Beating Simplex for Fractional Packing and Covering Linear Programs

(corrected version of FOCS 2007 paper, October 27, 2013)

Christos Koufogiannakis
University of California, Riverside

Neal E. Young
University of California, Riverside

Abstract

We give an approximation algorithm for packing and covering linear programs (linear programs with non-negative coefficients). Given a constraint matrix with n non-zeros, r rows, and c columns, the algorithm (with high probability) computes feasible primal and dual solutions whose costs are within a factor of $1+\varepsilon$ of OPT *(the optimal cost) in time $O(n+(r+c)\log(n)/\varepsilon^2)$.*

For dense problems (with $r,c = O(\sqrt{n})$) the time is $O(n+\sqrt{n}\log(n)/\varepsilon^2)$ — linear even as $\varepsilon \to 0$. In comparison, previous Lagrangian-relaxation algorithms generally take at least $\Omega(n\log(n)/\varepsilon^2)$ time, while (for small ε) the Simplex algorithm typically takes at least $\Omega(n\min(r,c))$ time.

1. Introduction

A *packing* problem is a linear program of the form $\max\{a \cdot x : Mx \le b, x \in P\}$, where the entries of the constraint matrix M are non-negative and P is a convex polytope admitting some form of optimization oracle. A *covering* problem is of the form $\min\{a \cdot \hat{x} : M\hat{x} \ge b, \hat{x} \in P\}$.

Here we focus on the explicitly given forms, that is, $\max\{a \cdot x : Mx \le b, x \ge 0\}$ and $\min\{a \cdot \hat{x} : M\hat{x} \ge b, \hat{x} \ge 0\}$, when the polytope P is the positive orthant. Explicitly given packing and covering are important special cases of linear programming, including, for example, fractional set cover, multicommodity flow problems with given paths, and variants of these problems.

We give a $(1 \pm \varepsilon)$-approximation algorithm — that is, an algorithm that returns feasible primal and dual solutions whose costs are within a given factor $1 \pm \varepsilon$ of OPT. The algorithm is inherently randomized. With high probability, it runs in time $O(n+(r+c)\log(n)/\varepsilon^2)$, where n is the number of non-zero entries in the constraint matrix and $r+c$ is the number of rows plus columns (i.e., constraints plus variables).

For even moderately dense problems, r and c are $o(n)$ and the $1/\varepsilon^2$ term in the running time is multiplied by only a sub-linear term.

The run time is linear if $\varepsilon \ge \Omega(\sqrt{(r+c)\log(n)/n})$.

In experiments reported here, our first implementation of the algorithm typically requires at most $12(r+c)\ln(n)/\varepsilon^2 + O(n)$ basic operations on dense instances. For comparison, the GLPK (Gnu Linear Programming Kit) Simplex algorithm typically requires at a minimum about $5\min(r,c)rc$ basic operation (and often many more) to compute a near-optimal solution. (Note: more sophisticated Simplex implementations that maintain sparsity may require only $O(\min(r,c)n)$ operations.) For example, when $\varepsilon = 0.01$, the algorithm is faster than GLPK Simplex by factors of 10-100 for problems with several thousand rows and columns. The speedup grows roughly linearly with rc.

Our first implementation is relatively simple (fewer than a thousand lines of C++, mostly devoted to a fast random-sampling data structure). In contrast to Simplex, the algorithm requires no special techniques to maintain sparsity of the constraint matrix, to deal with numerically ill-conditioned matrices, or to deal with basis cycling.

Related work on Lagrangian-relaxation algorithms. The algorithm is a Lagrangian-relaxation algorithm. There is a large literature on Lagrangian-relaxation algorithms. Bienstock gives an implementation-oriented, operations-research perspective [3]. Arora et al. discuss them from a computer-science perspective, highlighting connections to other fields such as learning theory [2]. Todd places them in the larger context of linear programming in general in his overview [13].

For explicitly given packing and covering, the fastest previous Lagrangian-relaxation algorithm that we know of runs in time $O((r+c)\bar{c}\log(n)/\varepsilon^2)$, where $\bar{c}$ is the maximum number of columns in which any variable appears [15]. That algorithm works for mixed packing and covering — a more general problem. One can improve that algorithm to run in time $O(n\log(n)/\varepsilon^2)$ (an unpublished result), but this is still impractically slow for small ε and large n.

For $U \doteq \max_{ij} M_{ij}/(b_i a_j)$, Grigoriadis and Khachiyan give an $O((r+c)\log(n)(U\,\text{OPT})^2/\varepsilon^2)$-time algorithm [7]. A pre-processing step [11, §2.1] can ensure that this is at

48th Annual IEEE Symposium on Foundations of Computer Science, 494-504 (2007)

most $O((r+c)\log(n)(\min(r,c)/\varepsilon)^4)$. Although impractical for general packing and covering problems, the algorithm is interesting in the following special case: given a two-player zero-sum matrix game with payoffs in $[-1,1]$, one wants to find mixed strategies that guarantee an expected payoff within an additive ε_+ of optimal. For this case the time is $O((r+c)\log(n)/\varepsilon_+^2)$, which can be *sublinear* in the input size. Their result uses an unusual technique of coupling primal and dual algorithms, a technique that is central to our algorithm also.

Tradeoffs in the dependence on $1/\varepsilon$. Recent works reduce the dependence on ε to $O(1/\varepsilon)$ for some packing and covering problems. Bienstock and Iyengar give an algorithm for concurrent multicommodity flow that solves $O^*(\varepsilon^{-1}k^{1.5}|V|^{0.5})$ shortest-path problems, where k is the number of commodities and $|V|$ is the number of vertices [4]. Chudak and Eleuterio continue this direction. For example, they give an algorithm for fractional set cover running in time $O^*(c^{1.5}(r+c)/\varepsilon + c^2 r)$ [5]. This direction is motivated by the observation that in practice, even for moderately small ε, the $1/\varepsilon^2$ factor in the running time makes previous algorithms impractically slow for large n. However, the decreased dependence on ε comes at the cost of increasing the dependence on other parameters.

Note that the approximation parameter ε plays a different role than the problem-size parameters — as computing power and problem sizes grow, the case when ε is a small constant (on the order of a fraction of a percent, say) will still be of interest. More concretely, compare the $O^*(c^{1.5}(r+c)/\varepsilon)$ term in the running time of Chudak et al.'s algorithm to the $O^*((r+c)/\varepsilon^2)$ term in the running time of the algorithm here. The former term is smaller only for $\varepsilon \le 1/c^{1.5}$, but for ε this small both algorithms take at least $\Omega^*(c^3(r+c))$ time — slower than alternatives such as Simplex. Arguably, neither algorithm is of practical interest for ε so small.

In sum, both theoretically and practically a main case of interest is when $1/\varepsilon$ is a moderately large constant (a hundred to a thousand), while the number of rows and columns still grows asymptotically (beyond several thousand or more). In this case we prefer an $O(1/\varepsilon^2)$ term in the running time to, say, an $O(c/\varepsilon)$ term. An important goal is to design algorithms that are practical for ε in this range.

Technical approach. The following lower bound shows that some dependence on $1/\varepsilon^2$ is necessary [9]. *With high probability, a packing problem with random matrix $M \in \{0,1\}^{c^2\times c}$ has no $(1-\varepsilon)$-approximate primal solution with $o(\log(n)/\varepsilon^2)$ non-zero entries.* (This lower bound only holds when ε is not too small — roughly $\Omega(\sqrt{c})$ — which is why it does not preclude algorithms with $o(1/\varepsilon^2)$ dependence on ε but larger dependence on other parameters.)

The lower bound implies that to build a $(1-\varepsilon)$-approximate solution requires at least $\Omega(\log(n)/\varepsilon^2)$ variables to be incremented (set to a non-zero value). Most algorithms take at least $\Omega(n)$ amortized time per increment, leading to total time at least $\Omega(n\log(n)/\varepsilon^2)$. From this perspective, the challenge is to reduce the time per increment as much as possible.

To do this we use the following main ideas. We start with a variant of Grigoriadis and Khachiyan's algorithm [7], which is essentially as follows. It starts with all-zero primal and dual solutions. In each iteration, one coordinate x_j of the primal solution is increased by 1, where j is randomly chosen from a distribution $\hat{p}$ that depends on $M^{\mathsf{T}}\hat{x}$, where $\hat{x}$ is the current *dual* solution. Likewise, a coordinate $\hat{x}_i$ of the dual solution is increased by 1, where i is chosen at random from a distribution p that depends on Mx, where x is the current primal solution.

(This algorithm can be interpreted as a form of fictitious play of a two-player zero-sum game, where in each round each player plays from a distribution concentrated around the best response to the aggregate of the opponent's historical plays.)

This random choice of increments can be implemented more efficiently than the more common approach of finding "optimal" increments. The reason is that the latter approach requires choosing an increment Δx to the primal solution to minimize $p^{\mathsf{T}}M\Delta x$, and separately an increment $\Delta\hat{x}$ to the dual solution to maximize $\hat{p}^{\mathsf{T}}M^{\mathsf{T}}\Delta\hat{x}$. The difficulty with this is that it requires maintaining the additional vectors $p^{\mathsf{T}}M$ and $\hat{p}^{\mathsf{T}}M^{\mathsf{T}}$ instead of just p and $\hat{p}$. (A change in one entry of x changes many entries in p, but even more entries in $p^{\mathsf{T}}M$. We are able to bound the total number of changes to entries in p by $O(r\log(n)/\varepsilon^2)$, but not so for changes to entries in $p^{\mathsf{T}}M$.)

This basic ("slow") algorithm is analyzed in Lemma 1. To speed it up (and generalize it), we incorporate Garg and Könemann's non-uniform-increments amortization scheme [6]. In each iteration, we make the algorithm increment the primal and dual variables not by 1, but by an amount just large enough so that some left-hand side (LHS) M_ix or $M_j^{\mathsf{T}}\hat{x}$ increases by $\Omega(1)$. This is small enough to allow the correctness proof to go through, but large enough to guarantee progress.

Finally, instead of maintaining Mx and $M^{\mathsf{T}}\hat{x}$ exactly, we maintain them approximately with a careful random sampling. This way, when some M_ix increases by a relatively small amount in an iteration (because some x_j increases where M_{ij} is relatively small), we have only a proportionally small chance of doing work to update the estimate of M_ix. Yet the estimates are still accurate in expectation and with high probability.

Sections 2, 2.1 and 2.2 give the main algorithm, prove correctness, and bound the worst-case run time, respectively. Section 3 presents our experimental results, including a comparison with the GLPK Simplex algorithm.

2. Algorithm and analysis

In the rest of the paper we assume the primal and dual problems are of the following restricted forms, respectively: $\max\{|x| : Mx \le \mathbf{1}, x \ge 0\}$, $\min\{|\hat{x}| : M^{\mathsf T}\hat{x} \ge \mathbf{1}, \hat{x} \ge 0\}$. This is without loss of generality by the transformation $M'_{ij} = M_{ij}/(b_i a_j)$.

Slow algorithm. For explanatory purposes, we first give a simpler but slow version of the algorithm, essentially a variant of Grigoriadis and Khachiyan's algorithm [7]. For this we assume $M_{ij} \in [0,1]$ and we don't analyze the running time, which can be large. This algorithm demonstrates the use of coupled random increments to the primal and dual. It returns a $(1-2\varepsilon)$-approximate primal-dual pair with high probability.

```
slow-alg($M \in [0,1]^{r\times c}, \varepsilon$)
Vectors $x, \hat{x} \leftarrow \mathbf{0}$; scalar $N = \lceil 2\ln(rc)/\varepsilon^2 \rceil$.
Repeat until $\max_i M_i x \ge N$:
    Let $p_i \doteq (1+\varepsilon)^{M_i x}$ and $\hat{p}_j \doteq (1-\varepsilon)^{M_j^{\mathsf T}\hat{x}}$.
    Choose random indices $j'$ and $i'$ respectively
        from probability distributions $\hat{p}/|\hat{p}|$ and $p/|p|$.
    Increase $x_{j'}$ and $\hat{x}_{i'}$ each by 1.
Let $(x^\star, \hat{x}^\star) \doteq (x/\max_i M_i x, \hat{x}/\min_j M_j^{\mathsf T}\hat{x})$.
Return $(x^\star, \hat{x}^\star)$.
```

Note that the scaling at the end ensures feasibility. Also, $|x|$ and $|\hat{x}|$ are always equal, so to prove the approximation guarantee we show $\max_i M_i x$ is not too large in comparison to $\min_j M_j^{\mathsf T}\hat{x}$ (we want $\min_j M_j^{\mathsf T}\hat{x} \ge (1 - O(\varepsilon))\max_i M_i x$ at the end). To show this, we show that $|p| \cdot |\hat{p}|$ (the product of the 1-norms) is a Lyapunov function for the system — that it is non-increasing in expectation.

Lemma 1 *The slow algorithm returns a $(1-2\varepsilon)$-approximate primal-dual pair (feasible primal and dual solutions $x^\star$ and $\hat{x}^\star$ such that $|x^\star| \ge (1-2\varepsilon)|\hat{x}^\star|$) with probability at least* $1 - 1/(rc)$.

proof: In a given iteration, let p and $\hat{p}$ denote the vectors at the start of the iteration. Let p' and $\hat{p}'$ denote the vectors at the end of the iteration. Let Δx denote the vector whose jth entry is the increase in x_j during the iteration (or if z is a scalar, Δz denotes the increase in z). Then (using that $\Delta M_i x = M_i \Delta x \in [0,1]$)

$$\begin{aligned} |p'| &= \sum_i p_i(1+\varepsilon)^{M_i\Delta x} \\ &\le \sum_i p_i(1+\varepsilon M_i \Delta x) \\ &= |p|[1+\varepsilon p^{\mathsf T} M\Delta x/|p|]. \end{aligned}$$

Likewise $|\hat{p}'| \le |\hat{p}|[1 - \varepsilon\hat{p}^{\mathsf T} M^{\mathsf T}\Delta\hat{x}/|\hat{p}|]$.

Multiplying these bounds on $|p'|$ and $|\hat{p}'|$, and using that $(1+a)(1-b) = 1+a-b-ab \le 1+a-b$ for $a, b \ge 0$,

$$|p'||\hat{p}'| \;\le\; |p||\hat{p}|\big[\,1 + \varepsilon(p/|p|)^{\mathsf T} M\Delta x - \varepsilon\Delta\hat{x}^{\mathsf T} M(\hat{p}/|\hat{p}|)\,\big].$$

This inequality motivates the "coupling" of primal and dual.

Taking expectations and plugging in $\mathrm{E}[\Delta x] = \hat{p}/|\hat{p}|$ and $\mathrm{E}[\Delta\hat{x}] = p/|p|$ from the definition of the algorithm, the right-hand terms cancel and we get $\mathrm{E}[|p'||\hat{p}'|] \le |p||\hat{p}|$.

This and Wald's equation (Lemma 9) imply that the expectation of $|p||\hat{p}|$ at termination is at most its initial value rc. So, by the Markov bound, with probability at least $1 - 1/rc$, at termination $|p||\hat{p}| \le (rc)^2$. Assume this happens. Then, for all i and j, $(1+\varepsilon)^{M_i x}(1-\varepsilon)^{M_j^{\mathsf T}\hat{x}} \le (rc)^2$.

Taking logs gives $(1-\varepsilon)\max_i M_i x \le \min_j M_j^{\mathsf T}\hat{x} + \varepsilon N$. (See the proof of Theorem 1 or Lemma 10 for details.)

By the termination condition $\max_i M_i x \ge N$, so this implies $(1-2\varepsilon)\max_i M_i x \le \min_j M_j^{\mathsf T}\hat{x}$.

This and $|x| = |\hat{x}|$ imply the performance guarantee. □

Full algorithm. In the remainder of the paper we analyze the full algorithm. The algorithm is in Fig. 1, except for some implementation details that are left until Section 2.2. In comparison to the slow algorithm, the algorithm incorporates the following additional features:

- It uses non-uniform increments. That is, in each iteration it increases the randomly chosen $x_{j'}$ and $\hat{x}_{i'}$ by some increment $\delta_{i'j'}$, chosen so that the maximum increase in any left-hand side (LHS) (i.e. $\max_i \Delta M_i x$ or $\max_j \Delta M_j^{\mathsf T}\hat{x}$) is $\Theta(1)$. It also deletes covering constraints once they become satisfied. (These basic ideas are from [6, 10].)

  It adjusts the sampling distributions accordingly to maintain that the expected changes still satisfy $\mathrm{E}[\Delta x] = \alpha\hat{p}/|\hat{p}|$ and $\mathrm{E}[\Delta\hat{x}] = \alpha p/|p|$ for an $\alpha > 0$.

  Implementing this requires maintaining the following data structures: a set J of indices of still-active columns (covering constraints); for each column $M_j^{\mathsf T}$ the maximum entry u_j; and for each row M_i, a close upper bound $\hat{u}_i$ on the maximum active entry.

```
solve($M \in \mathbb{R}_+^{r \times c}, \varepsilon$) — return a $(1-6\varepsilon)$-approximate primal-dual pair w/ high probability.
Initialize vectors $x, \hat{x}, y, \hat{y} \leftarrow \mathbf{0}$, and scalar $N = \lceil 2\ln(rc)/\varepsilon^2 \rceil$.
Fix $u_j \doteq \max\{M_{ij} : i \in [r]\}$ for $j \in [c]$. (This is the largest entry in column $j$ of $M$.)
The algorithm will increment $x$ and $\hat{x}$, maintaining $y$ and $\hat{y}$ so $\mathrm{E}[y] = Mx$, $\mathrm{E}[\hat{y}] = M^{\intercal}\hat{x}$.
It will also maintain vectors $p$ defined by $p_i \doteq (1+\varepsilon)^{y_i}$ and, as a function of $\hat{y}$:

    $J \doteq \{j \in [c] : \hat{y}_j \le N\}$                       (the active columns)
    $\hat{u}_i \in [1,2] \times \max\{M_{ij} : j \in J\}$          (approximates the largest active entry in row $i$ of $M$)
    $\hat{p}_j \doteq \begin{cases} (1-\varepsilon)^{\hat{y}_j} & \text{if } j \in J \\ 0 & \text{otherwise.} \end{cases}$

It will also maintain vectors $p \times \hat{u}$ and $\hat{p} \times u$.
Repeat until $\max_i y_i = N$ or $\min_j \hat{y}_j = N$:
    Let $(i', j') \leftarrow$ random-pair($p, \hat{p}, p \times \hat{u}, \hat{p} \times u$).
    Increase $x_{j'}$ and $\hat{x}_{i'}$ each by the same amount $\delta_{i'j'} \doteq 1/(\hat{u}_{i'} + u_{j'})$.
    Update $y, \hat{y}$, and the other vectors as follows: choose random $z \in [0,1]$ uniformly, and
        for each $i \in [r]$ such that $M_{ij'}\delta_{i'j'} \ge z$, increase $y_i$ by 1 (and multiply $p_i$ and $(p \times \hat{u})_i$ by $1+\varepsilon$);
        for each $j \in J$ such that $M_{i'j}\delta_{i'j'} \ge z$, increase $\hat{y}_j$ by 1 (and multiply $\hat{p}_j$ and $(\hat{p} \times u)_j$ by $1-\varepsilon$).
    For each $j$ leaving $J$, update $J$, $\hat{u}$, and $p \times \hat{u}$.
Let $(x^\star, \hat{x}^\star) \doteq (x/\max_i M_i x, \hat{x}/\min_j M_j^{\intercal}\hat{x})$. Return $(x^\star, \hat{x}^\star)$.
```

Figure 1. The full algorithm. Notation: $p \times \hat{u}$ denotes a vector with ith entry $p_i\hat{u}_i$; $[c]$ denotes $\{1, 2, \ldots, c\}$. Implementation details are in Section 2.2.

The increment $\delta_{i'j'}$ is taken to be $1/(\hat{u}_{i'} + u_{j'})$, so that when $x_{j'}$ and $\hat{x}_{i'}$ are increased by $\delta_{i'j'}$, the maximum increase in any LHS (any $M_i x$ or $M_j^{\intercal}\hat{x}$) is in $[1/4, 1]$.

- It maintains the vectors Mx and $M^{\intercal}\hat{x}$ only approximately (as y and $\hat{y}$), using a simple random sampling. For example, if $M_i x$ increases by $1/10$ in an iteration, then the algorithm increases y_i by 1 with probability $1/10$. This reduces the total work, yet maintains $y \approx Mx$ and $\hat{y} \approx M^{\intercal}\hat{x}$ with high probability.

  To implement this the algorithm chooses a random $z \in [0,1]$. It then increments y_i by 1 if the increase in $M_i x$ is at least z, and increments $\hat{y}_i$ by 1 if the increase in $M_j^{\intercal}\hat{x}$ is at least z. To do this efficiently, rows and columns are first internally sorted (or approximately sorted) in decreasing order.

With these changes, we bound the running time by charging the work done to increases in $|Mx| + |M^{\intercal}\hat{x}|$.

The following subroutine **random-pair()** chooses the random pair of indices (i', j'). Any given pair (i, j) is chosen with probability proportional to $p_i\hat{p}_j(\hat{u}_i + u_j) = p_i\hat{p}_j/\delta_{ij}$.

```
random-pair($p, \hat{p}, p \times \hat{u}, \hat{p} \times u$)
With probability $|p \times \hat{u}||\hat{p}|/(|p \times \hat{u}||\hat{p}| + |p||\hat{p} \times u|)$
    choose random $i'$ from distribution $p \times \hat{u}/|p \times \hat{u}|$,
    and independently choose $j'$ from $\hat{p}/|\hat{p}|$,
or, otherwise,
    choose random $i'$ from distribution $p/|p|$,
    and independently choose $j'$ from $\hat{p} \times u/|\hat{p} \times u|$.
Return $(i', j')$.
```

2.1. Correctness

Theorem 1 *With probability at least $1 - 3/rc$, the algorithm in Fig. 1 returns feasible primal and dual solutions $(x^\star, \hat{x}^\star)$ with $|x^\star|/|\hat{x}^\star| \ge 1 - 6\varepsilon$.*

proof: To start we show the algorithm's basic properties.

Lemma 2 *In each iteration, for $\alpha = |p||\hat{p}|/\sum_{ij} p_i\hat{p}_j/\delta_{ij}$,*

1. *The largest change in any relevant LHS is at least 1/4:* $\max\{\max_i \Delta M_i x, \max_{j \in J} \Delta M_j^{\intercal}\hat{x}\} \in [1/4, 1]$.

2. *The expected changes in various quantities are:*

$$\mathrm{E}[\Delta x] = \alpha\hat{p}/|\hat{p}|, \quad \mathrm{E}[\Delta y] = \mathrm{E}[\Delta Mx] = \alpha M\hat{p}/|\hat{p}|,$$

$$\mathrm{E}[\Delta\hat{x}] = \alpha p/|p|, \quad \mathrm{E}[\Delta\hat{y}] = \mathrm{E}[\Delta M^{\intercal}\hat{x}] = \alpha M^{\intercal}p/|p|.$$

proof: (1) By the choice of $\hat{u}$ and u, for the (i', j') chosen,

$$\begin{aligned} &\max\{\max_i M_{ij'}\delta_{i'j'}, \max_{j\in J} M_{i'j}\delta_{i'j'}\} \\ \in\quad & [1/2, 1]\delta_{i'j'} \max\{\hat{u}_{i'}, u_{j'}\} \\ \subseteq\quad & [1/4, 1]\delta_{i'j'}(\hat{u}_{i'} + u_{j'}) \\ =\quad & [1/4, 1]. \end{aligned}$$

(2) First, we verify that the probability that random-pair() returns a given (i, j) is $\alpha(p_i/|p|)(\hat{p}_j/|\hat{p}|)/\delta_{ij}$. Here is the calculation. The probability is proportional to

$$|p \times \hat{u}|\,|\hat{p}|\frac{p_i\hat{u}_i}{|p \times \hat{u}|}\frac{\hat{p}_j}{|\hat{p}|} + |p|\,|\hat{p} \times u|\frac{p_i}{p}\frac{\hat{p}_j u_j}{|\hat{p} \times u|}$$

which by algebra simplifies to $p_i\hat{p}_j(\hat{u}_i + u_j) = p_i\hat{p}_j/\delta_{ij}$.

Thus, the probability must be $\alpha(p_i/|p|)(\hat{p}_j/|\hat{p}|)/\delta_{ij}$, because the choice of α makes the sum over all i and j of the probabilities equal 1.

Next, note that part (1) of the lemma implies that in line 8 (given the chosen i' and j') the probability that a given y_i is incremented is $M_{ij'}\delta_{i'j'}$, while in line 9 the probability that a given $\hat{y}_j$ is incremented is $M_{i'j}\delta_{i'j'}$.

Now the equalities in (2) follow by direct calculation. For example:
$\mathrm{E}[\Delta x_j] = \sum_i(\alpha p_i/|p|)(\hat{p}_j/|\hat{p}|)/\delta_{ij})\delta_{ij} = \alpha\hat{p}_j/|\hat{p}|$. □

The next lemma is in the same spirit as Lemma 1. We use the coupling in the algorithm to show that the Lyapunov function $\phi \doteq |p||\hat{p}|$ is non-increasing in expectation and that this and the termination condition imply the approximation guarantee (as long as $y \approx Mx$ and $\hat{y} \approx M^{\mathsf{T}}\hat{x}$).

Lemma 3 *With probability at least* $1 - 1/rc$*, when the algorithm stops,* $\max_i y_i \le N$ *and* $\min_j \hat{y}_j \ge N(1 - 2\varepsilon)$.

proof: Let p' and $\hat{p}'$ denote p and $\hat{p}$ after a given iteration, while p and $\hat{p}$ denote the values before the iteration.

We claim that, given p and $\hat{p}$, $\mathrm{E}[|p'|\,|\hat{p}'|] \le |p|\,|\hat{p}|$ — with each iteration $|p|\,|\hat{p}|$ is non-increasing in expectation.

To prove it, note $|p'| = \sum_i p_i(1 + \varepsilon\Delta y_i) = |p| + \varepsilon p^{\mathsf{T}}\Delta y$ and, similarly, $|\hat{p}'| = |\hat{p}| - \varepsilon\hat{p}^{\mathsf{T}}\Delta\hat{y}$.

Multiply the latter two equations and drop a negative term to get

$$|p'|\,|\hat{p}'| \;\le\; |p|\,|\hat{p}| + \varepsilon|\hat{p}|p^{\mathsf{T}}\Delta y - \varepsilon|p|\hat{p}^{\mathsf{T}}\Delta\hat{y}.$$

The claim follows by applying linearity of expectation, then substituting $\mathrm{E}[\Delta y] = \alpha M\hat{p}/|\hat{p}|$ and $\mathrm{E}[\Delta\hat{y}] = \alpha M^{\mathsf{T}}p/|p|$ from Lemma 2.

By Wald's equation (Lemma 9), the claim implies that $\mathrm{E}[|p|\,|\hat{p}|]$ at termination is at most the initial value rc.

Applying the Markov bound, with probability at least $1 - 1/rc$, at termination $\max_i p_i \max_j \hat{p}_j \le |p||\hat{p}| \le (rc)^2$.

Assume this happens. The index set J is not empty at termination, so the minimum $\hat{y}_j$ is achieved for $j \in J$. Substituting in the definitions of p_i and $\hat{p}_j$ and taking logs, $\max_i y_i \ln(1+\varepsilon) \le \min_j \hat{y}_j \ln(1/(1-\varepsilon)) + 2\ln(rc)$.

Divide by $\ln(1/(1-\varepsilon))$, apply $1/\ln(1/(1-\varepsilon)) \le 1/\varepsilon$ and also $\ln(1+\varepsilon)/\ln(1/(1-\varepsilon)) \ge 1 - \varepsilon$. This gives $(1-\varepsilon)\max_i y_i \le \min_j \hat{y}_j + 2\ln(rc)/\varepsilon \le \min_j \hat{y}_j + \varepsilon N$.

By the termination condition $\max_i y_i \le N$ is guaranteed, and either $\max_i y_i = N$ or $\min_j \hat{y}_j = N$. So if $\min_j \hat{y}_j = N$, then the event in the lemma occurs, and otherwise $\max_i y_i = N$, which (with the inequality in previous paragraph) gives $(1-\varepsilon)N \le \min_j \hat{y}_j + \varepsilon N$. □

Next we establish that $y \approx Mx$ and $\hat{y} \approx M^{\mathsf{T}}\hat{x}$.

Lemma 4
(1) *For any* i*, with probability at least* $1 - 1/(rc)^2$*, when the algorithm terminates,* $(1-\varepsilon)M_i x \le y_i + \varepsilon N$.
(2) *For any* j*, with probability at least* $1 - 1/(rc)^2$*, after the last iteration with* $j \in J$*,* $(1-\varepsilon)\hat{y}_j \le M_j^{\mathsf{T}}\hat{x} + \varepsilon N$.

proof: **(1)** In each iteration $M_i x$ increases by at most 1 (by the choice of $\delta_{i'j'}$), y_i increases by at most 1, and (by Lemma 2) the expected increases in these two quantities are the same. So, by the Chernoff bound for random stopping times (Lemma 10), $\Pr[(1-\varepsilon)M_i x \ge y_i + \varepsilon N]$ is at most $\exp(-\varepsilon^2 N) \le 1/(rc)^2$. This proves (1).

The proof for (2) is similar, noting that, while $j \in J$, $M_j^{\mathsf{T}}\hat{x}$ increases by at most 1 each iteration. □

We now prove Theorem 1. Recall that the algorithm returns $(x^\star, \hat{x}^\star) \doteq (x/\max_i M_i x, \hat{x}/\min_j M_j^{\mathsf{T}}\hat{x})$.

By the naive union bound, with probability at least $1 - 3/rc$ the event in Lemma 3 occurs and (for all i and j) the events in Lemma 4 occur.

Assume all of these events happen.

By algebra, using $(1-a)(1-b) \ge 1 - a - b$ and $1/(1+\varepsilon) \ge 1 - \varepsilon$, we have $(1-2\varepsilon)\max_i M_i x \le N$ and $\min_j M_j^{\mathsf{T}}\hat{x} \ge (1-4\varepsilon)N$.

This implies $\min_j M_j^{\mathsf{T}}\hat{x}/\max_i M_i x \ge 1 - 6\varepsilon$.

Since the sizes $|x|$ and $|\hat{x}|$ increase by the same amount each iteration, they are equal. Thus, $|x^\star|/|\hat{x}^\star| = \min_j M_j^{\mathsf{T}}\hat{x}/\max_i M_i x \ge 1 - 6\varepsilon$. □

2.2. Running time

In this section we describe fast implementations of the algorithm. We assume that the matrix M is given in any standard sparse representation (so that the non-zero entries can be traversed in time proportional to the number of non-zero entries).

Simpler implementation. We first describe an implementation that takes $O(n \log n + (r+c)\log(n)/\varepsilon^2)$ time. We then describe how to modify it to remove the $\log n$ factor from the first term.

Theorem 2 *The algorithm can be implemented to return a $(1-6\varepsilon)$-approximate primal-dual pair for packing and covering in time $O(n \log n + (r+c)\log(n)/\varepsilon^2)$ with probability at least $1 - 4/rc$.*

proof: Use the data structure of [12] (see also [8]), which maintains a vector v and allows random sampling from the distribution $v/|v|$ and updating of entries of v in $O(1)$ time. Maintain such a data structure for each of the vectors p, $\hat{p}$, $p \times \hat{u}$, and $\hat{p} \times u$. Then random-pair() runs in $O(1)$ time, and each update of an entry of p, $\hat{p}$, $p \times \hat{u}$, or $\hat{p} \times u$ takes $O(1)$ time.

At the start, pre-process the matrix M. Build, for each row and column, a doubly linked list of the non-zero entries. Sort each list in descending order. Cross-reference the lists so that, given an entry M_{ij} in the ith row list, the corresponding entry M_{ij} in the jth column list can be found in constant time. The total time for pre-processing is $O(n \log n)$.

Maintain the data structures during each iteration as follows. Let $\mathcal{I}_t$ denote the set of indices i for which y_i is incremented in line 8 in iteration t. From the random $z \in [0,1]$ and the sorted j'th row list, compute this set $\mathcal{I}_t$ by traversing the row list, collecting elements until an i with $M_{ij'} < z/\delta_{i'j'}$ is encountered. Then, for each $i \in \mathcal{I}_t$, update y_i, p_i, and the ith entry in $p \times \hat{u}$. Likewise, let $\mathcal{J}_t$ denote the set of indices j for which $\hat{y}_j$ is incremented in line 9. Compute $\mathcal{J}_t$ from the sorted i'th column list. For each $j \in \mathcal{J}_t$, update $\hat{p}_j$, and the jth entry in $\hat{p} \times u$. The total time for these operations during the course of the algorithm is $O(\sum_t 1 + |\mathcal{I}_t| + |\mathcal{J}_t|)$.

For each element j that leaves J, update $\hat{p}_j$. Delete all entries in the jth column list from all row lists. For each row list i whose first (largest) entry is deleted, update the corresponding $\hat{u}_i$ by setting $\hat{u}_i$ to be the next (now first and maximum) entry remaining in the row list; also update $(p \times \hat{u})_i$. The total time for this during the course of the algorithm is $O(n)$, because each M_{ij} is deleted at most once.

This completes the implementation.

The total time is $O(n \log n)$ plus $O(\sum_t 1 + |\mathcal{I}_t| + |\mathcal{J}_t|)$. To finish we bound the latter term.

Lemma 5
$\sum_t |\mathcal{I}_t| + |\mathcal{J}_t| \le (r+c)N = O((r+c)\log(n)/\varepsilon^2)$.

proof: First, $\sum_t |\mathcal{I}_t| \le rN$ because each y_i can be increased at most N times before $\max_i y_i \ge N$ (causing termination). Second, $\sum_t |\mathcal{J}_t| \le cN$ because each $\hat{y}_j$ can be increased at most N times before j leaves J. □

So the total time is $O(n \log n + (r+c)\log(n)/\varepsilon^2)$ plus $O(\#\, t \text{ such that } |\mathcal{I}_t| + |\mathcal{J}_t| = 0)$. It remains to bound the latter term. Call iteration t *empty* if $|\mathcal{I}_t| + |\mathcal{J}_t| = 0$.

Lemma 6 *Given the state at the start of an iteration, the probability that it is empty is at most 3/4.*

proof: Given the (i', j') chosen in the iteration, by (1) of Lemma 2, there is either an i such that $M_{ij'}\delta_{i'j'} \ge 1/4$ or a j such that $M_{i'j}\delta_{i'j'} \ge 1/4$. In the former case, $i \in \mathcal{I}_t$ with probability at least $1/4$. In the latter case, $j \in \mathcal{J}_t$ with probability at least $1/4$. □

Lemma 7 *With probability at least $1 - 1/rc$, the number of empty iterations is $O((r+c)N)$.*

proof: Let E_t be 1 for empty iterations and 0 otherwise. By the previous lemma and the Chernoff bound tailored for random stopping times (Lemma 10), for any $\delta, A \ge 0$,

$$\Pr\Big[\,(1-\delta)\textstyle\sum_{t=1}^{T} E_t \;\ge\; 3\sum_{t=1}^{T}(1-E_t) \;+\; A\,\Big]$$

is at most $\exp(-\delta A)$. Taking $\delta = 1/2$ and $A = 2\ln(rc)$, it follows that with probability at least $1 - 1/rc$, the number of empty iterations is bounded by a constant times the number of non-empty iterations plus $2\ln(rc)$. We know the number of non-empty iterations is at most $(r+c)N$, so we conclude that with probability at least $1 - 1/rc$ the number of empty iterations is $O((r+c)N)$. □

If the event in Lemma 7 happens, then (since each empty iteration takes $O(1)$ time) the total time is $O(n \log n + (r+c)\log(n)/\varepsilon^2)$. This and Theorem 1 imply Theorem 2. □

Faster implementation. Next we remove the $\log n$ factor from the $n \log n$ term in the running time. The idea is that it suffices to *approximately* sort the row and column lists.

Theorem 3 *The algorithm can be implemented to return a $(1-7\varepsilon)$-approximate primal-dual pair for packing and covering in time $O(n + (c+r)\log(n)/\varepsilon^2)$ with probability at least $1 - 5/rc$.*

proof: Modify the algorithm as follows.

First, pre-process M as described in [11, §2.1] so that the non-zero entries have bounded range. Specifically, let $\beta = \min_j \max_i M_{ij}$. Let $M'_{ij} \doteq 0$ if $M_{ij} < \beta\varepsilon/c$ and $M'_{ij} \doteq \min\{\beta c/\varepsilon, M_{ij}\}$ otherwise. As shown in [11], any $(1-6\varepsilon)$-approximate primal-dual pair for the transformed problem will be a $(1-7\varepsilon)$-approximate primal-dual pair for the original problem.

In the pre-processing step, instead of sorting the row and column lists, *pseudo-sort* them — sort them based on keys $\lfloor \log_2 M_{ij} \rfloor$. These keys will be integers in the range $\log_2(\beta) \pm \log(c/\varepsilon)$. Use bucket sort, so that a row or column with k entries can be processed in $O(k + \log(c/\varepsilon))$ time. The total time for pseudo-sorting the rows and columns is $O(n + (r+c)\log(c/\varepsilon))$.

Then, in the tth iteration, maintain the data structures as before, except as follows.

Compute the set $\mathcal{I}_t$ as follows. Traverse the pseudo-sorted jth column until an index i with $M_{ij'}\delta_{i'j'} < z/2$ is found. (No indices later in the list can be in $\mathcal{I}_t$.) Take all the indices i seen with $M_{ij'}\delta_{i'j'} \geq z$. Compute the set $\mathcal{J}_t$ similarly. Total time for this is $O(\sum_t 1 + |\mathcal{I}'_t| + |\mathcal{J}'_t|)$, where $\mathcal{I}'_t$ and $\mathcal{J}'_t$ denote the sets of indices actually traversed (so $\mathcal{I}_t \subseteq \mathcal{I}'_t$ and $\mathcal{J}_t \subseteq \mathcal{J}'_t$).

When an index j leaves the set J, delete all entries in the jth column list from all row lists. For each row list affected, set $\hat{u}_i$ to *two* times the first element remaining in the row list. This ensures $\hat{u}_i \in [1,2]\max_{j\in J} M_{ij}$.

These are the only details that are changed.

The total time is now $O(n + (r+c)\log(c/\varepsilon))$ plus $O(\sum_t 1 + |\mathcal{I}'_t| + |\mathcal{J}'_t|)$. To finish we bound the latter term.

Lemma 8 *With probability at least* $1 - 2/rc$,

$$\textstyle\sum_t (1 + |\mathcal{I}'_t| + |\mathcal{J}'_t|) \;=\; O((r+c)N).$$

proof: Consider a given iteration. Fix i' and j' chosen in the iteration. For each i, note that, for the random $z \in [0,1]$,

$$\begin{aligned}
\Pr[i \in \mathcal{I}'_t] &\leq \Pr[z/2 \leq M_{ij'}\delta_{i'j'}] \\
&\leq 2M_{ij'}\delta_{i'j'} \\
&= 2\Pr[z \leq M_{ij'}\delta_{i'j'}] \\
&= 2\Pr[i \in \mathcal{I}_t].
\end{aligned}$$

Fix an i. Applying Chernoff for random stopping times (Lemma 10), for any $\delta, A \geq 0$,

$$\Pr\Big[\; (1-\delta)\textstyle\sum_t [i \in \mathcal{I}'_t] \;\geq\; 2\sum_t [i \in \mathcal{I}_t] \;+\; A \;\Big]$$

is at most $\exp(-\delta A)$. (Above $[i \in S]$ denotes 1 if $i \in S$ and 0 otherwise.)

Taking $\delta = 1/2$ and $A = 4\ln(rc)$, with probability at least $1 - (rc)^2$, $\sum_t [i \in \mathcal{I}'_t] \;\leq\; 4\sum_t [i \in \mathcal{I}_t] \;+\; 8\ln(rc)$.

Likewise, for any j, with probability at least $1 - 1/(rc)^2$, $\sum_t [j \in \mathcal{J}'_t] \;\leq\; 2\sum_t [j \in \mathcal{J}_t] \;+\; 8\ln(rc)$.

Taking the naive union bound over all i and j, with probability at least $1 - 1/rc$, $\sum_t (|\mathcal{I}'_t| + |\mathcal{J}'_t|)$ is at most $4\sum_t (|\mathcal{I}_t| + |\mathcal{J}_t|) \;+\; 8(r+c)\ln(rc)$.

By Lemma 5 this is $O((r+c)N)$.

We know (Lemma 7) that the number of empty iterations is $O((r+c)N)$ with probability at least $1 - 1/rc$. The lemma follows by applying the naive union bound. □

If the event in the lemma happens, then the total time is $O(n + (r+c)\log(n)/\varepsilon^2)$. This proves Theorem 3. □

3. Empirical results

We tested our algorithm experimentally and compared its running time to that of the GLPK (Gnu Linear Programming Kit) Simplex algorithm (glpsol version 4.15 with default options). Our first conclusion is that the running time of our implementation is well-predicted by the analysis, with a leading constant factor of about 12 basic operations in the big-O term in which ε occurs. Our second conclusion is that for large inputs the algorithm can be substantially faster than Simplex. For inputs with 2500-5000 rows and columns, the algorithm (with $\varepsilon = 0.01$) is faster than Simplex by factors ranging from tens to hundreds. The speedup grows roughly linearly in rc.

We used inputs with $r, c \in [739, 5000]$, $\varepsilon \in \{0.02, 0.01, 0.005\}$, and matrix density $d \in \{1/2^k : k = 1,2,3,4,5,6\}$. For each (r,c,d) tuple that we used, we generated a random 0/1 matrix with r rows and c columns, where each entry was 1 with probability d. We ran our solver for each ε and compared its running time to that taken by a Simplex solver to find a $(1-\varepsilon)$-approximate solution. GLPK failed to finish due to cycling on about 10% of the runs we initially tried; we excluded those inputs from our experiments. This left 167 runs. The complete data for the non-excluded runs is given at the end of the paper.

The running time of our algorithm includes (A) time for pre-processing and initialization, (B) time for sampling (line 5, once per iteration of the outer loop), and (C) time for increments (lines 8 and 9, once per iteration of the inner loops). Theoretically the dominant terms are $O(n)$ for (A) and $O((r+c)\log(n)/\varepsilon^2)$ for (C). For the inputs tested here, the significant terms in practice are for (B) and (C), with the role of (B) diminishing for larger problems. The time (number of basic operations) is well-predicted by the expression

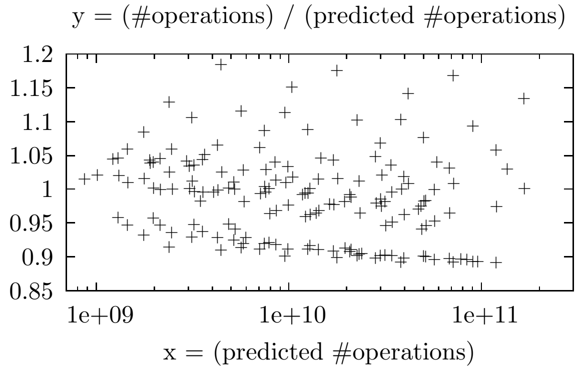
$$[12(r+c) \;+\; 480d^{-1}]\;\ln(rc)/\varepsilon^2 \qquad (1)$$

where $d = 1/2^k$ is the density (fraction of matrix entries that are non-zero, at least $1/\min(r,c)$).

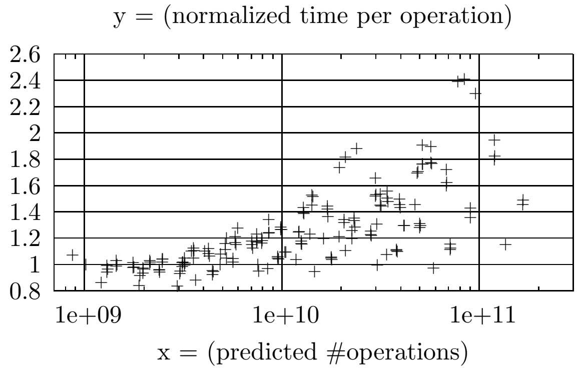
The $12(r+c)\ln(rc)/\varepsilon^2$ term is the time spent in (C), the inner loops; it is the most significant term in our experiments as r and c grow.

The less significant term $480d^{-1}\ln(rc)/\varepsilon^2$ is for (B), and is proportional to the number of samples (that is, iterations of the outer loop). Note that this term *decreases* as matrix density increases. (In our implementation we focused on reducing the time for (C), not for (B). It is probable that the constant 480 above can be reduced with a more careful implementation.)

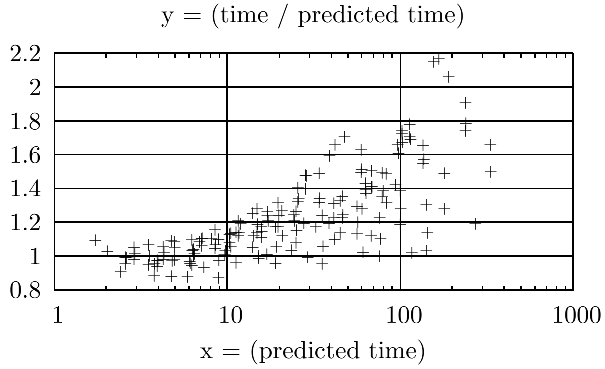
The plot below plots the actual running time divided by the estimate (1), as a function of the estimate, for each of the inputs. The x-axis is on a $\log_{10}$ scale. The plot shows that the actual number of basic operations is close to the estimate for all the inputs:



During our runs the average time per operation was not constant, it grew with larger instances by as much as a factor of two. We don't know why. We observed this effect across a number of machines. We suspect caching or memory allocation issues. The plot below shows the growth in average time per operation.

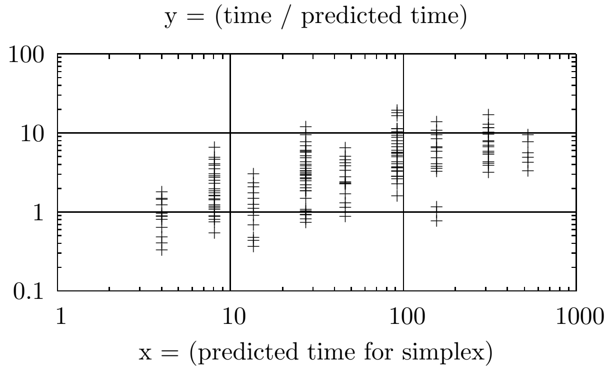


The next plot shows the run time in seconds, divided by the predicted time (estimated time per operation times the predicted number of operations (1)):



Next we compared the speed of our algorithm to that of the GLPK Simplex algorithm. We estimate the time for Simplex to find a near-optimal approximation to be at least $5\min(r,c)rc$ basic operations. This estimate comes from assuming that at least $\Omega(\min(r,c))$ pivot steps are required (because this many variables will be non-zero in the final solution), and each pivot step will take $\Omega(rc)$ time. (This holds even for sparse matrices due to rapid fill-in, although we note that more sophisticated solvers such as CPLEX may do a better job of maintaining sparsity.) The leading constant 5 comes from our experimental evaluation. We note that this estimate seems conservative, and indeed GLPK Simplex often exceeds it in our trials.

Here's a plot of the time for Simplex to find a $(1-\varepsilon)$-approximate solution, divided by the conservative estimate ($5\min(r,c)rc$ times the estimated time per operation):

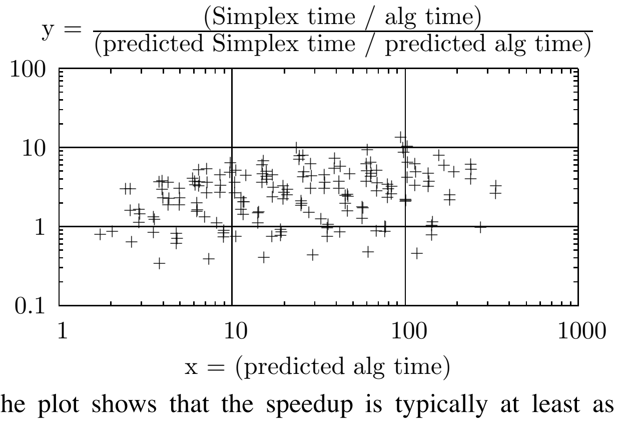


Combining the above estimates, a conservative estimate of the speed-up in using our algorithm (that is, the time for Simplex divided by the time for our algorithm) is

$$\frac{5\min(r,c)rc}{[12(r+c)\ +\ 480d^{-1}]\ \ln(rc)/\varepsilon^2.} \tag{2}$$

This is about $(r/310)^2/\ln(r)$ when $r \approx c$ and $\varepsilon = 0.01$ (for r large).

The plot below plots the actual measured speed-up divided by the conservative estimate (2), as a function of the estimated running time of our algorithm.



The plot shows that the speedup is typically at least as predicted in (2), and often more.

4. Implementation issues

The primary implementation issue is implementing the random sampling efficiently and precisely. The data structures in [12, 8], have two practical drawbacks. The constant factors in the running times are moderately large, and they implicitly or explicitly require that the probabilities being sampled remain in a polynomially bounded range. However, our application uses these data structures in a restricted way, and we were able to use the underlying ideas to build an appropriate data structure with very fast entry-update time and moderately fast sample time. We focused more on reducing the update time than the sampling time, because we expect more update operations than sampling operations. Full details of the practical implementation issues will be provided in a later paper or on request to the authors.

5. Future directions

Can one extend this approach to mixed packing and covering problems, or prove that this is not possible (in a reasonable model)? What about covering with "box" constraints (upper bounds on individual variables)? Can one extend the approach to general packing and covering, e.g. to maximum multicommodity flow (where P is the polytope whose vertices correspond to all $s_i \rightarrow t_i$ paths)? In all of these cases, correctness of a natural algorithm is easy to establish, but the running time is an issue. This seems to be because the coupling approach requires a symmetry (both primal and dual algorithms of a particular kind must exist) that the methods based on "optimal" increments do not require.

Can one adapt this algorithm to efficiently solve dynamic problems, or sequences of closely related problems (e.g. each problem comes from the previous one by a small change in the constraint matrix)? Adapting the algorithm to start with a given primal/dual pair seems straightforward.

Can one use this approach to improve parallel and distributed algorithms for packing and covering (e.g. [11, 15]), perhaps reducing the dependence on ε from $1/\varepsilon^4$ to $1/\varepsilon^3$? In this case, instead of incrementing a randomly chosen variable in each of the primal and dual solutions, one would increment the primal and dual solutions deterministically by a fractional vector: incrementing the primal vector x by $\alpha\hat{p}$ and the dual vector $\hat{x}$ by αp for some α. The correctness proof goes through. Can one bound the number of iterations, assuming the matrix is appropriately preprocessed?

Appendix

Utility lemmas

The first is a one-sided variant of Wald's equation:

Lemma 9 ([14, lemma 4.1]) *Let K be any finite number. Let $x_0, x_1, \ldots, x_T$ be a sequence of random variables, where T is a random stopping time with finite expectation.*

If $\mathrm{E}[x_t - x_{t-1} \,|\, x_{t-1}] \le \mu$ and (in every outcome) $x_t - x_{t-1} \le K$ for $t \le T$, then $\mathrm{E}[x_T - x_0] \le \mu\,\mathrm{E}[T]$.

The second is a Chernoff bound tailored for random stopping times.

Lemma 10 *Let $X = \sum_{t=1}^{T} x_t$ and $Y = \sum_{t=1}^{T} y_t$ be sums of non-negative random variables, where T is a random stopping time with finite expectation, and, for all t, $|x_t - y_t| \le 1$ and*

$$\mathrm{E}\big[\, x_t - y_t \;\big|\; \textstyle\sum_{s<t} x_s, \sum_{s<t} y_s \,\big] \;\le\; 0.$$

Let $\varepsilon \in [0,1]$ and $A \in \mathbb{R}$. Then

$$\Pr\big[\,(1-\varepsilon)X \;\ge\; Y + A\,\big] \;\le\; \exp(-\varepsilon A).$$

proof: Fix $\lambda > 0$. Consider the sequence $\pi_0, \pi_1, \ldots, \pi_T$ where $\pi_t = 0$ for $t > \lambda\mathrm{E}[T]$ and otherwise

$$\begin{aligned} \pi_t &\doteq \prod_{s \le t} (1+\varepsilon)^{x_s} (1-\varepsilon)^{y_s} \\ &= \pi_{t-1} (1+\varepsilon)^{x_t} (1-\varepsilon)^{y_t} \\ &\le \pi_{t-1} (1 + \varepsilon x_t - \varepsilon y_t) \end{aligned}$$

(using $(1+\varepsilon)^x(1-\varepsilon)^y \le (1+\varepsilon x - \varepsilon y)$ when $|x-y| \le 1$).

As $\mathrm{E}[x_t - y_t \,|\, \pi_{t-1}] \le 0$, we have $\mathrm{E}[\pi_t \,|\, \pi_{t-1}] \le \pi_{t-1}$.

Note that, from the use of λ, $\sum_{s\le t} x_s - y_s$ and (therefore) $\pi_t - \pi_{t-1}$ are bounded. Thus Wald's (Lemma 9), implies $\mathrm{E}[\pi_T] \le \pi_0 = 1$.

Applying the Markov bound,

$$\Pr[\pi_T \ge \exp(\varepsilon A)] \;\le\; \exp(-\varepsilon A).$$

So assume $\pi_T < \exp(\varepsilon A)$. Taking logs, if $T \le \lambda\mathrm{E}[T]$,

$$X \ln(1+\varepsilon) - Y \ln(1/(1-\varepsilon)) \;=\; \ln \pi_T \;<\; \varepsilon A.$$

Dividing by $\ln(1/(1-\varepsilon))$ and applying the inequalities $\ln(1+\varepsilon)/\ln(1/(1-\varepsilon)) \ge 1-\varepsilon$ and $\varepsilon/\ln(1/(1-\varepsilon)) \le 1$, we get $(1-\varepsilon)X < Y + A$. Thus,

$$\begin{aligned} &\Pr[(1-\varepsilon)X \ge Y + A] \\ &\quad\le\; \Pr[T \ge \lambda\mathrm{E}[T]] + \Pr[\pi_T \ge \exp(\varepsilon A)] \\ &\quad\le\; 1/\lambda + \exp(-\varepsilon A). \end{aligned}$$

Since λ can be arbitrarily large, the lemma follows. □

References

[1] *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, California, 9–11 Jan. 2000.

[2] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta algorithm and applications. http://www.cs.princeton.edu/∼arora/pubs/MWsurvey.pdf, 2005.

[3] D. Bienstock. *Potential Function Methods for Approximately Solving Linear Programming Problems: Theory and Practice*. Kluwer Academic Publishers, Boston, MA, 2002.

[4] D. Bienstock and G. Iyengar. Solving fractional packing problems in $O(1/\varepsilon)$ iterations. In *Proceedings of the Thirty First Annual ACM Symposium on Theory of Computing*, pages 146–155, Chicago, Illinois, 13–16 June 2004.

[5] F. Chudak and V. Eleuterio. Improved approximation schemes for linear programming relaxations of combinatorial optimization problems. In *Proceedings of the eleventh IPCO Conference, Berlin, Germany*. Springer, 2005.

[6] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Thirty Ninth Annual Symposium on Foundations of Computer Science*, Miami Beach, Florida, 20–22 Oct. 1998. IEEE.

[7] M. D. Grigoriadis and L. G. Khachiyan. A sublinear-time randomized approximation algorithm for matrix games. *Operations Research Letters*, 18(2):53–58, 1995.

[8] T. Hagerup, K. Mehlhorn, and J. I. Munro. Optimal algorithms for generating discrete random variables with changing distributions. *Lecture Notes in Computer Science*, 700:253–264, 1993. Proceedings 20th International Conference on Automata, Languages and Programming.

[9] P. Klein and N. E. Young. On the number of iterations for Dantzig-Wolfe optimization and packing-covering approximation algorithms. *Lecture Notes in Computer Science*, 1610:320–327, 1999.

[10] J. Könemann. Fast combinatorial algorithms for packing and covering problems. Master's thesis, Universität des Saarlandes, 1998.

[11] M. Luby and N. Nisan. A parallel approximation algorithm for positive linear programming. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, pages 448–457, San Diego, California, 16–18 May 1993.

[12] Y. Matias, J. S. Vitter, and W. Ni. Dynamic Generation of Discrete Random Variates. *Theory of Computing Systems*, 36(4):329–358, 2003.

[13] M. J. Todd. The many facets of linear programming. *Mathematical Programming*, 91(3):417–436, 2002.

[14] N. E. Young. K-medians, facility location, and the Chernoff-Wald bound. In ACM/SIAM [1], pages 86–95.

[15] N. E. Young. Sequential and parallel algorithms for mixed packing and covering. In *Forty Second Annual Symposium on Foundations of Computer Science*, pages 538–546, Las Vegas, NV, 14–17 Oct. 2002. IEEE.

Data

The following tables tabulate the details of the experimental results described earlier: "t-alg" is the time for our algorithm in seconds; "t-sim" is the time for Simplex to find a $(1-\varepsilon)$-optimal soln; "t-sim%" is that time divided by the time for Simplex to complete; "alg/sim" is t-alg/t-sim.

r	c	k	100ε	t-alg	t-sim	t-sim%	alg/sim
739	739	2	2.0	1	3	0.31	0.519
739	739	2	1.0	7	6	0.51	1.251
739	739	2	0.5	33	7	0.64	4.387
739	739	5	2.0	3	1	0.51	2.656
739	739	5	1.0	15	1	0.63	8.840
739	739	5	0.5	63	2	0.76	30.733
739	739	4	2.0	2	2	0.51	0.970
739	739	4	1.0	11	3	0.64	3.317
739	739	4	0.5	46	4	0.76	11.634
739	739	3	2.0	2	3	0.43	0.561
739	739	3	1.0	9	5	0.60	1.745
739	739	3	0.5	38	6	0.72	6.197
1480	740	3	2.0	2	9	0.37	0.304
1480	740	3	1.0	13	13	0.53	0.959
1480	740	3	0.5	57	16	0.64	3.478
1480	740	2	2.0	2	24	0.44	0.102
1480	740	2	1.0	11	33	0.60	0.342
1480	740	2	0.5	51	39	0.71	1.313
1480	740	5	2.0	4	4	0.41	0.928
1480	740	5	1.0	18	6	0.56	2.930
1480	740	5	0.5	77	7	0.66	10.447
1480	740	4	2.0	3	6	0.34	0.495
1480	740	4	1.0	15	10	0.49	1.496
1480	740	4	0.5	64	12	0.60	5.239
740	1480	3	2.0	3	14	0.35	0.211
740	1480	3	1.0	14	21	0.51	0.667
740	1480	3	0.5	63	29	0.71	2.139
740	1480	2	2.0	2	13	0.27	0.192
740	1480	2	1.0	11	25	0.51	0.462
740	1480	2	0.5	54	34	0.68	1.597
740	1480	5	2.0	5	7	0.59	0.699
740	1480	5	1.0	22	9	0.72	2.460
740	1480	5	0.5	94	10	0.82	9.054
740	1480	1	2.0	2	23	0.24	0.097
740	1480	1	1.0	9	41	0.44	0.237
740	1480	1	0.5	47	55	0.59	0.848
740	1480	4	2.0	3	12	0.47	0.313
740	1480	4	1.0	17	15	0.61	1.130
740	1480	4	0.5	73	19	0.75	3.803
1110	1110	3	2.0	3	21	0.30	0.142
1110	1110	3	1.0	13	33	0.48	0.399
1110	1110	3	0.5	58	43	0.62	1.354
1110	1110	6	2.0	6	5	0.64	1.327
1110	1110	6	1.0	29	6	0.76	4.763
1110	1110	6	0.5	121	6	0.83	17.903
1110	1110	5	2.0	4	9	0.48	0.480
1110	1110	5	1.0	20	13	0.64	1.575
1110	1110	5	0.5	86	15	0.77	5.439
1110	1110	4	2.0	3	17	0.43	0.203
1110	1110	4	1.0	16	24	0.60	0.649

r	c	k	100ε	t-alg	t-sim	t-sim2	alg/sim
1110	1110	4	0.5	68	29	0.71	2.325
1111	2222	1	2.0	3	94	0.15	0.036
1111	2222	1	1.0	15	198	0.30	0.077
1111	2222	1	0.5	78	344	0.53	0.227
1111	2222	4	2.0	5	94	0.49	0.057
1111	2222	4	1.0	26	123	0.64	0.212
1111	2222	4	0.5	119	148	0.77	0.803
1111	2222	3	2.0	4	109	0.35	0.042
1111	2222	3	1.0	21	163	0.52	0.134
1111	2222	3	0.5	104	222	0.71	0.467
1111	2222	6	2.0	9	23	0.66	0.426
1111	2222	6	1.0	44	26	0.76	1.664
1111	2222	6	0.5	187	29	0.84	6.346
1111	2222	2	2.0	3	83	0.18	0.047
1111	2222	2	1.0	18	169	0.36	0.111
1111	2222	2	0.5	91	269	0.57	0.339
1111	2222	5	2.0	6	63	0.57	0.110
1111	2222	5	1.0	32	77	0.69	0.415
1111	2222	5	0.5	140	88	0.79	1.594
2222	1111	4	2.0	4	53	0.38	0.092
2222	1111	4	1.0	23	75	0.54	0.311
2222	1111	4	0.5	107	91	0.65	1.185
2222	1111	3	2.0	4	53	0.29	0.080
2222	1111	3	1.0	21	84	0.46	0.253
2222	1111	3	0.5	97	115	0.63	0.848
2222	1111	6	2.0	7	21	0.49	0.373
2222	1111	6	1.0	34	26	0.61	1.297
2222	1111	6	0.5	148	30	0.71	4.816
2222	1111	2	2.0	3	102	0.36	0.037
2222	1111	2	1.0	17	139	0.49	0.127
2222	1111	2	0.5	88	173	0.61	0.513
2222	1111	5	2.0	5	42	0.41	0.141
2222	1111	5	1.0	27	57	0.56	0.472
2222	1111	5	0.5	120	70	0.68	1.696
1666	1666	4	2.0	5	117	0.40	0.045
1666	1666	4	1.0	24	163	0.56	0.153
1666	1666	4	0.5	111	201	0.69	0.554
1666	1666	3	2.0	4	112	0.29	0.040
1666	1666	3	1.0	21	185	0.48	0.114
1666	1666	3	0.5	98	245	0.64	0.400
1666	1666	6	2.0	8	42	0.51	0.210
1666	1666	6	1.0	38	55	0.66	0.697
1666	1666	6	0.5	165	63	0.76	2.612
1666	1666	2	2.0	3	109	0.20	0.036
1666	1666	2	1.0	18	221	0.41	0.083
1666	1666	2	0.5	88	313	0.58	0.282
1666	1666	5	2.0	6	82	0.44	0.080
1666	1666	5	1.0	29	109	0.58	0.269
1666	1666	5	0.5	130	133	0.71	0.981
1666	3332	2	2.0	5	354	0.12	0.017
1666	3332	2	1.0	30	857	0.29	0.036
1666	3332	2	0.5	162	1594	0.54	0.102
1666	3332	5	2.0	9	509	0.51	0.020
1666	3332	5	1.0	51	654	0.65	0.078
1666	3332	5	0.5	227	762	0.76	0.299
1666	3332	1	2.0	5	350	0.09	0.015
1666	3332	1	1.0	24	1003	0.25	0.025
1666	3332	1	0.5	135	1881	0.46	0.072
1666	3332	4	2.0	7	578	0.38	0.014
1666	3332	4	1.0	42	899	0.58	0.047
1666	3332	4	0.5	204	1087	0.71	0.188
1666	3332	3	2.0	6	533	0.20	0.013
1666	3332	3	1.0	36	1095	0.41	0.033
1666	3332	3	0.5	180	1741	0.65	0.104
1666	3332	6	2.0	13	255	0.56	0.051
1666	3332	6	1.0	60	319	0.70	0.190
1666	3332	6	0.5	271	361	0.79	0.752
3332	1666	5	2.0	9	275	0.38	0.033
3332	1666	5	1.0	45	392	0.54	0.115
3332	1666	5	0.5	213	482	0.66	0.441
3332	1666	4	2.0	7	274	0.30	0.028
3332	1666	4	1.0	40	414	0.45	0.097
3332	1666	4	0.5	195	556	0.60	0.352
3332	1666	3	2.0	6	316	0.24	0.020
3332	1666	3	1.0	34	544	0.41	0.063
3332	1666	3	0.5	178	703	0.53	0.254
3332	1666	6	2.0	11	154	0.39	0.071
3332	1666	6	1.0	52	218	0.56	0.238
3332	1666	6	0.5	233	273	0.70	0.854
2499	2499	2	2.0	5	530	0.13	0.011
2499	2499	2	1.0	29	1556	0.40	0.019
2499	2499	2	0.5	159	2275	0.58	0.070
2499	2499	5	2.0	9	580	0.42	0.016
2499	2499	5	1.0	46	793	0.58	0.059
2499	2499	5	0.5	217	960	0.70	0.227
2499	2499	4	2.0	8	662	0.31	0.012
2499	2499	4	1.0	42	1064	0.50	0.040
2499	2499	4	0.5	195	1369	0.64	0.143
2499	2499	7	2.0	17	125	0.50	0.139
2499	2499	7	1.0	76	162	0.65	0.475
2499	2499	7	0.5	327	190	0.77	1.715
2499	2499	3	2.0	6	618	0.18	0.011
2499	2499	3	1.0	35	1079	0.32	0.032
2499	2499	3	0.5	174	1774	0.53	0.099
2500	5000	6	2.0	19	2525	0.52	0.008
2500	5000	6	1.0	98	3337	0.69	0.029
2500	5000	6	0.5	458	3828	0.79	0.120
2500	5000	7	2.0	26	1042	0.60	0.026
2500	5000	7	1.0	124	1272	0.73	0.098
2500	5000	7	0.5	556	1427	0.82	0.390
5000	2500	3	2.0	10	2165	0.23	0.005
5000	2500	3	1.0	62	3828	0.40	0.016
5000	2500	3	0.5	338	5586	0.58	0.061
5000	2500	6	2.0	17	1352	0.39	0.013
5000	2500	6	1.0	90	1832	0.53	0.049
5000	2500	6	0.5	418	2297	0.66	0.182
5000	2500	5	2.0	14	1752	0.33	0.008
5000	2500	5	1.0	82	2592	0.49	0.032
5000	2500	5	0.5	397	3330	0.63	0.119
5000	2500	4	2.0	12	1916	0.26	0.006
5000	2500	4	1.0	70	3177	0.44	0.022
5000	2500	4	0.5	367	4197	0.58	0.087
3750	3750	7	2.0	23	1828	0.50	0.013
3750	3750	7	1.0	111	2343	0.64	0.047
3750	3750	7	0.5	506	2712	0.74	0.187
3750	3750	6	2.0	18	3061	0.40	0.006
3750	3750	6	1.0	91	4263	0.55	0.022
3750	3750	6	0.5	432	5279	0.68	0.082