

# Approximating the Solution to Mixed Packing and Covering LPs in Parallel $\tilde{O}(\epsilon^{-3})$ Time

Michael W. Mahoney<sup>1</sup>, Satish Rao<sup>2</sup>, Di Wang<sup>3</sup>, and Peng Zhang<sup>4</sup>

1 International Computer Science Institute and Department of Statistics, UC Berkeley, Berkeley, USA

[mmahoney@stat.berkeley.edu](mailto:mmahoney@stat.berkeley.edu)

2 Department of Electrical Engineering and Computer Sciences, UC Berkeley, Berkeley, USA

[satishr@berkeley.edu](mailto:satishr@berkeley.edu)

3 Department of Electrical Engineering and Computer Sciences, UC Berkeley, Berkeley, USA

[wangd@eecs.berkeley.edu](mailto:wangd@eecs.berkeley.edu)

4 Department of Computer Science, Georgia Tech, Atlanta, USA

[pzhang60@gatech.edu](mailto:pzhang60@gatech.edu)

---

## Abstract

We study the problem of approximately solving positive linear programs (LPs). This class of LPs models a wide range of fundamental problems in combinatorial optimization and operations research, such as many resource allocation problems, solving non-negative linear systems, computing tomography, single/multi commodity flows on graphs, etc. For the special cases of pure packing or pure covering LPs, recent result by Allen-Zhu and Orecchia [2] gives  $\tilde{O}(\frac{1}{\epsilon^3})$ -time parallel algorithm, which breaks the longstanding  $\tilde{O}(\frac{1}{\epsilon^4})$  running time bound by the seminal work of Luby and Nisan [10].

We present new parallel algorithm with running time  $\tilde{O}(\frac{1}{\epsilon^3})$  for the more general mixed packing and covering LPs, which improves upon the  $\tilde{O}(\frac{1}{\epsilon^4})$ -time algorithm of Young [18, 19]. Our work leverages the ideas from both the optimization oriented approach [2, 17], as well as the more combinatorial approach with phases [18, 19]. In addition, our algorithm, when directly applied to pure packing or pure covering LPs, gives a improved running time of  $\tilde{O}(\frac{1}{\epsilon^2})$ .

**1998 ACM Subject Classification** F.2 Analysis of Algorithms and Problem Complexity

**Keywords and phrases** Mixed packing and covering, Linear program, Approximation algorithm, Parallel algorithm

**Digital Object Identifier** 10.4230/LIPIcs.ICALP.2016.52

## 1 Introduction

Mixed packing and covering linear programs (LPs) are LPs formulated with non-negative coefficients, non-negative constraints and non-negative variables. They model a wide range of fundamental problems in combinatorial optimization and operations research, thus have long drawn interest in theoretical computer science [10, 18, 2, 1]. Notable special cases include pure packing LPs and pure covering LPs, which apply to most resource allocation problems, and can be formulated respectively as  $\max_{x \geq 0} \{c^T x : Ax \leq b\}$  and  $\min_{x \geq 0} \{c^T x : Ax \geq b\}$  where  $c, A, b \geq 0$ . More general than the pure packing and covering LPs, the mixed packing and covering LPs further capture problems requiring both packing and covering constraints, including solving non-negative linear systems, computing tomography, and single/multi commodity flows on graphs.



© Michael Mahoney and Satish Rao and Di Wang and Peng Zhang;  
licensed under Creative Commons License CC-BY

43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016).

Editors: Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi;

Article No. 52; pp. 52:1–52:14



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



Formally, the mixed packing and covering LP is the optimization problem

$$\min\{\lambda : \mathbf{P}\mathbf{x} \leq \lambda\mathbf{p}, \mathbf{C}\mathbf{x} \geq \mathbf{c}, \mathbf{x} \geq 0\} \quad (1)$$

where  $\mathbf{P}$ ,  $\mathbf{C}$ ,  $\mathbf{p}$ ,  $\mathbf{c}$  all have non-negative entries. A  $(1 + \epsilon)$ -approximation is a feasible solution  $\lambda$ ,  $\mathbf{x}$  achieving  $\lambda \leq (1 + \epsilon)\lambda_{\text{OPT}}$ .

Although one can use general LP solvers such as interior point method to solve packing and covering with convergence rate of  $\log(\frac{1}{\epsilon})$  [8, 9, 5], such algorithms usually have very high per-iteration cost, as methods such as the computation of the Hessian and matrix inversion are involved. With the abundance of large-scale datasets, as well as the growing reliance on multiprocessors and cloud computing, low precision iterative solvers that can be highly parallelized are often more popular choices. Such parallel solvers compute approximate solutions usually in time with a poly-log dependence on the problem size, and nearly-linear total work, but they have  $\text{poly}(\frac{1}{\epsilon})$  dependence on the approximation parameter  $\epsilon$ .

Based on whether the running time depends on the width  $\rho$ , a parameter which typically depends on the dimension and the largest entry of  $\mathbf{A}$ , these algorithms can be divided into width-dependent solvers and width-independent solvers. Width-dependent solvers are usually pseudo-polynomial, as the running time depends at least linearly on  $\rho \text{OPT}$ , which itself can be large, while width-independent solvers are more efficient in the sense that they provide truly polynomial-time approximation solvers.

In this paper we focus on width-independent algorithms that produce  $1 + \epsilon$  approximations in  $\text{poly}(\log n, \frac{1}{\epsilon})$  time and nearly-linear work. Time and work are standard notions from parallel algorithms that correspond to the longest chain of dependent operations and the total operations performed. In particular, time has a natural correspondence with iteration count, and these two measures have been used to study the performance of packing/covering LPs before [19]. Since the focus of this line of work is not on optimizing the log factors, we will follow the standard practice of using  $\tilde{O}$  to hide poly-log factors (which are at most  $\log^3 n$ ) in our discussion<sup>1</sup>.

## 1.1 Previous work

Most of the works on mixed packing and covering LPs, as well as the works on the special cases of pure packing and pure covering LPs, take one of the two approaches. The first approach is based on turning the constrained LPs into convex and smooth objective functions with trivial or no constraints. Approximately solving the LP is then reduced to approximately optimizing the smoothed function (see [13]), and general-purpose optimization schemes are usually applied directly. This approach traditionally gives algorithms that work in more general settings, but are width-dependent in the case of packing and covering (e.g., [12, 3, 13, 16]). Recent breakthroughs in [2, 1] leverage the insight from optimization ([21]) and the special structure of packing and covering problems, and get the first width-independent algorithms using first order optimization methods. In particular, [2] gives a parallel algorithm that takes  $\tilde{O}(\frac{1}{\epsilon^3})$  time and  $\tilde{O}(\frac{N}{\epsilon^3})$  work. Here  $N$  is the size of the input, i.e., the total number of non-zeros in the constraint matrices  $\mathbf{P}$  and  $\mathbf{C}$ . The result can be improved to run in  $\tilde{O}(\frac{1}{\epsilon^2})$  time and  $\tilde{O}(\frac{N}{\epsilon^2})$  work with simple modifications [17]. As for sequential algorithms, the remarkable result in [1] combines width-independence with Nesterov-like acceleration ([13, 14]), and gets a randomized algorithm with running time  $\tilde{O}(\frac{N}{\epsilon})$ . However, both results

---

<sup>1</sup>  $\tilde{O}(f(n))$  is often used to denote  $O(f(n)\log^{O(1)}(f(n)))$ , we modify it to include an additional factor of  $\log^{O(1)} n$ .

in [2, 1] are limited to pure packing and pure covering problems, and prior to our work no methods are able to obtain a time of  $\tilde{O}(\frac{1}{\epsilon^3})$  or better for mixed covering and packing LPs [19].

The other approach is based on the Lagrangian-relaxation framework, where, similar to the optimization approach, certain hard constraints are replaced by a soft scalar-value penalty function, and the partial solution is iteratively updated to satisfy the remaining constraints while minimizing the increase of the penalty function. The analysis of the Lagrangian-relaxation algorithms have more of a combinatorial flavor compared to the optimization schemes. Examples include the seminal work of Luby and Nisan [10], which gives the first width-independent algorithm for packing and covering, as well as subsequent works which improve the Luby and Nisan result in various ways [6, 18, 19, 7, 4, 11]. Among them, only [18, 19, 11] work with mixed packing and covering LPs, while others only work in the pure packing and pure covering setting. For algorithms working on mixed packing and covering, the results in [18, 19] have the best theoretical guarantee on parallel running time, with  $\tilde{O}(\frac{1}{\epsilon^4})$  running time and  $\tilde{O}(\frac{N}{\epsilon^2})$  total work. The result in [11] has worse bounds, but is stateless, which is a computational model on distributed algorithms with more restrictions on the processors (see [4]).

## 1.2 Our results

In this paper, we present a parallel algorithm that, given a mixed packing and covering LP with  $m$  variables and  $n$  total constraints, in  $\tilde{O}(\frac{1}{\epsilon^3})$  iterations computes a  $(1 + \epsilon)$ -approximate solution, or correctly reports the original mixed packing and covering LP is infeasible. The algorithm is deterministic and width-independent.

The bottleneck of each iteration is a matrix-vector multiplication, and can be implemented in  $O(\log N)$  depth, in which case the running time of our algorithm is  $\tilde{O}(\frac{1}{\epsilon^3})$ . The total work of the algorithm we present in the paper is  $\tilde{O}(\frac{N}{\epsilon^3})$ . In particular, our result improves upon the current fastest parallel algorithm of mixed packing and covering LPs in [18, 19], where the running time is  $\tilde{O}(\frac{1}{\epsilon^4})$ . The work of the parallel algorithm in [18, 19] is  $\tilde{O}(\frac{N}{\epsilon^2})$ . We note that using a simple lazy update modification on the algorithm, which is the same technique used in [18, 19], we can reduce the work of our algorithm to  $\tilde{O}(\frac{N}{\epsilon^2})$ . Same as in [18, 19], this comes at the cost of requiring a centralized step in the parallel algorithm. Since the iteration count is the more interesting side of this line of work, we will not incorporate the lazy update in our algorithm for simplicity.

Furthermore, in the case of pure packing problem or pure covering problem, our algorithm allows a similar but simplified analysis, and will converge in  $\tilde{O}(\frac{1}{\epsilon^2})$  iterations. This matches the running time achieved by the line of work [2, 17], but has the advantage of being deterministic and without centralized steps. We note that the technique we use in the analysis of the potential function in this work can be applied in a straightforward way to improve the result of [2] to have  $\tilde{O}(\frac{1}{\epsilon^2})$  running time.

## 2 Technical overview

To compute  $(1 + \epsilon)$ -approximation of a mixed packing and covering LP in the optimization form (1), via standard reduction and scaling (e.g., see [18]), it suffices to solve a  $(1 + O(\epsilon))$ -feasibility problem as specified in (3) and (4).

At a high level, our work follows the Lagrangian-relaxation approach as in [18, 19]. In particular, we replace the hard packing and covering constraints with a scalar-valued potential function, which is continuous and smooth. The potential function measures how far away

the current solution is from satisfying all the captured constraints. As in [18], we use the soft-max  $\text{lmax}(\mathbf{P}\mathbf{x})$  and soft-min  $\text{lmin}(\mathbf{C}\mathbf{x})$  in our potential function

$$\text{lmax}(\mathbf{P}\mathbf{x}) = \ln\left(\sum_j \exp(\mathbf{P}\mathbf{x}_j)\right), \text{ and } \text{lmin}(\mathbf{C}\mathbf{x}) = -\ln\left(\sum_j \exp(-\mathbf{C}\mathbf{x}_j)\right).$$

In particular, these functions give smooth approximation to  $\max_j(\mathbf{P}\mathbf{x})_j$  and  $\min_j(\mathbf{C}\mathbf{x})_j$ :

$$\begin{aligned} \max_j(\mathbf{P}\mathbf{x})_j &\leq \text{lmax}(\mathbf{P}\mathbf{x}) \leq \max_j(\mathbf{P}\mathbf{x})_j + \ln n \\ \min_j(\mathbf{C}\mathbf{x})_j &\geq \text{lmin}(\mathbf{C}\mathbf{x}) \geq \min_j(\mathbf{C}\mathbf{x})_j - \ln n. \end{aligned} \tag{2}$$

The potential function we use will be

$$f(\mathbf{x}) = \text{lmax}(\mathbf{P}\mathbf{x}) - \text{lmin}(\mathbf{C}^t\mathbf{x}),$$

which is approximately the difference between  $\max_j(\mathbf{P}\mathbf{x})_j$  and  $\min_j(\mathbf{C}\mathbf{x})_j$ .

The overall framework is to start with a  $\mathbf{x}$  of very small values, and iteratively increase  $\mathbf{x}$  while keeping  $f(\mathbf{x})$  small. Roughly, when  $\mathbf{x}$  is large enough so that

$$\max\{\max_j(\mathbf{P}\mathbf{x})_j, \min_j(\mathbf{C}\mathbf{x})_j\} \geq \frac{1}{\epsilon} f(\mathbf{x}),$$

we know that  $\max_j(\mathbf{P}\mathbf{x})_j \leq (1 + O(\epsilon)) \min_j(\mathbf{C}\mathbf{x})_j$ .

Same as in [18, 19], each iteration a subset of the variables are picked based on the gradients of the potential function, and are updated within a local smooth region so we can bound the change of the potential function. In particular, for each variable  $\mathbf{x}_i$  we compute the packing gradient  $\mathbf{a}_i = \nabla_i \text{lmax}(\mathbf{P}\mathbf{x})$ , and the covering gradient  $\mathbf{b}_i = \nabla_i \text{lmin}(\mathbf{C}\mathbf{x})$ . We want to update variables without increasing  $f(\mathbf{x})$ , and since  $\nabla_i f(\mathbf{x}) = \mathbf{a}_i - \mathbf{b}_i$ , a variable will be included in the update subset only when its covering gradient  $\mathbf{b}_i$  is larger than its packing gradient  $\mathbf{a}_i$ .

However, unlike the parallel algorithm in [18, 19] that multiplicatively updates all variables in the subset with a uniform step size, we further incorporate the gradients  $\mathbf{a}$  and  $\mathbf{b}$  into step sizes of individual variables' updates. This discriminative multiplicative step size allows more aggressive updates on average, and is directly motivated by the line of works using gradient based optimization methods [2, 1, 17]. However, we move away from the optimization oriented view of these update steps in favor of more localized and adhoc analyses, which was developed to analyze direct adaptations of Young's algorithm for purely-packing SDPs [15], leading to bounds similar to the optimization based approaches [20].

Similar to Young's algorithm [18], the overall progress of the algorithm is captured by how large the constraints become. A sufficient termination condition for the algorithm is when a variable is increased by more than a certain amount, so we know  $\max_j(\mathbf{P}\mathbf{x})_j$  and  $\min_j(\mathbf{C}\mathbf{x})_j$  are large enough. To bound the number of iterations before any variable gets too large, we combine the notion of phases from [18] with the more refined analysis of gradient updates from more recent works [17]. This is owing to the clearer combinatorial structure of our interpretation of discriminative multiplicative steps.

On a high level, the analysis considers *phases*, where each phase is a consecutive sequence of iterations. The definition of a phase captures a local window, where the algorithm only makes limited global progress. The limited global progress ensures that inside a single phase the landscape doesn't change too much, which translate to certain monotonicity-like property on the gradients within the interactions captured by a single phase. In [18], each phase is defined to cover very little overall progress, which gives a very strong monotonicity-like

property. In particular, any variable being increased at the last iteration of a phase must have been increased in every iteration of the phase, which, coupled with a lower bound on each increase, gives a bound of the number of iterations in a phase.

In our analysis, we significantly expand the phases to capture larger global progress, leading to a smaller number of phases. The larger phases lead to a weaker monotonicity-like property on the gradients within a phase, and we develop a new approach to bound the number of iterations in our phase. We divide the iterations into two groups: some initial warm-up *bad* iterations followed by subsequent *good* iterations containing more interesting segments of the path to convergence (see Definition 7 for formal definitions). The bad iterations are ones where packing gradients are much smaller than covering gradients. These are the easy iterations to deal with, since a small ratio of the packing gradient over the covering gradient is a clear signal to increase a variable by a lot. An analysis identical to Young’s algorithm [18] shows that they must occur near the very start of a phase, and there cannot be too many of them. In the subsequent good iterations, the packing gradients for all variables are relatively large comparing to their covering counterparts. These iterations capture the more difficult part of the path to convergence, since we only get weak signals as to which variables to update, and we can only increase them by small steps. We can show that as long as the problem is feasible, there cannot be too many good iterations in a phase. Intuitively, if the primal LP is feasible, the dual solution certifies that there must be some key variable(s) we increase during the phase to achieve the fixed global progress. Particularly, we know that there is at least one variable that on average has smaller packing gradients than covering gradients. Moreover, since in the good iterations, the packing gradients are all at least on the same scale as the covering gradients, an argument in the spirit of Markov’s inequality then implies that the corresponding variable was increased by a large amount, which in turn leads to a bound on the number of iterations of a phase.

## 2.1 Remarks

We note that the phases in our result are virtual: we only need them in the analysis, but not in the actual execution of the algorithm. In particular, this modification to Young’s algorithm [18] removes the dependency of updates on the phase of the overall algorithm as well as the current gradient. We believe this direct removal of phases also apply to other variants of Young’s algorithm [19].

Furthermore, we believe that there is a more natural variant of the analysis that does not rely on phases, and treats all the iterations in a completely symmetric manner. Such an analysis is likely crucial for extending our results to the SDP setting, where the gradients exhibit much weaker monotonicity behaviors [20]. We are optimistic that it will lead to an  $\tilde{O}(\frac{1}{\epsilon^2})$  bound for the mixed packing-covering case, which we believe is the more likely asymptotic behaviors of phase-less, gradient update variants of Young’s algorithm [18, 19].

## 3 Parallel Algorithm for Mixed Packing and Covering LPs

### 3.1 Preliminaries

To compute  $(1 + \epsilon)$ -approximation of a mixed packing and covering LP in the optimization form (1), via standard reduction and scaling (e.g., see [18]), it suffices to solve the following  $(1 + O(\epsilon))$ -feasibility problem, that is, either find  $\mathbf{x} \geq 0$  such that

$$0 < \max_j (\mathbf{P}\mathbf{x})_j \leq (1 + \epsilon) \min_j (\mathbf{C}\mathbf{x})_j. \quad (3)$$

or conclude the following LP is infeasible

$$\begin{aligned} \mathbf{C}\mathbf{x} &\geq \mathbf{1} \\ \mathbf{P}\mathbf{x} &\leq (1 - 10\epsilon)\mathbf{1} \\ \mathbf{x} &\geq 0. \end{aligned} \tag{4}$$

We present our parallel  $\tilde{O}(1/\epsilon^3)$  routine in Algorithm 1 for solving the  $(1 + O(\epsilon))$ -feasibility problem above, that is, either find  $\mathbf{x} \geq 0$  satisfying (3), or certify the infeasibility of (4). The input contains a packing constraint matrix  $\mathbf{P} \in \mathbb{R}_{\geq 0}^{n_P \times m}$ , a covering constraint matrix  $\mathbf{C} \in \mathbb{R}_{\geq 0}^{n_C \times m}$ , and an error parameter  $\epsilon > 0$ . That is, there are  $m$  variables,  $n_P$  packing constraints, and  $n_C$  covering constraints. We also use  $n = n_P + n_C$  to denote the total number of constraints.

To certify that (4) is infeasible, we rely on the dual LP of (4).

► **Lemma 1.** *By duality, (4) is infeasible if there exists  $\mathbf{y}, \mathbf{z} \geq 0$  s.t.*

$$(1 - 10\epsilon) \frac{\mathbf{C}^T \mathbf{z}}{\mathbf{1}^T \mathbf{z}} < \frac{\mathbf{P}^T \mathbf{y}}{\mathbf{1}^T \mathbf{y}}. \tag{5}$$

**Proof.** Eqn. (5) is a direct reformulation of the dual LP of (4). Since we only need the sufficient condition, the result is by weak duality. If there exists any  $\mathbf{x} \geq 0$  satisfying (4), we have

$$\begin{aligned} (1 - 10\epsilon) \frac{\mathbf{x}^T \mathbf{C}^T \mathbf{z}}{\mathbf{1}^T \mathbf{z}} &\geq (1 - 10\epsilon) \frac{\mathbf{1}^T \mathbf{z}}{\mathbf{1}^T \mathbf{z}} = 1 - 10\epsilon, \\ \frac{\mathbf{x}^T \mathbf{P}^T \mathbf{y}}{\mathbf{1}^T \mathbf{y}} &\leq (1 - 10\epsilon) \frac{\mathbf{1}^T \mathbf{y}}{\mathbf{1}^T \mathbf{y}} = 1 - 10\epsilon. \end{aligned}$$

Together they give  $1 < 1$ , contradiction. ◀

### 3.2 Algorithm

We start with small  $\mathbf{x}_i^{(0)} = \frac{1}{m \|\mathbf{P}_{:,i}\|_\infty}$ ,  $\forall i \in [m]$ , and keep increasing  $\mathbf{x}$  properly, until it reaches the termination condition in line 4, that is,  $\max\{\max_j(\mathbf{P}\mathbf{x})_j, \min_j(\mathbf{C}\mathbf{x})_j\} \geq K = \frac{10 \ln n}{\epsilon}$ . The reason of the chosen  $K$  value is stated in Lemma 5.

In each iteration of the while-loop, we first delete all covering constraints which have already reached  $K$ . Since  $\mathbf{x}$  never decreases, we know that once a row is deleted, we no longer need to look at it. Note the covering matrix cannot be empty, since we enter the iteration with  $\min_j(\mathbf{C}\mathbf{x})_j < K$ . We compute the vectors  $\mathbf{y}, \mathbf{z}$ , which are exponentials of the values of the packing and covering constraints respectively. We then compute  $\mathbf{a}$  and  $\mathbf{b}$ , which can be considered as gradients of  $\text{lmax}(\mathbf{P}\mathbf{x})$  and  $\text{lmin}(\mathbf{C}\mathbf{x})$  respectively, and use them to guide our update on  $\mathbf{x}$ . In particular, we update  $\mathbf{x}_i$  if  $\mathbf{a}_i \leq (1 - \epsilon/50)\mathbf{b}_i$  (i.e.,  $i \in B$ ). Furthermore, we update  $\mathbf{x}_i$  multiplicatively by a factor depends on the ratio of  $\frac{\mathbf{a}_i}{\mathbf{b}_i}$ , as specified in Eqn. (6) and line 12. Note that the smallest update in our algorithm is by a factor of  $(1 + \Omega(\frac{\epsilon^2}{\ln n}))$ , which is the same as the fixed update step size in [19], and in general our updates take larger steps.

Note that in our analysis, we equivalently view  $\mathbf{z}$  as the full  $n_C$ -dimensional vector, where the coordinates corresponding to deleted constraints are filled by 0's. In particular, the matrix-vector product of the original  $\mathbf{C}$  with the  $n_C$ -dimensional  $\mathbf{z}$  will be the same as the product of the reduced covering matrix  $\mathbf{C}^{(t)}$  and reduced  $\mathbf{z}$ .

---

**Algorithm 1** Parallel algorithm for mixed packing and covering LPs
 

---

**Input:**  $\mathbf{P}, \mathbf{C}, \epsilon$ **Output:** "infeasible" or  $\mathbf{x} \geq 0$  s.t.  $\max_j(\mathbf{P}\mathbf{x})_j \leq (1 + \epsilon) \min_j(\mathbf{C}\mathbf{x})_j$ 

- 1: Let  $K = \frac{10 \ln n}{\epsilon}$ ,  $\alpha = \frac{1}{K}$ , where  $n$  is the number of constraints.
  - 2: Initialize  $\mathbf{x}_i^{(0)} = \frac{1}{m \|\mathbf{P}_{:,i}\|_\infty}$ ,  $\forall i \in [m]$ , where  $m$  is the number of variables.
  - 3: Let  $t = 0$ .
  - 4: **while**  $\max_j(\mathbf{P}\mathbf{x})_j < K$  and  $\min_j(\mathbf{C}\mathbf{x})_j < K$  **do**
  - 5:   Let  $\mathbf{C}^{(t)}$  be  $\mathbf{C}$  with rows  $j$  such that  $(\mathbf{C}\mathbf{x}^{(t)})_j \geq K$  deleted.
  - 6:   Let  $\mathbf{y}^{(t)} = \exp(\mathbf{P}\mathbf{x}^{(t)})$ ,  $\mathbf{z}^{(t)} = \exp(-\mathbf{C}^{(t)}\mathbf{x}^{(t)})$ .
  - 7:
  - 8:    $\mathbf{a}^{(t)} = \frac{\mathbf{P}^T \mathbf{y}^{(t)}}{\mathbf{1}^T \mathbf{y}^{(t)}}$ ,  $\mathbf{b}^{(t)} = \frac{(\mathbf{C}^{(t)})^T \mathbf{z}^{(t)}}{\mathbf{1}^T \mathbf{z}^{(t)}}$ .
  - 9:   Define  $B^{(t)} = \{i : \mathbf{a}_i^{(t)} \leq (1 - \frac{\epsilon}{50}) \mathbf{b}_i^{(t)}\}$ .
  - 10:   If  $B^{(t)} = \emptyset$ , then return "infeasible".
  - 11:   Let
 
$$\Delta_i^{(t)} = \begin{cases} \frac{1}{2} (1 - \frac{\mathbf{a}_i^{(t)}}{\mathbf{b}_i^{(t)}}) \in [\epsilon/100, \frac{1}{2}] & \text{if } i \in B^{(t)} \\ 0 & \text{if } i \notin B^{(t)} \end{cases} \quad (6)$$
  - 12:    $\mathbf{x}_i^{(t+1)} \leftarrow \mathbf{x}_i^{(t)} (1 + \alpha \Delta_i^{(t)})$ .
  - 13:    $t \leftarrow t + 1$ .
  - 14: **end while**
  - 15: **return**  $\mathbf{x} = \frac{\mathbf{x}^{(t)}}{K}$ .
- 

### 3.3 Proof of Correctness

In this section we will show Algorithm 1 will terminate, and output the correct answer.

Lemma 2 shows that empty  $B$  certifies the infeasibility of the input instance (4), which proves the correctness if we end up in the case of line 10.

► **Lemma 2.** *If the problem instance (4) is feasible, then*

$$\forall \mathbf{x} \geq 0, B = \{i : \mathbf{a}_i \leq (1 - \frac{\epsilon}{50}) \mathbf{b}_i\} \neq \emptyset.$$

**Proof.** Assume by contradiction,  $\exists \mathbf{x} \geq 0, \forall i \in [m], \mathbf{a}_i^{(t)} > (1 - \frac{\epsilon}{50}) \mathbf{b}_i^{(t)}$ . By definition of  $\mathbf{a}, \mathbf{b}$ , it is equivalent to  $\exists \mathbf{y}, \mathbf{z} \geq 0$  such that

$$(1 - \frac{\epsilon}{50}) \frac{\mathbf{C}^T \mathbf{z}}{\mathbf{1}^T \mathbf{z}} < \frac{\mathbf{P}^T \mathbf{y}}{\mathbf{1}^T \mathbf{y}}.$$

Then the result follows directly from Lemma 1. ◀

If Algorithm 1 does not terminate with line 10, it must increase at least one variable by at least a factor of  $(1 + \frac{\epsilon^2}{10 \ln n})$  each iteration, so the algorithm must reach the termination condition of the while loop at some point, and we need to show the output  $\mathbf{x}$  satisfies (3). Recall that we use the potential function,

$$f(\mathbf{x}^{(t)}) = \text{lmax}(\mathbf{P}\mathbf{x}^{(t)}) - \text{lmin}(\mathbf{C}^{(t)}\mathbf{x}^{(t)}) = \ln(\mathbf{1}^T \mathbf{y}^{(t)}) + \ln(\mathbf{1}^T \mathbf{z}^{(t)}).$$

We first quantify the changes of  $\text{lmax}$  and  $\text{lmin}$  when we update the variables. This type of smoothness analysis is standard in analyzing algorithms that make updates using gradient information. Similar results are derived in other works on packing and covering (see [2, 18]). The particular analysis we develop can deal with larger gradient steps. In particular, the approach of our analysis allows updates that may move the gradients of some variables out of their respective coordinate-wise smooth regions, as long as we can still bound the combined impact on the potential function from updates of all variables. This approach can extend straightforwardly to show larger updates also work in [2], and improve their pure packing algorithm to run in  $\tilde{O}(\frac{1}{\epsilon^2})$  iterations. Since the proof is straightforward but technically tedious, we omit it.

► **Lemma 3.** *At each iteration  $t$ ,*

$$\text{lmax}(\mathbf{P}\mathbf{x}^{(t+1)}) \leq \text{lmax}(\mathbf{P}\mathbf{x}^{(t)}) + \alpha \langle \mathbf{a}^{(t)}, (1 + \Delta^{(t)}) \circ \Delta^{(t)} \circ \mathbf{x}^{(t)} \rangle$$

and

$$\text{lmin}(\mathbf{C}^{(t+1)}\mathbf{x}^{(t+1)}) \geq \text{lmin}(\mathbf{C}^{(t)}\mathbf{x}^{(t)}) + \alpha \langle \mathbf{b}^{(t)}, (1 - \Delta^{(t)}) \circ \Delta^{(t)} \circ \mathbf{x}^{(t)} \rangle,$$

where  $\Delta \circ \mathbf{x}$  is the entry-wise product vector, i.e.,  $(\Delta \circ \mathbf{x})_i = \Delta_i \mathbf{x}_i$ .

With the above bounds on the changes of the two components  $\text{lmax}(\mathbf{P}\mathbf{x})$  and  $\text{lmin}(\mathbf{C}\mathbf{x})$ , we can show how our updates move the potential function  $f(\mathbf{x})$ .

► **Lemma 4.** *Given  $\max_j(\mathbf{P}\mathbf{x}^{(t)})_j < \frac{10 \ln n}{\epsilon}$  and  $\min_j(\mathbf{C}\mathbf{x}^{(t)})_j < \frac{10 \ln n}{\epsilon}$ , we always have  $f(\mathbf{x}^{(t)}) \leq 2 \ln n$  during the execution of Algorithm 1.*

**Proof.** Initially,  $\mathbf{x}_i^{(0)} = \frac{1}{m \|\mathbf{P}_{:,i}\|_\infty}$ , we have  $\mathbf{P}\mathbf{x}^{(0)} \leq \mathbf{1}$  and  $\mathbf{C}\mathbf{x} \geq \mathbf{0}$ , thus  $f(\mathbf{x}^{(0)}) \leq 2 \ln n$ . To show  $f(\mathbf{x}) \leq 2 \ln n$  for all iterations  $t$  before terminate, it suffices to show that  $f(\mathbf{x})$  is non-increasing during the process. From Lemma 3,

$$\begin{aligned} f(\mathbf{x}^{(t+1)}) - f(\mathbf{x}^{(t)}) &\leq \alpha \langle \mathbf{a}, (1 + \Delta) \circ \Delta \circ \mathbf{x}^{(t)} \rangle - \alpha \langle \mathbf{b}, (1 - \Delta) \circ \Delta \circ \mathbf{x}^{(t)} \rangle \\ &= \sum_i \alpha \Delta_i \mathbf{x}_i (\mathbf{a}_i (1 + \Delta_i) - \mathbf{b}_i (1 - \Delta_i)). \end{aligned}$$

For each  $i \in [m]$ , by our update rule (6), either  $\Delta_i$ , or  $\Delta_i = \frac{1}{2}(1 - \frac{\mathbf{a}_i}{\mathbf{b}_i})$ , in which case

$$\mathbf{a}_i (1 + \Delta_i) - \mathbf{b}_i (1 - \Delta_i) = \frac{3\mathbf{a}_i \mathbf{b}_i - \mathbf{a}_i^2}{2\mathbf{b}_i} - \frac{\mathbf{a}_i + \mathbf{b}_i}{2} = \frac{2\mathbf{a}_i \mathbf{b}_i - \mathbf{a}_i^2 - \mathbf{b}_i^2}{2\mathbf{b}_i} \leq 0,$$

so all the summands are non-positive, thus  $f(\mathbf{x})$  is non-increasing. ◀

The above lemma guarantees that the difference between  $\text{lmax}(\mathbf{P}\mathbf{x})$  and  $\text{lmin}(\mathbf{C}\mathbf{x})$  is bounded by  $2 \ln n$ , which by Eqn. (2) suggests  $\max_j(\mathbf{P}\mathbf{x})_j \leq \min_j(\mathbf{C}\mathbf{x})_j + O(\ln n)$ . Then when the two terms are large at termination, we are approximately feasible as the difference is a factor of  $\epsilon$  smaller.

► **Lemma 5.** *If Algorithm 1 terminates with line 15, then it returns an  $\mathbf{x} \geq \mathbf{0}$  with  $0 < \max_j(\mathbf{P}\mathbf{x})_j \leq (1 + \epsilon) \min_j(\mathbf{C}\mathbf{x})_j$ .*

**Proof.** Suppose the algorithm terminates at iteration  $T$ , that is,  $\max_j(\mathbf{P}\mathbf{x}^{(T)})_j \geq \frac{10 \ln n}{\epsilon}$  or  $\min_j(\mathbf{C}\mathbf{x}^{(T)})_j \geq \frac{10 \ln n}{\epsilon}$ . Consider iteration  $T-1$ , the covering matrix is not empty (otherwise, the algorithm terminates before iteration  $T$ ). Since  $\mathbf{x}^{(T)} = \mathbf{x}^{(T-1)} \circ (1 + \alpha \Delta^{(T-1)}) \leq (1 + \frac{5\epsilon}{\ln n}) \mathbf{x}^{(T-1)}$ , we have  $\max_j(\mathbf{P}\mathbf{x}^{(T-1)})_j \geq \frac{5 \ln n}{\epsilon}$  or  $\min_j(\mathbf{C}\mathbf{x}^{(T-1)})_j \geq \frac{5 \ln n}{\epsilon}$ .



By Lemma 4,

$$\begin{aligned} \max_j(\mathbf{P}\mathbf{x}^{(T-1)})_j &\leq \text{lmax}(\mathbf{P}\mathbf{x}^{(T-1)}) \leq \text{lmin}(\mathbf{C}\mathbf{x}^{(T-1)}) + 2 \ln n \\ &\leq \min_j(\mathbf{C}\mathbf{x}^{(T-1)})_j + 2 \ln n. \end{aligned}$$

Since  $2 \ln n \leq \epsilon \cdot \frac{5 \ln n}{\epsilon}$ , we have

$$\max_j(\mathbf{P}\mathbf{x}^{(T-1)})_j \leq (1 + \epsilon) \min_j(\mathbf{C}\mathbf{x}^{(T-1)})_j.$$

This also gives  $\max_j(\mathbf{P}\mathbf{x}^{(T)})_j \leq (1 + \epsilon) \min_j(\mathbf{C}\mathbf{x}^{(T)})_j$ , since  $\mathbf{x}^T$  is within in multiplicative factor  $1 + \frac{\epsilon}{10 \ln n}$  of  $\mathbf{x}^{T-1}$ . Since we start with  $\mathbf{x} > 0$ , and only increase  $\mathbf{x}$ , we also have  $\max_j(\mathbf{P}\mathbf{x})_j > 0$ . So the  $\mathbf{x}$  we return at the end satisfies (3). ◀

### 3.4 Analysis of Convergence

So far we have proved that Algorithm 1 will terminate, and will either output  $\mathbf{x}$  satisfying (3) at the end, or terminate earlier and correctly certify (4) is infeasible. In this section we show that if (4) is feasible, Algorithm 1 must finish with the first case in  $\tilde{O}(\frac{1}{\epsilon^3})$  iterations, so if the algorithm takes more than  $\frac{1000 \ln n \ln(\frac{n}{\epsilon})}{\epsilon^3}$  iterations to complete, we can terminate it, and correctly output that (4) is infeasible.

We adapt the concept *phase* from Young's algorithm. Note phase is only used in our analysis, and our algorithm does not contain phase. Formally, phase  $s$  contains the iterations  $t$  such that

$$\frac{n_P}{n_C} \cdot 2^s \leq \frac{\mathbf{1}^T \mathbf{y}^{(t)}}{\mathbf{1}^T \mathbf{z}^{(t)}} < \frac{n_P}{n_C} \cdot 2^{s+1}$$

where  $n_P$  is the number of packing constraints and  $n_C$  is the number of covering constraints.

Since we only increase  $\mathbf{x}$ ,  $\frac{\mathbf{1}^T \mathbf{y}}{\mathbf{1}^T \mathbf{z}}$  is monotonically increasing, so each phase covers a consecutive sequence of iterations. Furthermore, as  $\ln(\frac{\mathbf{1}^T \mathbf{y}}{\mathbf{1}^T \mathbf{z}}) = \text{lmax}(\mathbf{P}\mathbf{x}) + \text{lmin}(\mathbf{C}\mathbf{x})$  measures global progress towards termination, each phase captures a fixed amount of progress. From our definition of phases, and the termination condition, we have the following lemma.

► **Lemma 6.** *The total number of phases in Algorithm 1 is  $O(\frac{\log n}{\epsilon})$ .*

**Proof.** Since  $\mathbf{x}$  is monotonically increasing,  $\mathbf{y} = \exp(\mathbf{P}\mathbf{x})$  and  $\mathbf{z} = \exp(-\mathbf{C}\mathbf{x})$  are monotonically increasing and decreasing respectively, which implies that the quantity  $\frac{\mathbf{1}^T \mathbf{y}}{\mathbf{1}^T \mathbf{z}}$  is monotonically increasing. Initially  $\mathbf{P}\mathbf{x}^{(0)} \geq 0$ ,  $\mathbf{C}\mathbf{x}^{(0)} \geq 0$ , we know  $\frac{\mathbf{1}^T \mathbf{y}^{(0)}}{\mathbf{1}^T \mathbf{z}^{(0)}} \geq \frac{n_P}{n_C}$ . By the termination condition in Algorithm 1, the ratio never goes beyond  $n_P \exp(\frac{10 \log n}{\epsilon})$ . Therefore, the total number of phases is  $O(\frac{\log n}{\epsilon})$ . ◀

We now bound the number of iterations in a single phase. The iterations of a phase are divided into two groups, the bad iterations and the good iterations, formally defined as follows.

► **Definition 7.** If in an iteration  $t$ , we have for all  $i$

$$\frac{\mathbf{a}_i^{(t)}}{\mathbf{b}_i^{(t)}} > \frac{1}{3}, \tag{7}$$

then we call it a *good* iteration. Otherwise we call it a *bad* iteration.

Note that it is also possible for a phase to contain only bad iterations or only good iterations. We bound the total number of iterations in the two groups separately.

As discussed earlier, the bad iterations capture the initial warm-up iterations of a phase, where in any bad iteration, we can identify some variable  $\mathbf{x}_i$  with a strong signal (i.e.,  $\frac{\mathbf{a}_i}{\mathbf{b}_i} \leq \frac{1}{3}$ ), so we can increase the variable by a lot. This restricts the warm-up sequence from getting too long, and we formalize the intuition in the following lemma.

► **Lemma 8.** *In a single phase, the number of bad iterations is at most  $O(\ln n \ln(\frac{m}{\epsilon})/\epsilon)$ .*

**Proof.** We will prove the result by showing that there cannot be any bad iteration after the initial  $100 \ln n \ln(\frac{m}{\epsilon})/\epsilon$  iterations of a phase. By contradiction, if for any variable  $i$ , after  $\Omega(\ln n \ln(\frac{m}{\epsilon})/\epsilon)$  iterations of a phase, we have at iteration  $t$  such that for some  $i$ ,

$$\frac{\mathbf{a}_i^{(t)}}{\mathbf{b}_i^{(t)}} = \frac{(\mathbf{P}^T \mathbf{y}^{(t)})_i}{\mathbf{1}^T \mathbf{y}^{(t)}} \frac{\mathbf{1}^T \mathbf{z}^{(t)}}{(\mathbf{C}^T \mathbf{z}^{(t)})_i} \leq \frac{1}{3},$$

then this ratio is at most  $\frac{2}{3}$  in all previous iterations of this phase, since  $\frac{(\mathbf{P}^T \mathbf{y})_i}{(\mathbf{C}^T \mathbf{z})_i}$  is monotonically increasing, and  $2^s \leq \frac{\mathbf{1}^T \mathbf{y}}{\mathbf{1}^T \mathbf{z}} < 2^{s+1}$  in this phase. Equivalently, this is saying  $\mathbf{a}_i \leq \frac{2}{3} \mathbf{b}_i$ , so  $i \in B$  in all previous  $\Omega(\ln n \ln(\frac{m}{\epsilon})/\epsilon)$  iterations of the phase, and  $\Delta_i \geq \frac{1}{6}$  in all those iterations.

Each iteration the multiplicative update on  $\mathbf{x}_i$  is  $(1 + \alpha \Delta_i)$ , which is  $(1 + \Theta(\frac{\epsilon}{10 \ln n}))$  since  $\Delta_i \geq \frac{1}{6}$ . As  $\mathbf{x}_i$  starts with  $\frac{1}{m \|P_{:,i}\|_\infty}$ , after  $100 \ln n \ln(\frac{m}{\epsilon})/\epsilon$  updates, we have  $\mathbf{x}_i \gg \frac{10 \ln n}{\epsilon \|P_{:,i}\|_\infty}$ , which gives  $\max_j (\mathbf{P} \mathbf{x})_j \gg \frac{10 \ln n}{\epsilon}$ , so the algorithm must have terminated. ◀

The above lemma guarantees that all iterations after the first  $100 \ln n \ln(\frac{m}{\epsilon})/\epsilon$  must be good iterations, so we proceed to bound the number of these good iterations in a single phase. Without loss of generality, we index these good iterations in a phase as  $1, \dots, T$  by shifting  $t$ .

We first identify one variable that must be updated extensively in these iterations.

► **Lemma 9.** *Suppose the instance (4) is feasible, then There exists  $i \in [m]$  such that*

$$\sum_{t=1}^T \mathbf{b}_i^{(t)} - \mathbf{a}_i^{(t)} \geq 10\epsilon \sum_{t=1}^T \mathbf{b}_i^{(t)}. \quad (8)$$

**Proof.** Define  $\bar{\mathbf{y}}$  and  $\bar{\mathbf{z}}$  to be the sum of the normalized gradients of iterations  $1, \dots, T$ , that is,

$$\bar{\mathbf{y}} = \sum_{t=1}^T \frac{\mathbf{y}^{(t)}}{\mathbf{1}^T \mathbf{y}^{(t)}}, \quad \bar{\mathbf{z}} = \sum_{t=1}^T \frac{\mathbf{z}^{(t)}}{\mathbf{1}^T \mathbf{z}^{(t)}}.$$

Note  $\mathbf{1}^T \bar{\mathbf{y}} = \mathbf{1}^T \bar{\mathbf{z}} = T$ . Recall  $\mathbf{a}_i^{(t)}$  and  $\mathbf{b}_i^{(t)}$  are respectively  $\frac{(\mathbf{P}^T \mathbf{y}^{(t)})_i}{\mathbf{1}^T \mathbf{y}^{(t)}}$  and  $\frac{(\mathbf{C}^T \mathbf{z}^{(t)})_i}{\mathbf{1}^T \mathbf{z}^{(t)}}$ , then

$$\sum_{t=1}^T \mathbf{a}_i^{(t)} = \frac{T(\mathbf{P}^T \bar{\mathbf{y}})_i}{\mathbf{1}^T \bar{\mathbf{y}}}, \quad \sum_{t=1}^T \mathbf{b}_i^{(t)} = \frac{T(\mathbf{C}^T \bar{\mathbf{z}})_i}{\mathbf{1}^T \bar{\mathbf{z}}}.$$

Assume by contradiction,  $\forall i \in [m]$ ,  $\sum_{t=1}^T \mathbf{a}_i^{(t)} > (1 - 10\epsilon) \sum_{t=1}^T \mathbf{b}_i^{(t)}$ , that is,

$$\frac{\mathbf{P}^T \bar{\mathbf{y}}}{\mathbf{1}^T \bar{\mathbf{y}}} > (1 - 10\epsilon) \frac{\mathbf{C}^T \bar{\mathbf{z}}}{\mathbf{1}^T \bar{\mathbf{z}}}.$$

By Lemma 1,  $\bar{\mathbf{y}}, \bar{\mathbf{z}}$  certify infeasibility of the instance (4), which contradicts the assumption. ◀

The above claim gives us a variable that on average has smaller packing gradients than covering gradients in this iteration. Together with the property we have on the good iterations (7), we can bound the number of good iterations.

► **Lemma 10.** *In a single phase, the number of good iterations is at most  $O(\ln n \ln(\frac{m}{\epsilon})/\epsilon^2)$ .*

**Proof.** Let  $\mathbf{x}_i$  be a variable satisfying Eqn. (8). We want to turn Eqn. (8) into some lower bound on the total multiplicative update on  $\mathbf{x}_i$  through these iterations. Intuitively, a bad case is that in some iteration  $t$ ,  $\mathbf{a}_i^{(t)}$ ,  $\mathbf{b}_i^{(t)}$  are much larger than the values in other iterations, since they can dominate the terms from other iterations in Eqn. (8), but not much to the total update of  $\mathbf{x}_i$ , since their ratio is what matters to the update. However, since we are inside one single phase, and only looking at good iterations, we can show the bad scenario will not show up.

Formally, let  $l = \mathbf{a}_i^{(1)}$  and  $u = \mathbf{b}_i^{(1)}$ . Since  $(\mathbf{P}^T \mathbf{y})_i$  monotonically increases, and  $\mathbf{1}^T \mathbf{y}$  will increase but not by more than a factor of 2 in a phase, we have

$$\mathbf{a}_i^{(t)} = \frac{(\mathbf{P}^T \mathbf{y}^{(t)})_i}{\mathbf{1}^T \mathbf{y}^{(t)}} \geq l/2 \quad \forall t = 1, \dots, T. \quad (9)$$

Similarly, we have

$$\mathbf{b}_i^{(t)} = \frac{(\mathbf{C}^T \mathbf{z}^{(t)})_i}{\mathbf{1}^T \mathbf{z}^{(t)}} \leq 2u \quad \forall t = 1, \dots, T. \quad (10)$$

Furthermore, since we are looking at the good iterations, we have

$$l \geq \frac{1}{3}u. \quad (11)$$

The inequalities above allow us to turn the difference-based guarantee from Eqn. (8) into lower bounds on ratios we need.

By the update (6), we have

$$\Delta_i^{(t)} \geq \frac{(1 - \frac{\epsilon}{50})\mathbf{b}_i^{(t)} - \mathbf{a}_i^{(t)}}{2\mathbf{b}_i^{(t)}}.$$

So we can lower bound the total update on  $\mathbf{x}_i$  as follows

$$\begin{aligned} \mathbf{x}_i^{(T)} &\geq \mathbf{x}_i^{(1)} \exp\left(\frac{\alpha \sum_t \Delta_i^{(t)}}{2}\right) \\ &= \mathbf{x}_i^{(1)} \exp\left(\frac{\alpha}{4} \sum_t \frac{(1 - \frac{\epsilon}{50})\mathbf{b}_i^{(t)} - \mathbf{a}_i^{(t)}}{\mathbf{b}_i^{(t)}}\right) \\ &\geq \mathbf{x}_i^{(1)} \exp\left(\frac{\alpha}{4} \sum_t \left(\frac{\mathbf{b}_i^{(t)} - \mathbf{a}_i^{(t)}}{2u} - \frac{\epsilon}{50}\right)\right) \end{aligned}$$

where we used (10) in the last line.

From Eqn. (8), we have

$$\begin{aligned} \sum_t \mathbf{b}_i^{(t)} - \mathbf{a}_i^{(t)} &\geq 10\epsilon \sum_i \mathbf{b}_i^{(t)} \\ &\geq \frac{10\epsilon}{1 - 10\epsilon} \sum_t \mathbf{a}_i^{(t)} \\ &\geq \frac{\epsilon u T}{1 - 10\epsilon} \geq \epsilon u T. \end{aligned}$$

## 52:12 Approximating the Solution to Mixed Packing and Covering LPs

The first two lines both follow from Eqn. (8), the next line follows from  $\mathbf{a}_i^{(t)} \geq l/2 \geq u/6$ .

Thus

$$\mathbf{x}_i^{(T)} \geq \mathbf{x}_i^{(1)} \exp\left(\frac{\epsilon\alpha T}{8} - \frac{\epsilon\alpha T}{200}\right) \geq \frac{1}{m\|\mathbf{P}_{:,i}\|_\infty} \exp\left(\frac{\epsilon\alpha T}{10}\right).$$

If  $T \geq \frac{100 \ln n \ln \frac{m}{\epsilon}}{\epsilon^2} \geq \frac{100 \ln \frac{m}{\epsilon}}{\epsilon\alpha}$ , we have  $\mathbf{x}_i^{(T)} \gg \frac{10 \ln n}{\epsilon\|\mathbf{P}_{:,i}\|_\infty}$ . So the algorithm must have terminated since  $\max_j (\mathbf{P}\mathbf{x})_j \gg \frac{10 \ln n}{\epsilon}$ . ◀

Lemma 8 and Lemma 10 bound the total number of iterations in a phase by  $\tilde{O}(\frac{1}{\epsilon^2})$ , together with the bound on the number of phases, which is  $\tilde{O}(\frac{1}{\epsilon})$ , we guarantee the total number of iterations in Algorithm 1 is  $\tilde{O}(\frac{1}{\epsilon^3})$  if the LP in (4) is feasible.

► **Theorem 11.** *Algorithm 1 solves the  $(1+\epsilon)$ -feasibility problem correctly. It runs in parallel time  $\tilde{O}(1/\epsilon^3)$  with the total work  $\tilde{O}(N/\epsilon^3)$ , where  $N$  is the number of non-zero entries in the constraint matrix.*

**Proof.** The correctness and convergence follows from the lemmas in the prior sections. We only need to look at the running time and total work.

At each iteration, we compute all updated values in  $O(\log N)$  parallel time. Since the total number of iterations is  $\tilde{O}(\frac{1}{\epsilon^3})$ , the algorithm terminates in parallel time  $\tilde{O}(\frac{1}{\epsilon^3})$ .

To see the total work, consider the following implementation. For each  $i \in [m]$ , we maintain  $\mathbf{P}_{ji}\mathbf{x}_i$  if  $\mathbf{P}_{ij} \neq 0$ ; similarly we maintain  $\mathbf{C}_{ji}\mathbf{x}_i$  if  $\mathbf{C}_{ij} \neq 0$ . Besides, we maintain the values of  $\mathbf{y}, \mathbf{z}, \mathbf{P}^T\mathbf{y}, \mathbf{C}^T\mathbf{z}, \mathbf{1}^T\mathbf{y}$  and  $\mathbf{1}^T\mathbf{z}$ . When we update  $\mathbf{x}_i$ , we update these values accordingly, with work proportional to the number of non-zero entries in the  $i$ th column of the constraint matrix. For each fixed variable  $\mathbf{x}_i$ , the total time of updates is at most  $\tilde{O}(\frac{1}{\epsilon^2})$ . Thus, the work on this part is  $\tilde{O}(\frac{N}{\epsilon^2})$ . Additionally, we need to compute the  $\mathbf{a}_i, \mathbf{b}_i$  for all variables at the beginning of each iteration to determine which variables to update, this takes  $\tilde{O}(N)$  work each iteration, so the total work is  $\tilde{O}(\frac{N}{\epsilon^3})$ . ◀

► **Remark.** We see the majority of the work is actually on computing the gradients for the variables we may not update. We point out that we can implement the same lazy update as in [19], which on a high level is just that if a variable has a large  $\frac{\mathbf{a}_i}{\mathbf{b}_i}$  in an iteration, and is not updated, we do not recompute its gradients, until  $\frac{\mathbf{1}^T\mathbf{y}}{\mathbf{1}^T\mathbf{z}}$  grows by more than a factor of  $1+\epsilon$ . This can reduce the work to  $\tilde{O}(\frac{N}{\epsilon^2})$ , but requires a centralized step to control the phases. We omit the details as it is a straightforward adaptation.

### 4 Pure Packing and Pure Covering LPs

We point out that in the case of pure packing or pure covering LPs, Algorithm 1 converges in  $\tilde{O}(\frac{1}{\epsilon^2})$  iterations. This improves upon the result of [17], since our algorithm is deterministic, and does not need centralized steps.

The analysis follows the same approach as the mixed case, but the result of Lemma 10 can be improved to bound the total number of good iterations over all phases by  $O(\ln n \ln(\frac{m}{\epsilon})/\epsilon^2)$ , since for pure packing (or pure covering) LPs, the  $\mathbf{a}_i$ 's (or the  $\mathbf{b}_i$ 's) are constants through the algorithm. We omit the details as the proof is a straightforward adaptation.

**Acknowledgments.** We thank Richard Peng for helpful discussions on results and writing of the paper. DW was supported by ARO Grant W911NF-12-1-0541, SR was funded by NSF Grant CCF-1118083, and MM acknowledges the support of the NSF, AFOSR, and DARPA.

## References

- 1 Zeyuan Allen-Zhu and Lorenzo Orecchia. Nearly-linear time positive LP solver with faster convergence rate. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC'15, pages 229–236, 2015. Newer version available at <http://arxiv.org/abs/1411.1124>.
- 2 Zeyuan Allen-Zhu and Lorenzo Orecchia. Using optimization to break the epsilon barrier: A faster and simpler width-independent algorithm for solving positive linear programs in parallel. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'15, pages 1439–1456, 2015.
- 3 Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(6):121–164, 2012. URL: <http://www.theoryofcomputing.org/articles/v008a006>, doi:10.4086/toc.2012.v008a006.
- 4 Baruch Awerbuch and Rohit Khandekar. Stateless distributed gradient descent for positive linear programs. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 691–700, 2008.
- 5 S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
- 6 Lisa Fleischer. A fast approximation scheme for fractional covering problems with variable upper bounds. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 1001–1010, 2004.
- 7 Christos Koufogiannakis and Neal E. Young. A nearly linear-time PTAS for explicit fractional packing and covering linear programs. *Algorithmica*, 70(4):648–674, 2014. doi:10.1007/s00453-013-9771-6.
- 8 Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in  $\tilde{O}(\sqrt{\text{rank}})$  iterations and faster algorithms for maximum flow. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 424–433, 2014.
- 9 Yin Tat Lee and Aaron Sidford. Efficient inverse maintenance and faster algorithms for linear programming. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 230–249, 2015.
- 10 Michael Luby and Noam Nisan. A parallel approximation algorithm for positive linear programming. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 448–457, 1993.
- 11 Faraz Makari Manshadi, Baruch Awerbuch, Rainer Gemula, Rohit Khandekar, Julián Mestre, and Mauro Sozio. A distributed algorithm for large-scale generalized matching. *PVLDB*, 6(9):613–624, 2013. Available at <http://www.vldb.org/pvldb/vol6/p613-makarimanshadi.pdf>.
- 12 Arkadi Nemirovski. Prox-method with rate of convergence  $O(1/t)$  for variational inequalities with Lipschitz continuous monotone operators and smooth convex-concave saddle point problems. *SIAM Journal on Optimization*, 15(1):229–251, 2004. doi:10.1137/S1052623403425629.
- 13 Yurii Nesterov. Smooth minimization of non-smooth functions. *Math. Program.*, 103(1):127–152, 2005. doi:10.1007/s10107-004-0552-5.
- 14 Yurii Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012. doi:10.1137/100802001.
- 15 Richard Peng and Kanat Tangwongsan. Faster and simpler width-independent parallel algorithms for positive semidefinite programming. In *Proceedings of the 24th ACM symposium on Parallelism in algorithms and architectures, SPAA '12*, pages 101–108, 2012. Available at <http://arxiv.org/abs/1201.5135>.

- 16 James Renegar. Efficient first-order methods for linear programming and semidefinite programming. *CoRR*, abs/1409.5832, 2014. URL: <http://arxiv.org/abs/1409.5832>.
- 17 Di Wang, Michael W. Mahoney, Nishanth Mohan, and Satish Rao. Faster parallel solver for positive linear programs via dynamically-bucketed selective coordinate descent. *CoRR*, abs/1511.06468, 2015. URL: <http://arxiv.org/abs/1511.06468>.
- 18 Neal E. Young. Sequential and parallel algorithms for mixed packing and covering. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 538–546, 2001.
- 19 Neal E. Young. Nearly linear-time approximation schemes for mixed packing/covering and facility-location linear programs. *CoRR*, abs/1407.3015, 2014. URL: <http://arxiv.org/abs/1407.3015>.
- 20 Zeyuan Allen Zhu, Yin Tat Lee, and Lorenzo Orecchia. Using optimization to obtain a width-independent, parallel, simpler, and faster positive SDP solver. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1824–1831, 2016. Available at <http://arxiv.org/abs/1507.02259>.
- 21 Zeyuan Allen Zhu and Lorenzo Orecchia. Linear coupling: An ultimate unification of gradient and mirror descent. *CoRR*, abs/1407.1537, 2014. URL: <http://arxiv.org/abs/1407.1537>.