find a covering constraint such that $(C \cdot \vec{x}^1)_{i_1} < 1$. For our example, we will consider row 1 as i_1 and proceed to WHACK it.

WHACK-ing:

- For the chosen constraint $i_1 = 1$, we update $\hat{x}$ using Algorithm 2, our WHACK algorithm, such that

$$\hat{x}^2 \leftarrow (1 + 0.1\frac{\begin{bmatrix} 0.8 & 0.1 \end{bmatrix}}{0.8}) \cdot \mathbb{1} = \begin{bmatrix} (1 + 0.5 \cdot \frac{0.8}{0.8}) \cdot 1 \\ (1 + 0.5 \cdot \frac{0.1}{0.8}) \cdot 1 \end{bmatrix} \approx \begin{bmatrix} 1.5 \\ 1.0625 \end{bmatrix} \tag{7}$$

- Now, we normalize, such that

$$x^2 \leftarrow \frac{\hat{x}^2}{\|\hat{x}^2\|_1} = \begin{bmatrix} \frac{1.5}{2.5625} & \frac{1.0625}{2.5625} \end{bmatrix} \approx (0.585, 0.415) \tag{8}$$

Recheck Constraints:

- We once again move through checking our constraints, i.e.

$$(Cx^1)_1 = 0.8 \cdot 0.585 + 0.1 \cdot 0.415 = 0.5095 \tag{9}$$

- Now, $0.5095 > 0.5$. So, our covering constraint is satisfied. In just one iteration of WHACK-ing, we have found satisfaction of all constraints, so we return $\vec{x} = (0.585, 0.415)$ as an approximate covering solution with $\mathbb{1}^T x^2 \approx 1$

This toy is clearly quite small-scale, and we easily satisfy the covering constraint in 1 iteration. Let's instead consider a more challenging (but similarly small) problem where the covering constraint cannot be satisfied within the allotted T iterations (and thus we find the dual packing LP) and highlights the *width-independent* nature of this algorithm. Consider **Example 2**:

$$C = \begin{bmatrix} 1 & 0.5 & 1 & 0.3 & 0 \\ 0.2 & 1 & 0.7 & 0 & 1 \end{bmatrix}, \text{ and let } \lambda = 1, \epsilon = 0.1 \tag{10}$$

For the **covering** problem, we seek $\vec{x} = (x_1, x_2, x_3, x_4, x_5) \in \mathbb{R}^5_{\geq 0}$ that minimizes $\vec{x}$ such that

$$\begin{aligned} 1x_1 + 0.5x_2 + 1x_3 + 0.3x_4 + 0x_5 \geq 0.9 \\ 0.2x_1 + 1x_2 + 0.7x_3 + 0x_4 + 1x_5 \geq 0.9 \end{aligned} \tag{11}$$

The **packing** problem dual is to find $\vec{y} = (y_1, y_2) \in \mathbb{R}^2_{\geq 0}$ that maximizes $\vec{y}$ such that

$$C^T y = \begin{bmatrix} 1 & 0.2 \\ 0.5 & 1 \\ 1 & 0.7 \\ 0.3 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \leq \begin{bmatrix} 0.9 \\ 0.9 \\ 0.9 \\ 0.9 \\ 0.9 \end{bmatrix} \tag{12}$$

As we increase the number of variables (columns), we see that the problem gets "wider", but the MWU-style algorithm described under Algorithm 1 scales effectively with additional dimensionality. In later sections, we will show that the complexity grows only as $\lambda \ln(n)/\epsilon^2$ and the approximation guarantees remain the same: the algorithm will either find a near-feasible covering solution or a near-feasible packing solution.

We initialize with $\vec{x} = (1/5, 1/5, 1/5, 1/5, 1/5)$ following the process demonstrated in example 1, and we check coverage, i.e.

$$\begin{aligned} 1 \cdot 0.2 + 0.5 \cdot 0.2 + 1 \cdot 0.2 + 0.3 \cdot 0.2 + 0 \cdot 0.2 = 0.56 \not\geq 0.9 \\ 0.2 \cdot 0.2 + 1 \cdot 0.2 + 0.7 \cdot 0.2 + 0 \cdot 0.2 + 1 \cdot 0.2 = 0.58 \not\geq 0.9 \end{aligned} \tag{13}$$

So, our constraints are not satisfied. So, we set off to WHACK our constraints

$$\begin{aligned} \hat{x} \leftarrow (1 + 0.1 \begin{bmatrix} 1 & 0.5 & 1 & 0.3 & 0 \end{bmatrix}) \cdot \mathbb{1} \\ \hat{x} \leftarrow (1.1, 1.05, 1.1, 1.03, 1) \\ x = \frac{\hat{x}}{5.28} \approx (0.2083, 0.1989, 0.2093, 0.1951, 0.1894) \end{aligned} \tag{14}$$

Now, when we recheck our constraints,

$$\begin{aligned} 1 \cdot 0.2083 + 0.5 \cdot 0.1989 + 1 \cdot 0.2093 + 0.3 \cdot 0.1951 + 0 \cdot 0.1894 = 0.57558 \not\geq 0.9 \\ 0.2 \cdot 0.2083 + 1 \cdot 0.1989 + 0.7 \cdot 0.2093 + 0 \cdot 0.1951 + 1 \cdot 0.1894 = 0.57647 \not\geq 0.9 \end{aligned} \tag{15}$$

We have moved constraint 1 slightly towards our goal of 0.9, while moving constraint 2 slightly further away from our given goal. While progress is made, this problem does not converge within T iterations, and instead returns to us the dual $\vec{y} = (0.4906, 0.5093)$, satisfying the **packing dual**

$$C^T y = \begin{bmatrix} 1 & 0.2 \\ 0.5 & 1 \\ 1 & 0.7 \\ 0.3 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0.4906 \\ 0.5093 \end{bmatrix} \approx \begin{bmatrix} 0.5925 \\ 0.7546 \\ 0.8471 \\ 0.14718 \\ 0.6093 \end{bmatrix} \leq \begin{bmatrix} 0.9 \\ 0.9 \\ 0.9 \\ 0.9 \\ 0.9 \end{bmatrix} \tag{16}$$

For more formal details on the theoretic guarantees for template, see Section 3 on Theoretic Analysis. For more details on the width-independent running time, see Section 4.

2.3 Near Linear Time Implementation

We conclude this section by highlighting that this algorithm can be run in near-linear time. To accomplish this, the whack-a-mole MWU algorithm is split into *phases*, allowing us to only consider each constraint at most once, whacking it repeatedly on a single run until satisfied. Consider the following reformulation in Algorithm 3.

In each phase, weights are set as $W \leftarrow \|\hat{x}^t\|_1$ and remain constant throughout the iteration. Since $\|\hat{x}^t\|_1 \leq (1 - \epsilon/2)^{-1}$, $\frac{\hat{x}^t}{W}$ remains a good estimation for $x^t := \frac{\hat{x}^t}{\|\hat{x}^t\|_1}$. Now, using this new near linear time implementation, we scan through all covering constraints in a single phase. If a constraint is violated, we apply the ENFORCE algorithm which calls as a subroutine STEP-SIZE. Together, these algorithms figure out how many times the algorithm needs to be WHACK-ed, and does so until the constraint is satisfied, otherwise the dual is provided.

Algorithm 3 Linear Time Implementation

$\hat{x}^1 \leftarrow \mathbb{1}, t \leftarrow 1$, and $T \leftarrow \lambda \ln(n)/\epsilon^2$.
for $t = 1$ to T **do**
  "Start" : $W \leftarrow \|\hat{x}\|_1$.
  for $i \in [m]$ **do**
    if $(C \cdot \frac{\hat{x}^t}{W})_i < 1 - \epsilon/2$, **then**
      $\delta \leftarrow$ ENFORCE$(i, t, \hat{x}^t, W)$.
      $t \leftarrow t + \delta$
      if $t = T$, **then**
        Return (NULL, y), where $y := (1/T) \cdot \sum_{t'=1}^{T} y^{t'}$.
        Terminate the loop.
      end if
      if $\|\hat{x}^t\|_1 > (1 - \epsilon/2)^{-1} \cdot W$, **then**
        Go to "Start". (Initiate a new phase).
      end if
    end if
  end for
  Terminate the Loop and Return (x^t, NULL), where $x^t := \frac{\hat{x}^t}{\|\hat{x}^t\|_1}$
end for

Algorithm 4 ENFORCE$(i, t, \hat{x}^t, W)$

$\delta \leftarrow$ STEP-SIZE$(i, t, \hat{x}^t, W)$.
for $t' = t$ to $(t + \delta - 1)$, **do**
  $\hat{x}^{t'+1} \leftarrow$ WHACK$(i, \hat{x}^{t'})$
  $i'_t \leftarrow i$
  Let $y^{t'} \in \Delta^m$ be the vector where $(y^{t'})_i = 1$.
  Let $(y^{t'})_{i'} = 0$ for all $i' \in [m] \backslash \{i\}$.
end for
Return δ

At the end of the full scan, W still accurately estimates the weight $\|\hat{x}^t\|_1$ and $t < T$, then we get back the primal covering LP. In Section 3, we will analyze and guarantee that the template provided in Section 2.1 matches the newly defined near-linear time implementation within this section.

Algorithm 5 STEP-SIZE$(i, t, \hat{x}^t, W)$

for Every integer $k \geq 1$ **do**
  Let $z^k \in \mathbb{R}^n_{\geq 0}$ be such that $(z^k)_j = (1 + \epsilon \cdot \frac{C_{ij}}{\lambda})^k \cdot (\hat{x}^t)_j$ for $j \in [n]$.
end for
if $(C \cdot \frac{z^{T-t}}{W})_i < 1$, **then**
  $\delta \leftarrow T - t$
else
  Use binary search, compute the smallest integer $\delta \in [T - t]$ s.t. $(C \cdot \frac{z^\delta}{W})_i \geq 1$.
end if
Return δ

3 Theoretic Analysis in a Black-Box Manner

In this section, we establish a theoretical guarantee for the Algorithm 3 by presenting a *reformulation* that realizes the algorithm as an adversarial bandit problem with MWU policy updates. This reformulation elegantly bridges the dual optimality of the packing-covering LP with the optimal policy in the adversarial bandit learning scenario.

3.1 Reformulation as an Adversarial Bandit

We show that the template algorithm (Algorithm 1) can be realized by running the MWU algorithm as an adversarial bandit, enabling a *black-box* application of bounds on MWU algorithm.

Adversarial Bandit. We start by recalling the adversarial bandit setting seen in [6, Chapter 11]. Let $n > 1$ be the number of arms. An n-armed adversarial bandit is a sequence of reward vectors $\{r^t\}_{t=1}^T$, where $r^t \in [0, 1]^n$. In each round t, the agent selects a probability distribution P_t over the $[n]$ actions. Subsequently, action $A_t \in [n]$ is drawn from P_t, and the agent receives reward $r^t_{A_t}$. The environment then reveals the reward vector r^t.

A policy, in this context, is the function $\pi : ([n] \times [0, 1])^* \to P_{n-1}$ that maps the history of past actions and rewards to a probability distributions over n actions. The performance of a policy π in environment x is evaluated based on the expected regret, defined as the expected loss of the policy relative to the best fixed action in hindsight. The MWU updates the policy as

$$\pi_{t+1}(j) \leftarrow (1 + \epsilon r^t_j)\pi_t(j), \forall j \in [n]$$
$$\pi_{t+1} \leftarrow \frac{\pi_{t+1}}{\|\pi_{t+1}\|}.$$

where ϵ is a small positive parameter. It is known that MWU policy-update is near-optimal against the best fixed action in hindsight [6].

Proposition 1 (Regret bound of the MWU algorithm)**.** *Consider an agent following policy π_t at the t-th round for each t, then we have*

$$\sum_{t=1}^{T} \mathbb{E}_{A_t \sim \pi_t}[r^t_{A_t}] \geq (1-\epsilon) \max_{j \in [n]} \sum_{t=1}^{T} r^t_j - \frac{\log n}{\epsilon}.$$

Reformulation. The template (Algorithm 1) can be reformulated as playing an adversarial bandit following the MWU update:

- At the t-th round, the agent selects an action $j \in [n]$ following the policy $\frac{x^t}{\|x^t\|_1}$.
- The agent then gets the reward $\frac{1}{\lambda} C_{i_t,j}$.
- The environment reveals the reward function and the agent updates the policy following (17).

One can check that this formulation is corresponded to setting $r^t := \frac{1}{\lambda} C_{i_t}$ where C_{i_t} is the i_t-th row, $i_t \in [m]$ of the matrix C and λ is the maximum entry of C. It is also clear that $r^t \in [0,1]^n$. The whacking and normalization steps corresponds to the aforementioned MWU update in (17).

Optimality. Our formulation connects the optimal policy in the bandit scenario with the optimality in Packing LP. For any $t \geq 0$, the definition of y^t shows that the entry $C_{i_t,j}$ becomes $(C^T y^t)_j$ for any j. The total reward of the best fixed action in hindsight becomes

$$\max_{j \in [n]} \frac{1}{T\lambda} \sum_{t=1}^{T} (C^T y^t)_j = \frac{1}{\lambda} \left\| C^T y \right\|_\infty, \tag{17}$$

where $y = \frac{1}{T} \cdot \sum_{t=1}^{T} y^t$ (the output of Algorithm 3 at the T-th round).

3.2 Theoretical Guarantees for the Template Algorithm

Theorem 3.1 (Theoretic guarantee for Algorithm 1)**.** *The output of the template (Algorithm 1) is an approximate solution to the* ***packing*** *LP.*

Proof. With the formulation from Section 3.1, we can apply the bound for MWU (Proposition 1) in a **blackbox** manner. Directly applying Proposition 1 and (17) shows that

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{j=1}^{n} \frac{1}{\lambda} C_{i_t,j} x^t_j \geq \frac{1-\epsilon}{\lambda} \left\| C^T y \right\|_\infty - \frac{\log n}{\epsilon T} = \frac{1-\epsilon}{\lambda} \left\| C^T y \right\|_\infty - \frac{\epsilon}{\lambda}. \tag{18}$$

Since the i_t-th constraint is necessarily $(C \cdot x^t) < 1 - \epsilon$, otherwise the algorithm would conclude if $(C \cdot x^t)_i \geq 1 - \epsilon$ before the t-th step, we have

$$\sum_{j=1}^{n} C_{i_t,j} x_j^t = (Cx^t)_{i_t} \leq 1,$$

for any t. Combining this with (18) shows that $C^T y \leq \epsilon/(1-\epsilon)\mathbb{1} \leq 2\epsilon\mathbb{1}$, for small enough ϵ. Meanwhile, the definition of y automatically ensures that $\mathbb{1}^T y = 1$. These together verify that the output y of Algorithm 1 is an approximate solution to the packing LP defined in Definition 1. □

3.3 Theoretic Guarantee for Algorithm 3

Moving forward, we refer to each call to the WHACK-ing algorithm as a *round* and each update W as the end of a *phase*. We begin with Lemma 3.1, verifying that Algorithm 3 implements the basic template, (Algorithm 1).

Lemma 3.1. *Algorithm 3 implements the template (Algorithm 1).*

Proof. We show that for a given phase of the ENFORCE protocol, the enforcing of a given constraint $i \in [m]$ in rounds $t' \in [t, t+\delta-1]$ is consistent with the WHACK-ing of the template (Algorithm 1). More precisely, for a constraint $i \in [m]$ that is enforced at the start of a round $t \in [T]$ in the protocol ENFORCE $(i, t, \hat{x}^t, W)$, the WHACK-ing Algorithm 3 shows that

$$\left(C \cdot \frac{\hat{x}^{t'}}{W}\right)_i < 1 \text{ for all } t \leq t' \leq t+\delta-1. \tag{19}$$

Since W can only increase with time, it follows that $\left\|\hat{x}^{t'}\right\|_1 \geq W$ for all $t' \geq t$. This implies that

$$\left(C \cdot x^{t'}\right)_i < 1 \text{ for all } t \leq t' \leq t+\delta-1 \tag{20}$$

where $x^{t'} := \frac{\hat{x}^{t'}}{\left\|\hat{x}^{t'}\right\|_1}$. In other words, WHACK-ing the i-th constraint for round $[t, t+\delta-1]$ is consistent with basic template (Algorithm 1). □

Lemma 3.1 establishes that if Algorithm 3 ends at time T and returns the vector y, then y is an approximate solution to the covering LP outlined in Section 1. If Algorithm 3 returns the vector x, the following result ensures that x is an approximate solution to the packing LP.

Lemma 3.2. *If Algorithm 3 returns an x, then it is is an approximate solution to the packing LP.*

Proof sketch. Consider the last phase starts with (x', W') that ends with (x, W). The stopping condition implies that W is close to W'. Meanwhile, the ENFORCE protocol ensures that $\frac{x}{W'}$ satisfies the constraint $C^T \frac{x}{W'} \geq 1$. Together, this leads to $\frac{x}{W}$ approximately satisfies the constraint. See Appendix A for a complete proof.

4 Width-independent Running Time

In this section, we establish a log-width running time for Algorithm 3 and explain the underlying rationale. By cleverly incorporating the well-known *Binary Exponentiation* method, we are able to reduce the computational load and achieve a more efficient implementation. Our main result is as follows:

Theorem 4.1 (Running time for Algorithm 3)**.** *Algorithm 3 can be implemented in* $\mathcal{O}\left(N\frac{\log n}{\epsilon^2}(\log\frac{\lambda n}{\epsilon})^2\right)$*. The width* λ *is the maximum entry of the matrix* C *and* N *is the number of nonzero elements in the matrix* C*.*

4.1 Implementation based on Binary Exponentiation

The key to achieving a log-width running time is the following well-known result on computing the exponentiation.

Proposition 2 (Binary Exponentiation)**.** *For a fixed real number* x*, there exists a* $\mathcal{O}(\log n)$*-time algorithm that computes* x^n*.*

Compared with the primitive algorithm that multiplies x by n times and has the running time as $\mathcal{O}(n)$, the Binary Exponentiation algorithm is faster, with a $\mathcal{O}(\log n)$ running time. We now explain how the Binary Exponentiation help us reach a width-independent running time.

Example 1. *Consider a toy example that the algorithm ends after 4 phases with the outputs as* (x^1, x^2, x^3, x^4)*, and each phase consist of* ***consecutively*** *whacking the first constraint for* λ *times. Thus, for the* i*-phase for any* $i \in [4]$*, instead of computing*

$$(1+\frac{\epsilon}{\lambda}C_{1,j})^k x_j^i, \text{ for } k = 1 \sim \lambda,\ j = 1 \sim n,$$

which takes $\mathcal{O}(m\lambda) -$ *time, we would only need to compute*

$$(1+\frac{\epsilon}{\lambda}C_{1,j})^\lambda x_j^i,\ j = 1 \sim n.$$

Applying the Binary Exponentiation algorithm (Proposition 2) for each phase yields the running time of $\mathcal{O}(m\log\lambda)$*.*

Example 1 shows that by integrating multiple rounds into a phase, we can apply Binary Exponentiation algorithm to each phase to reduce the computational load. This intuition is formalized as follows.

Lemma 4.1 (Total number of phases)**.** *There are no greater than* $\mathcal{O}\left(\log n/\epsilon^2\right)$ *many phases in Algorithm 3.*

Proof Sketch. Note that we restart a new phase only when $\|x\|_1$ increases by a factor of $(1-\epsilon/2)^{-1}$. Meanwhile, for each round, the $\|x\|_1$ increase by a factor of (no greater than) $(1+\epsilon/\lambda)$. Since there are T rounds in total, the number of phases is bounded by $\log_{(1-\epsilon/2)^{-1}}(1+\epsilon/\lambda)^T = \mathcal{O}\left(\log n/\epsilon^2\right)$. See Appendix A for a detailed proof.

Lemma 4.2 (Computational time for each phases)**.** *For any phase and any $i \in [m]$, it take no greater than $\mathcal{O}\left(N_i(\log T)^2\right)$-times to enforce the i-th constraint in this phase. Here N_i is the number of non-zero entries in row i of the matrix C.*

Proof. The binary search for $\delta \in [0, T]$ can be done in $\log T$-times. To implement WHACK-ing the i-th constraint δ times, we only need to compute $(1+C_{i_j}/\lambda)^\delta x_j$ for each j. This can be done in $N_i \log T$ times using the Binary Exponentiation. □

Theorem 4.1 directly follows from Lemma 4.1 and 4.2.

5 Conclusion and Discussion

This note studies the packing-covering LP solver by [4]. We discuss the connection between the LP solver and MWU policy update in an adversarial bandit learning scenario, which enables a black-box-type theoretic guarantee. We also describe the algorithm's width-independent running time and explain the underlying rationale.

Extension to the Dynamic Update. Algorithm 3 above can be seamlessly extended to the dynamic setting where the matrix C undergoes a sequence of restricting updates. Here each restricting update decreases some entries of the matrix C. The theoretical guarantee for the accuracy and running time is similar to that of the static setting. We refer the readers to [4] for more details.

References

[1] Arora, S., Hazan, E., and Kale, S. (2012). The multiplicative weights update method: a meta-algorithm and applications. *Theory of computing*, 8(1):121–164.

[2] Azar, Y., Buchbinder, N., Chan, T. H., Chen, S., Cohen, I. R., Gupta, A., Huang, Z., Kang, N., Nagarajan, V., Naor, J., et al. (2016). Online algorithms for covering and packing problems with convex objectives. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 148–157. IEEE.

[3] Balas, E. (1982). Tutorial paper x: A class of location, distribution and scheduling problems: modelling and solution methods. *JORBEL-Belgian Journal of Operations Research, Statistics, and Computer Science*, 22(2):36–69.

[4] Bhattacharya, S., Kiss, P., and Saranurak, T. (2023). Dynamic algorithms for packing-covering lps via multiplicative weight updates. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–47. SIAM.

[5] Chekuri, C. and Quanrud, K. (2018). Randomized mwu for positive lps. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 358–377. SIAM.

[6] Lattimore, T. and Szepesvári, C. (2020). *Bandit algorithms*. Cambridge University Press.

[7] Plotkin, S. A., Shmoys, D. B., and Tardos, É. (1995). Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20(2):257–301.

[8] Quanrud, K. (2020). Nearly linear time approximations for mixed packing and covering problems without data structures or randomization. In *Symposium on Simplicity in Algorithms*, pages 69–80. SIAM.

A Detailed Proof

A.1 Proof for Lemma 4.1

Proof. Consider any round $t \in [T]$ in ENFORCE$(i, t, \hat{x}_t, W)$ we have

$$\left\|\hat{x}^{t+1}\right\|_1 - \left\|\hat{x}^t\right\|_1 = \sum_{j\in[n]} \left(\left(\hat{x}^{t+1}\right)_j - \left(\hat{x}^t\right)_j\right) = \sum_{j\in[n]} \left(\hat{x}^t\right)_j \cdot \left(\epsilon \cdot \frac{C_{i_t,j}}{\lambda}\right).$$

since $\left(Cx^t\right)_{i_t} < 1$ for $x^t := \hat{x}^t / \left\|\hat{x}^t\right\|_1$, the increment can be further bounded by

$$\left\|\hat{x}^{t+1}\right\|_1 - \left\|\hat{x}^t\right\|_1 \leq \frac{\epsilon}{\lambda} \cdot \left(C\hat{x}^t\right)_{i_t} < \frac{\epsilon}{\lambda} \cdot \left\|\hat{x}^t\right\|_1.$$

Since there are T rounds in total and $\left\|\hat{x}^0\right\|_1 = n$, the L_1 norm can be bounded as $\left\|\hat{x}^t\right\|_1 \leq \left(1 + \frac{\epsilon}{\lambda}\right)^T \cdot \left\|\hat{x}^0\right\|_1 \leq n^{(1/\epsilon)}$ for all $t \in [T]$.

Meanwhile, we start a new phase only when $\left\|\hat{x}^t\right\|$ increases by a multiplicative factor of $(1-\epsilon/2)^{-1}$. Thus, previous discussion shows that the number of phases is at most $O\left(\log_{(1-\epsilon/2)^{-1}} n^{(1/\epsilon)}\right) = O\left(\frac{\log n}{\epsilon^2}\right)$. □

A.2 Proof for Lemma 3.2

Proof. We consider the very last phase, which spans from round t' to round t'', where $t' < t''$. Let W be the value of $\left\|\hat{x}^t\right\|_1$ at the start of this phase. Fix any constraint $i \in [m]$, chosen at the start of round $t_i \in [t', t'']$. Now, there are two possible cases.

Case I: The constraint i did not get enforced in this phase. This happens if $\left(C \cdot \frac{\hat{x}^{t_i}}{W}\right)_i \geq 1 - \epsilon/2$. Here, we derive that $\left(C \cdot \frac{\hat{x}^{t''}}{W}\right)_i \geq \left(C \cdot \frac{\hat{x}^{t_i}}{W}\right)_i \geq 1 - \epsilon/2$, since each co-ordinate of $\hat{x}$ can only increase over time.

Case II: The constraint i got enforced in this phase, by getting repeatedly whacked δ times starting from round t_i. Thus, we have $t_i + \delta < T$ (otherwise, the algorithm would return a dual packing solution y) and $\left(C \cdot \frac{\hat{x}^{t_i+\delta}}{W}\right)_i \geq 1$. Analogous to Case I, here we derive that $\left(C \cdot \frac{\hat{x}^{t''}}{W}\right)_i \geq \left(C \cdot \frac{\hat{x}^{t_i+\delta}}{W}\right)_i \geq 1 \geq 1 - \epsilon/2$.

To summarize, we have the following guarantee for every constraint $i \in [m]$ at the start of round t''.

$$\left(C \cdot \frac{\hat{x}^{t''}}{W}\right)_i \geq 1 - \epsilon/2$$

Since no new phase was initiated before round t'' , we infer that $\left\|\hat{x}^{t''}\right\|_1 \leq (1 - \epsilon/2)^{-1} \cdot W$. Thus we get the following guarantee for every constraint $i \in [m]$.

$$\left(C \cdot x^{t''}\right)_i \geq \left(C \cdot \frac{\hat{x}^{t''}}{W}\right)_i \cdot (1 - \epsilon/2) \geq (1 - \epsilon/2)^2 \geq 1 - \epsilon, \text{ where } x^{t''} := \frac{\hat{x}^{t''}}{\|\hat{x}^{t''}\|_1}$$

In other words, the vector $x^{t''}$ satisfies the inequality $C \cdot x^{t''} \geq (1 - \epsilon) \cdot \mathbb{1}$, and hence the decision to return $(x^{t''}, \text{NULL})$. □

B Reduction to General LPs

Consider a general *packing and covering* linear programs (LPs) defined as follows:

$$\begin{array}{ll} \text{Covering LP:} & \min_{x \in \mathbb{R}^n_{\geq 0}} \left\{c^\top x \mid Ax \geq b\right\}, \\ \text{Packing LP:} & \max_{y \in \mathbb{R}^m_{\geq 0}} \left\{b^\top y \mid A^\top y \leq c\right\}. \end{array}$$

Here each entry of the matrix coefficients $A \in \mathbb{R}^{m \times n}_{\geq 0}, b \in \mathbb{R}^m_{\geq 0}, c \in \mathbb{R}^n_{\geq 0}$ is non-negative. We will now explain how to use Algorithm 3 to obtain ϵ-approximate optimal solution to this pair of LPs in the static setting.

Proposition 3. *There is a deterministic ϵ-approximation algorithm for solving LP that runs in $\widetilde{\mathcal{O}}\left(N \log^3(nU/L)\right)\epsilon^{-3}$ time. Here N denotes the number of non-zero entries in the matrix C, and (L, U) respectively denotes a lower (resp. upper) bound on the minimum (resp. maximum) value of any non-zero entry in the coefficients C, a, b.*

Proof. We construct a matrix $C' \in [L/U^2, U/L^2]^{m \times n}$ by $C'_{ij} := C_{ij}/\left(a_j b_i\right)$ for all $i \in [m], j \in [n]$. Note that the matrix C' can be computed in $O(N)$ time. It is clear that the Packing-Covering LP can be rewritten using C' as follows:

$$\begin{array}{ll} \text{Min } \mathbb{1}^\top x & \text{such that } C'x \geq \mathbb{1} \text{ and } x \in \mathbb{R}^n_{\geq 0} \\ \text{Max } \mathbb{1}^\top y & \text{such that } \left(C'\right)^\top y \leq \mathbb{1} \text{ and } y \in \mathbb{R}^m_{\geq 0} \end{array} \tag{21}$$

Note that the optimal objective value of the LP defined by C' falls in the range of $[L^2/U, nU^2/L]$. To apply Algorithm 3, we consider the set

$$I := \{(1+\epsilon)^i : \text{ for all } i \text{ such that } (1+\epsilon)^i \in [L^2/U, nU^2/L]\}$$

and consider the following problem for each $\mu \in I$.

Problem 1. *Either return an $x \in \mathbb{R}^n_{\geq 0}$ such that $\mathbb{1}^\top x \leq (1+\Theta(\epsilon)) \cdot \mu$ and $C'x \geq (1-\Theta(\epsilon)) \cdot \mathbb{1}$, or return a $y \in \mathbb{R}^m_{\geq 0}$ such that $\mathbb{1}^\top y \geq (1-\Theta(\epsilon)) \cdot \mu$ and $(C')^\top y \leq (1+\Theta(\epsilon)) \cdot \mathbb{1}$.*

Section 4.1 show that we can solve Problem 1 for each μ using Algorithm 3 in $O\left(N \cdot \frac{\log(n)}{\epsilon^2} \cdot \log^2\left(\frac{nU\log(n)}{\epsilon L}\right)\right)$ time. One can easily check that we can solve the original packing-covering LP if we solve Problem 1 for each $\mu \in I$. This concludes the proof. □