Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Black-Box Acceleration of Monotone Convex Program Solvers

Palma London, Shai Vardi, Reza Eghbali, Adam Wierman

To cite this article:
Palma London, Shai Vardi, Reza Eghbali, Adam Wierman (2022) Black-Box Acceleration of Monotone Convex Program Solvers. Operations Research

Published online in Articles in Advance 19 Oct 2022

. https://doi.org/10.1287/opre.2022.2352

https://pubsonline.informs.org/journal/opre

OPERATIONS RESEARCH
Articles in Advance, pp. 1–20
ISSN 0030-364X (print), ISSN 1526-5463 (online)

Methods

Black-Box Acceleration of Monotone Convex Program Solvers

Palma London,[a,*] Shai Vardi,[b] Reza Eghbali,[c] Adam Wierman[a]

[a] California Institute of Technology, Pasadena, California 91125; [b] Purdue University, West Lafayette, Indiana 47907; [c] University of California, Berkeley, California 94720

*Corresponding author

Contact: plondon@caltech.edu, https://orcid.org/0000-0001-6472-8293 (PL); svardi@purdue.edu, https://orcid.org/0000-0003-4720-6826 (SV); eghbali@berkeley.edu, reza.eghbali@ucsf.edu (RE); adamw@caltech.edu (AW)

Received: April 9, 2020
Revised: March 26, 2022
Accepted: May 31, 2022
Published Online in Articles in Advance: October 19, 2022

Area of Review: Optimization

https://doi.org/10.1287/opre.2022.2352

Abstract. This paper presents a black-box framework for accelerating packing optimization solvers. Our method applies to packing linear programming problems and a family of convex programming problems with linear constraints. The framework is designed for high-dimensional problems, for which the number of variables n is much larger than the number of measurements m. Given an $(m \times n)$ problem, we construct a smaller $(m \times \epsilon n)$ problem, whose solution we use to find an approximation to the optimal solution. Our framework can accelerate both exact and approximate solvers. If the solver being accelerated produces an α-approximation, then we produce a $(1-\epsilon)/\alpha^2$-approximation of the optimal solution to the original problem. We present worst-case guarantees on run time and empirically demonstrate speedups of two orders of magnitude.

Funding: Financial support from the National Science Foundation [Grants AitF-1637598, CNS-151894, and CPS-154471] and the Linde Institute is gratefully acknowledged.

Keywords: linear programming • convex optimization • dimension reduction

1. Introduction

This paper proposes a black-box framework that can be used to accelerate both exact and approximate convex programming solvers for packing problems, while maintaining high-quality solutions.

The need to solve extremely large-scale optimization problems is ubiquitous across disciplines. As problems grow in size, a fundamental bottleneck is the availability of fast, efficient, accurate convex solvers. Practical applications require solvers to work at extreme scales, and, despite decades of work, state-of-the-art solvers do not scale as desired in many cases.

In this paper, we propose a framework for accelerating (speeding up) existing convex program (CP) solvers, allowing them to run faster on larger problems. We exploit characteristics of the problem structure to enable these solvers to run in a fraction of the original time. Examples of state-of-the-art solvers include the Splitting Conic Solver (SCS) (O'Donoghue et al. 2016), the Embedded Conic Solver (ECOS) (Domahidi et al. 2013), and commercial linear programing solvers, such as Gurobi and Cplex.

Our focus is on convex problems with *packing constraints*. Specifically, our framework applies to problems of the following form:

$$\text{maximize} \quad \sum_{j=1}^{n} f_j(x_j), \tag{1a}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \le b_i \qquad i \in [m], \tag{1b}$$

$$0 \le x_j \le 1 \qquad j \in [n], \tag{1c}$$

where $a_{ij} \in [0,1]$ is an element of matrix A of size $(m \times n)$, $b \in \mathbb{R}^m_{\ge 0}$, and $f_j : [0,1] \to \mathbb{R}$ are continuous concave nondecreasing functions differentiable over $(0,1)$. We further assume that $f_j(0) = 0$. Note that if this is not the case, the functions can be appropriately shifted. An important special case of the above are packing linear programs (LPs):

$$\text{maximize} \quad \sum_{j=1}^{n} c_j x_j, \tag{2a}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \le b_i \qquad i \in [m], \tag{2b}$$

$$0 \le x_j \le 1 \qquad j \in [n], \tag{2c}$$

where $c \in \mathbb{R}^n_{\ge 0}$. In this paper, we develop a general algorithm for problems of Form (1) and an algorithm specific to Packing Linear Programs (2).

Many problems in machine learning, inference, and resource allocation are packing problems at their

core, and providing fast solvers for these problems is crucial. Examples of such problems include the network utility maximization problem and large-scale wireless networks (Shi et al. 2015), channel transmission (Johansson and Sternad 2005), inference problems in biology (Martino and Martino 2018), scheduling and graph embedding (Plotkin et al. 1995), and associative Markov networks (Taskar et al. 2004). Packing linear programs arise in a wide variety of settings, including the maximum-cut problem (Trevisan 1998), zero-sum matrix games (Nesterov 2005), flow controls (Bartal et al. 2004), auction mechanisms (Zurel and Nisan 2001), wireless sensor networks (Byers and Nasser 2000), and many other areas. In machine learning, they show up in an array of problems—for example, in applications of graphical models (Ravikumar et al. 2010) and in relaxations of maximum a posteriori estimation problems (Sanghavi et al. 2008), among others.

Our focus in this paper is on a class of packing problems for which data are either very costly or hard to obtain. In these situations, $m \ll n$; that is, the number of data points m available is much smaller than the number of variables, n. In a machine-learning setting, this regime is increasingly prevalent because it is often advantageous to consider larger and larger feature spaces, while not necessarily obtaining proportionally more data. Such instances are also common in areas such as genetics, astronomy, and chemistry. There has been considerable research focusing on this class of problems in recent years, in the context of LPs (Donoho and Tanner 2005, Chowdhury et al. 2020) and also more generally in convex optimization and compressed sensing (Donoho 2006, Sun et al. 2019), low-rank matrix recovery (Recht et al. 2010, Chi et al. 2019), and graphical models (Mohan et al. 2014, Kumar et al. 2019).

Because of the increasing size of data sets and the demands this places on convex solvers like Cplex, Gurobi, SCS, and ECOS, there has been considerable effort to develop faster and more efficient algorithms that can scale.

1.1. Contributions

We propose a framework for accelerating exact and approximate convex programming solvers for packing problems of Form (1), of which an important special case is Linear Programing (2). Analytically, we provide worst-case guarantees on both the run time and the quality of the solution produced. Empirically, we show that our framework speeds up Gurobi and SCS by two orders of magnitude, while maintaining a near-optimal solution with error less than 4%.

The approach of this paper is not to design a *new* algorithm, but to design a black-box acceleration framework that can speed up *existing* algorithms. Given a convex program solver $\mathcal{A}$ and a problem of dimension $(m \times n)$, we select a subset of the variables of size $\epsilon_s n$ uniformly at random. We then construct a *sample problem* of dimension $(m \times \epsilon_s n)$, defined only on those selected variables. Thus, the number of variables in the sample problem is greatly reduced compared with the original formulation. We then solve the dual of the sample problem using solver $\mathcal{A}$, treating it as a black box. Finally, we set the values of the original primal variables x_j for all $j \in [n]$ based on the approximate dual solution.

There are two fundamental trade-offs in the framework. The first is captured by the sample size, ϵ_s. Setting ϵ_s to be small yields a dramatic speedup of the algorithm $\mathcal{A}$; however, if ϵ_s is set too small, the quality of the solution suffers. A second trade-off involves feasibility. In order to ensure that the output of the framework is feasible with high probability (and not just that each constraint is satisfied in expectation), the constraints of the sample problem are scaled down by a factor denoted by ϵ_f. Feasibility is guaranteed if ϵ_f is large enough; however, if it is too large, the quality of the solution (as measured by the approximation ratio) suffers.

Our main technical result is a worst-case characterization of the impact of ϵ_s and ϵ_f on the speedup provided by the framework and the quality of the solution. Assuming that algorithm $\mathcal{A}$ gives an α-approximation to the optimal solution of the dual, we prove that the acceleration framework guarantees a $(1-\epsilon_f)/\alpha^2$-approximation to the optimal solution of the original problem, under some assumptions about the input and ϵ_f. We formally state the result in Theorem 1.

The technical requirements for ϵ_f in Theorem 1 impose some restrictions on both the family of problems that can be provably solved using our framework and the algorithms that can be accelerated. In particular, Theorem 1 requires $\min_i b_i$ to be large and the algorithm $\mathcal{A}$ to satisfy approximate complementary slackness conditions (see Section 2). Although the condition on b_i is restrictive, the condition on the algorithms is satisfied by most common solvers—for example, exact solvers and many primal dual approximation algorithms. Further, our experimental results demonstrate that these technical requirements are conservative—the framework produces solutions of comparable quality to the original solver in settings that are far from satisfying the theoretical requirements. In addition, the accelerator works in practice for algorithms that do not satisfy approximate complementary slackness conditions—for example, for gradient algorithms, as in Sridhar et al. (2013). In particular, our experimental results show that the accelerator obtains solutions that are close in quality to those obtained by the algorithms being accelerated on the complete problem *and* that the solutions are obtained considerably faster (by up to two orders of magnitude). The results reported in this paper demonstrate this by accelerating the state-of-the-art commercial solver Gurobi and the SCS solver on a wide array of randomly

generated packing LPs and CPs and obtaining solutions with < 4% relative error and a more than 150 × speedup.

When applied to parallel algorithms, there are added opportunities for the framework to reduce error while increasing the speedup through *speculative execution*: The framework runs multiple *clones* of the algorithm speculatively. The original algorithm is executed on a separate sample, and then the thresholding rule is applied by each clone in parallel, asynchronously. This can improve both the solution quality and the speed. It improves the quality of the solution because the best solution across the multiple samples can be chosen. It improves the speed because it mitigates the impact of *stragglers*, tasks that take much longer than expected due to contention or other issues. In our experiments, incorporating "cloning" into the acceleration framework triples the speedup obtained, while reducing the error by 12%.

1.2. Related Literature

The approach underlying our framework is motivated by recent work that uses ideas from online algorithms to make offline algorithms more scalable—for example, Mansour et al. (2012) and London et al. (2017; 2019)—and builds on our preliminary work (London et al. 2018). A specific inspiration for this work is an online algorithm (Agrawal et al. 2014) that uses a two-step procedure: It solves an LP based on a subsampled constraint matrix, acquired during the first stages of the online algorithm. It then uses the LP solution as the basis of a rounding scheme in later stages. The algorithm only works when the arrival order is random, which is analogous to sampling in the offline setting. However, Agrawal et al. (2014) rely on exactly solving the sample LP acquired in the first stages; considering approximate solutions of the sampled problem, as we do, adds complexity to the algorithm and analysis. Also, unlike their approach, we leverage the offline setting to fine-tune ϵ_f in order to optimize our solution while ensuring feasibility, which is not possible in the online setting. Additionally, we extend these ideas about LPs to convex problems of Form (1).

In general, our work is related to the literature on online algorithms for packing LPs and convex problems under the random permutation model. As discussed above, Agrawal et al. (2014) present an online algorithm for packing LPs. Subsequently, algorithms for online packing LPs and various generalizations to the convex case have been studied by Agrawal and Devanur (2015), Kesselheim et al. (2014), Gupta and Molinaro (2016), and Vera and Banerjee (2021). However, these algorithms either require an update of the dual variable at each online step or solve the primal problem to optimality at each step. Neither of those approaches can be adapted to design an acceleration framework for the convex case in the way that we adapt the work of Agrawal et al. (2014) to design an algorithm for LPs, as discussed above. Hence, the design presented for the convex case deviates significantly from the online algorithms literature.

The sampling phase of our framework is reminiscent of the method of *sketching*; see Woodruff (2014) and references therein. To construct a sketch of a matrix, the matrix is premultiplied by a random matrix, essentially selecting a subset of the measurements, or rows of the matrix. We, however, have a different goal: to select a subset of the variables, or columns of the matrix. Although sketching is designed for solving overdetermined regression problems for which $m \gg n$, we consider the $m \ll n$ setting, where there are relatively few measurements, and we would like to consider a subset of the variables.

Additionally, the second step of our algorithm is a departure from the sketching approach. In sketching, the smaller problem associated with the sketched matrix is solved, producing an approximate primal solution. The guarantees produced are often in the form of bounds on norms of the sketched matrix, which translate directly to results about the optimality of the solution. This is possible because the objective function depends only on norms involving the data to be sketched: A and b. For example, in linear regression, the objective function is $\|Ax-b\|_2^2$. Sketching results typically produce bounds on $\|S(Ax-b)\|_2^2$, where S is the sketching matrix. These bounds translate directly to the quality of the approximation of the solution. However, in more general problems, the objective function does not conveniently depend on a norm of the sketched matrix. For example, in linear programs, the objective function is c^Tx; new analysis is needed to account for how the c vector affects the quality of the solution, independently from the subsampled data matrix. In our approach, we develop a second step, in which we assign the primal variables of the original problem based on the dual solution to the sampled problem. This assignment is dependent on the objective function. No such analogy is present in sketching methods. Thus, despite the similarity to sketching in our first step, sketching results do not apply here.

The sampling phase of the framework is also reminiscent of the *experiment design* problem, in which the goal is to solve the least-squares problem using only a subset of available data, while minimizing the error covariance of the estimated parameters—see, for example, Boyd and Vandenberghe (2004). Recent work by Riquelme et al. (2017) applies these ideas to online algorithms when collecting data for regression modeling. Like sketching, experiment design is applied in the overdetermined setting, whereas we consider the underdetermined scenario. Additionally, instead of sampling constraints, we sample variables.

The second step of our algorithm is a thresholding step, which is related to the rich literature of LP

rounding; see Bertsimas and Vohra (1998) for a survey. Typically, rounding is used to arrive at a solution to an integer LP (ILP); however, we use thresholding to "extend" the solution of a sampled problem to the original problem. The scheme we use is a deterministic threshold based on complementary slackness conditions. It is inspired by Agrawal et al. (2014) in the LP case, but adapted here for more general problems and extended to handle approximate solvers, rather than exact solvers. Within the rounding LP literature, the most related recent work is that of Sridhar et al. (2013), which proposes a scheme for rounding an approximate LP solution. However, Sridhar et al. (2013) use all of the problem data during the approximation step, whereas we show that it is enough to use a (small) sample of the data.

A key feature of our framework is that it can be parallelized easily when used to accelerate a distributed or parallel algorithm. There is a rich literature on distributed and parallel LP solvers—for example, Yarmish and Slyke (2009), Richert and Cortés (2015), and Nedić et al. (2018). More specifically, there is significant interest in distributed strategies for approximately solving covering and packing linear problems, such as the problems we consider here—for example, Luby and Nisan (1993), Young (2001), Bartal et al. (2004), Allen-Zhu and Orecchia (2015), and Kyng et al. (2020).

Our algorithm is related to *column generation*, a heuristic technique developed for LPs with an extremely large number of variables; see Desaulniers et al. (2005) and Nemhauser (2012) for a survey. In column generation, a series of smaller problems, all defined on a subset of the variables, are solved cyclically. More specifically, in column generation, a subset of the variables is used to define a smaller *master* problem. The dual solution of the master problem is used to define a new *subproblem*, the solution to which will help identify a new variable to be added to the master problem. The master problem is then resolved, and this process is repeated. In contrast, in our approach, we solve a single smaller problem based on one subset of the variables. To make a direct connection with our approach, it is as if we solve the master problem only once—a sort of one-shot column generation.

Column generation has made it possible to find nearly optimal solutions to huge LPs, which would otherwise be considered intractable to solve, and has made a significant impact in, for example, airline crew scheduling problems (Barnhart et al. 2002, Gopalakrishnan and Johnson 2005), transportation routing problems, and scheduling problems. In general, column generation is a heuristic approach; see, for example, Pesnea et al. (2012) and Sadykov and Vanderbeck (2013). Theoretical bounds on the solution quality exist only for problem-specific applications regarding ILPs; for example, the classical cutting stock problem has been studied in this context (Gilmore and Gomory 1963). For solving ILPs in general, column generation along with *branch-and-bound* rounding techniques have been developed to produce the so called *branch-and-price* algorithm (Barnhart et al. 2000). In the context of column generation, the results in our paper can be thought of as theoretical groundwork for a one-shot column generation for packing problems, where, in order to ensure feasibility, we require an assumption on the maximum value of the b vector (recall that we scale $a_{ij} \in [0,1]$ without loss of generality).

Another related algorithmic idea that has been applied to LPs is *constraint sampling*, in which a subproblem is formed from a subset of the constraints; for example, Ho-Nguyen and Kilinc-Karzan (2018). We note that subsampling the constraints differs from subsampling variables (as in column generation), in that in constraint sampling a complete $x \in \mathbb{R}^n$ solution is acquired after solving the subproblem, whereas in column generation, additional work is required to recover the primal solution. More recently, neural networks have been used to speed up subroutines of mixed integer programming solvers (Nair et al. 2020), enabling them to solve larger problems quicker. There has been recent work in the area of dimensionality-reduction techniques that focuses on identifying a subset of relatively important variables or features. In the context of regularized loss minimization, Jalali (2020) proposes a dimensionality-reduction technique that identifies a subset of features that are guaranteed to have zero coefficients in the optimal solution. There is also recent work on *coresets*, a data-summarization technique related to importance sampling, where a small weighted subset of input points or variables is identified. Recent work on coresets includes Tukan et al. (2020) and Borsos et al. (2021) and applications of coresets to neural networks (Borsos et al. 2020, Shim et al. 2021).

In the general context of convex solvers, there has been considerable work over the past decade. Convex solvers are typically either interior-point methods or first-order methods,. Interior-point methods are used in most off-the-shelf public software packages, like SDPT3 (Toh et al. 1999) and SeDuMi (Sturm 1999); the commercial software package MOSEK (Andersen and Andersen 2000); and the more recent embedded conic solver ECOS (Domahidi et al. 2013). However, the computational cost of the second-order methods used in interior-point methods is often prohibitive for high-dimensional problems. In comparison, first-order methods tend to scale well for very large problems. For a survey on large-scale optimization methods and first-order methods, see Esser et al. (2010) and Cevher et al. (2014). SCS (O'Donoghue et al. 2016) is a widely used first-order method. It employs an operator-splitting approach, or, specifically, an *alternating-directions method of multipliers* (Gabay and Mercier 1976, Boyd et al. 2011). Given the popularity of SCS, we use it in

this paper to illustrate the acceleration provided by our framework in the convex case.

2. A Black-Box Acceleration Framework

In this section, we introduce our acceleration framework. At a high level, the framework works by using an existing solver in a black-box fashion to solve a small problem, defined on a subset of the variables. This approximate solution is then incorporated in a deterministic thresholding scheme to assign the variables in the original problem.

The framework applies to solvers that are either exact or approximate. In the approximate case, the solution satisfies the approximate complementary slackness if the following holds. Let $x_1, \ldots, x_n$ be a feasible solution to the primal and $y_1, \ldots, y_m$ be a feasible solution to the dual.

- *Primal Approximate Complementary Slackness*: For $\alpha_p \geq 1$ and $j \in [n]$, if $x_j > 0$, then $c_j \leq \sum_{i=1}^{m} a_{ij} y_i \leq \alpha_p \cdot c_j$.
- *Dual Approximate Complementary Slackness*: For $\alpha_d \geq 1$ and $i \in [m]$, if $y_i > 0$, then $b_i/\alpha_d \leq \sum_{j=1}^{n} a_{ij} x_j \leq b_i$.

We call an algorithm $\mathcal{A}$ whose solution is guaranteed to satisfy the above conditions an (α_p, α_d)-*approximation algorithm*. This terminology is nonstandard, but is instructive when describing our results. It stems from a foundational result for LPs that an algorithm $\mathcal{A}$ that satisfies the above conditions is an α-approximation algorithm, where $\alpha = \alpha_p \alpha_d$—for example, Buchbinder and Naor (2009).

The framework we present can be used to accelerate any $(1, \alpha_d)$-approximation algorithm. Although this is a stronger condition than simply requiring that $\mathcal{A}$ is an α_d-approximation algorithm, many common dual ascent algorithms satisfy this condition—for example, Agrawal et al. (1995), Balakrishnan et al. (1989), Bar-Yehuda and Even (1981), Erlenkotter (1978), and Goemans and Williamson (1995). For example, the vertex cover and Steiner tree approximation algorithms of Agrawal et al. (1995) and Bar-Yehuda and Even (1981), respectively, are both (1, 2)-approximation algorithms.

2.1. Setup

We define the dual problem to (1). Let $\phi \in \mathbb{R}^m$ be the dual variables corresponding to the Constraints (1a), and let $\psi \in \mathbb{R}^n$ correspond to Constraints (1b). The dual problem is

$$\underset{\phi \in \mathbb{R}^m_+, \psi \in \mathbb{R}^n_+}{\text{minimize}} \quad b^T \phi - \sum_{j=1}^{n} f_j^\star(a_j^T \phi + \psi_j) + \mathbf{1}^T \psi,$$

where $\mathbf{1}$ is the vector of ones, and $f^\star$ is the concave conjugate function defined as

$$f_j^\star(v) = \inf_{x \in \mathbb{R}_+} v x_j - f_j(x_j). \tag{3}$$

The Lagrangian for Problem (1) is,

$$L(x, \phi, \psi) := \sum_{t=1}^{n} f_j(x_j) - \phi^T (Ax - b) - \psi^T (\mathbf{1} - x), \tag{4}$$

where $x \geq 0$, from which we derive the following optimality condition:

$$x_j \in \arg\max_{x_j \geq 0} \quad f_j(x_j) - (a_j^T \phi + \psi_j) x_j \quad \forall j \in [n]. \tag{5}$$

Note that

$$f_j'(x_j) = a_j^T \phi + \psi_j \quad \forall j \in [n]. \tag{6}$$

The complementary slackness conditions $\psi_j(1 - x_j) = 0$ for all $j \in [n]$ imply that if $\psi_j > 0$, then $x_j^* = 1$, where * denotes optimality. However, if $\psi_j = 0$, then $f_j'(x_j^*) = a_j^T \phi^*$. We use these facts to motivate our algorithm.

2.2. Acceleration Algorithm

Given a $(1, \alpha_d)$-approximation algorithm $\mathcal{A}$, the acceleration framework comprises the following two steps. The approach is summarized in Algorithm 1.

Step 1. Uniformly, at random, select a subset of the variables, $S \subset [n]$, $|S| = s = \lceil \epsilon_s n \rceil$. Relabel the sampled variables by $1, \ldots, s$ for clarity, and define the *sample problem* on these s variables as follows:

$$\underset{x \in \mathbb{R}^s}{\text{maximize}} \quad \sum_{j=1}^{s} f_j(x_j), \tag{7a}$$

$$\text{subject to} \quad \sum_{j=1}^{s} a_{ij} x_j \leq \frac{(1 - \epsilon_f)\epsilon_s}{\alpha_d} b_i \qquad i \in [m], \tag{7b}$$

$$0 \leq x_j \leq 1 \qquad j \in [s]. \tag{7c}$$

Here, α_d is the parameter of the dual approximate complementary slackness guarantee of $\mathcal{A}$, $\epsilon_f > 0$ is a parameter set to ensure feasibility during the thresholding step, and $\epsilon_s > 0$ is the parameter that determines the fraction of the primal variables that are be sampled. Our analytic results give insights for setting ϵ_f and ϵ_s; see Section 4.3.

Next, use solver $\mathcal{A}$ to solve the dual of the Sample Problem (7). Let $\phi^S \in \mathbb{R}^m$ be the dual variables corresponding to the Constraints (7b), and let $\psi^S \in \mathbb{R}^s$ correspond to Constraints (7c). Then, the sample dual problem is:

$$\underset{\phi \in \mathbb{R}^m_+, \psi \in \mathbb{R}^s_+}{\text{minimize}} \quad \frac{(1 - \epsilon_f)\epsilon_s}{\alpha_d} b^T \phi - \sum_{j=1}^{s} f_j^\star(a_j^T \phi + \psi_j) + \mathbf{1}^T \psi, \tag{8}$$

where $\mathbf{1}$ is the vector of ones, and $f^\star$ is the concave conjugate function, as defined in (3). Finally, retrieve the sample dual solution (ϕ^S, ψ^S), which is an approximation to the dual solution of the original Problem (1).

Step 2. The second step in our acceleration framework uses the dual solution from the sample problem to define a deterministic thresholding procedure, which is used to construct the solution of Problem (1).

This procedure is motivated by the Optimality Condition (5).

Evaluating (5) when f' is noninvertible (equivalently, f^* is nondifferentiable; $f'^{-1} = f^{*\prime}$), we find, for $j \in [s]$,

$$x_j^S \in \partial f^*(a_j^T \phi^S + \psi_j^S) = \operatorname*{arg\,max}_{0 \le x_j^S \le 1} (a_j^T \phi^S + \psi_j^S) x_j^S - f_j(x_j^S). \tag{9}$$

The set of subgradients $\partial f^*(a_j^T \phi^S + \psi_j^S)$ is not necessarily a singleton set. We assign our approximation of the primal solution, which we denote as $x_j(\phi^S, \psi^S)$, to be the smallest element in $\partial f^*(a_j^T \phi^S + \psi_j^S)$. Such an element exists because the set is a closed convex subset of $[0,1]$. We define the *allocation rule*, $x_j(\phi^S, \psi^S)$, in which we set the primal variables for all $j \in [n]$ as a function of the dual solution:

$$x_j(\phi^S, \psi^S) := \begin{cases} 1 & \text{if } a_j^T \phi^S < f_j'(1) \\ \arg\min \partial f_j^*(a_j^T \phi^S + \psi^S) & \text{otherwise.} \end{cases} \tag{10}$$

In both the strictly concave and linear cases, ψ^S does not play a role in the allocation, and it is omitted from the notation. If f is strictly concave, (10) reduces to the following:

$$x_j(\phi^S) := \begin{cases} 1 & \text{if } a_j^T \phi^S < f_j'(1) \\ f_j'^{-1}(a_j^T \phi^S) & \text{otherwise.} \end{cases} \tag{11}$$

For an example of (11), consider the following: If $f_j(x_j) = c_j \log(x_j)$, then $x_j(\phi^S) = 1$ if $a_j^T \phi^S < c_j$, and $x_j(\phi^S) = \frac{c_j}{a_j^T \phi^S}$ otherwise. If f is linear, (10) reduces to the following binary thresholding procedure:

$$x_j(\phi^S) = \begin{cases} 1 & \text{if } a_j^T \phi^S < c_j \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

The allocation rule (10) reduces to (12) in the linear case because the derivative of f_j is c_j for all x. In (10), we take the smallest value in the set of subgradients $\partial f^*(a_j^T \phi^S + \psi_j^S)$, which, in this case, is $x = 0$.

Algorithm 1 (Acceleration Framework)
Input: Convex Program $\mathcal{C}$, solver $\mathcal{A}$, $\epsilon_s > 0$, $\epsilon_f > 0$
Output: $\hat{x} \in \mathbb{R}^n$

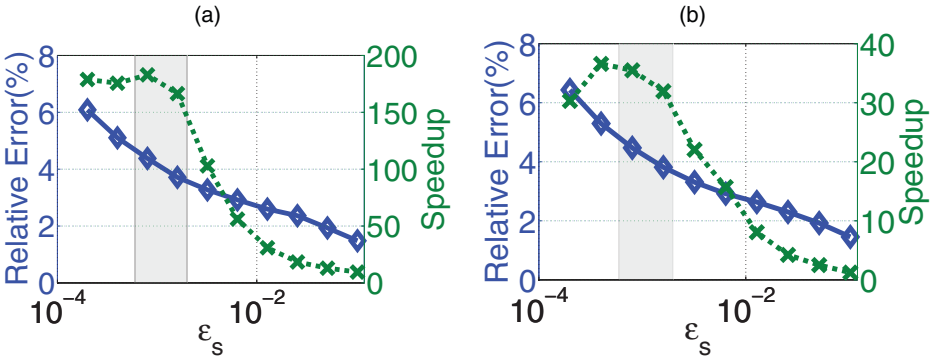
1. Select $s = \lceil \epsilon_s n \rceil$ primal variables uniformly at random. Solve the Sample Dual Problem (8) using $\mathcal{A}$ to find an (approximate) dual solution $y^S = [\phi^S, \psi^S] \in [\mathbb{R}^m, \mathbb{R}^s]$.
2. Set $\hat{x}_j = x_j(\phi^S, \psi^S)$ as defined in (10), for all $j \in [n]$.

2.2.1. Setting ϵ_s and ϵ_f.

The parameters ϵ_s and ϵ_f must be chosen when using the framework. In general, parameter ϵ_s is chosen based on the user's requirement for accuracy versus speed; it directly controls the size of the sample problem. The parameter ϵ_f is related to the feasibility and nearness to optimality of the solution.

Concretely, we suggest choosing ϵ_s and ϵ_f using the following approach. First, choose ϵ_s based on the user's requirement for speed versus accuracy. This trade-off is illustrated in the experiments section in Figures 1 and 4. Given a fixed ϵ_s, a concrete approach for choosing ϵ_f is given in Algorithm 2. Here, we set $\epsilon_f = 0$ and increase ϵ_f iteratively, solving the sample problem with different values of the parameter. Solving multiple problems in order to identify the best ϵ_f is generally not time-prohibitive, because the sample problems are very small compared with the original. One can also simply set ϵ_f in accordance with the theoretical bounds. Our analytic results in the next section provide guarantees on the largest ϵ_f that guarantees feasibility. However, we find, in practice, that we can often use a smaller ϵ_f and get a closer-to-optimal solution without violating feasibility. Thus, it is useful to search for the minimal ϵ_f that provides feasibility for a given problem. A practical discussion of setting both ϵ_s and ϵ_f is found in Section 4.3.

Figure 1. (Color online) Illustration of the Relative Error and Speedup Across Sample Sizes, ϵ_s



Notes. The shaded area depicts a reasonable setting range for ϵ_s; the speedup is significant, while the relative error is low. Two levels of sparsity, p, are shown. (a) $p = 0.8$. (b) $p = 0.4$.

Algorithm 2 (An Approach for Setting ϵ_f)
Input: Convex Program $\mathcal{L}$, solver $\mathcal{A}$, $\epsilon_s > 0$, $\epsilon_f > 0$
Output: $\hat{x} \in \mathbb{R}^n$
Set $\epsilon_f = 0$.
while $\epsilon_f < 1$ **do**
  $\hat{x}$ = Algorithm 1($\mathcal{L}$, $\mathcal{A}$, ϵ_s, ϵ_f).
  if $\hat{x}$ *is a feasible solution to* $\mathcal{L}$ **then**
    Return $\hat{x}$.
  else
    Increase ϵ_f.

2.2.2. Discussion. Before moving to the analysis, we provide some context about why the Allocation Rule (10) works. Recall that (6) states that $f_j'(x_j) = a_j^T\phi + \psi_j$ for all $j \in [n]$. Hence, if $\psi_j > 0$, the complementary slackness conditions $\psi_j(1 - x_j) = 0$ imply that $x_j^* = 1$. However, if $\psi_j = 0$, then $x_j^* = f_j'^{-1}(a_j^T\phi^*)$. Thus, in the allocation rule, we set $x_j(\phi)$ based on these optimality conditions. Notice that we do not have access to the optimal dual solution ϕ^* to Problem (1), but, rather, to an approximation ϕ^S, the solutions to the Sample Dual Problem (8). Thus, we cannot satisfy the complementary slackness conditions exactly. The key argument we need to make is that, despite having only an approximate ϕ^S, the solution $x_j(\phi^S)$ is nearly optimal. To account for the fact that an approximate dual solution is produced in the first step, we preemptively scale the b vector by the factor $\frac{(1-\epsilon_f)\epsilon_s}{\alpha_d}$ in Constraints (1(a)). Given this rescaling, the solution $x_j(\phi^S)$ is feasible and nearly optimal with high probability.

3. Feasibility and Optimality Guarantees

In this section, we present our main technical results, which provide worst-case guarantees on the feasibility and optimality of the solution provided by our acceleration framework.

Let $\mathcal{C}$ be a packing problem with n variables and m constraints, as in (1), and define $B := \min_{i\in[m]}\{b_i\}$. The following theorem bounds the quality of the solution provided by the acceleration framework.

Theorem 1. *Let $\mathcal{C}$ be a packing problem of Form (1) or (2), with n variables and m constraints, and define $B := \min_{i\in[m]}\{b_i\}$. Let $\mathcal{A}$ be a $(1, \alpha_d)$-approximation algorithm for $\mathcal{C}$, with run time $f(n, m)$. For any $\epsilon_s, \epsilon_f > 0$, if,*

$$B \geq \frac{20\left(m \log n + \log\left(\frac{m}{\epsilon_s}\right)\right)}{\epsilon_f^2 \epsilon_s}, \tag{13}$$

then Algorithm 1 runs in time $f(\epsilon_s n, m) + O(n)$, and obtains a feasible $\left(\frac{1-\epsilon_f}{\alpha_d^2}\right)$-approximation to the optimal solution for $\mathcal{C}$ with probability at least $1 - 2\epsilon_s$.

The key trade-off in the acceleration framework is between the size of the sample problem, determined by ϵ_s, and the resulting quality of the solution, determined by the feasibility parameter, ϵ_f. The accelerator provides a large speedup if ϵ_s can be made small without causing ϵ_f to be too large. Theorem 1 quantifies the trade-off between ϵ_f and ϵ_s, along with the relation to B. The bound on B in Theorem 1 defines the class of problems for which the accelerator is guaranteed to perform well: problems for which $m \ll n$ and B is not too small. Nevertheless, our experimental results successfully apply the framework well outside of these parameters; the theoretical analysis provides a very conservative view on the applicability of the framework.

We note that the result does not depend on the form of the concave function f. In addition, we can obtain a slightly tighter bound for the linear case, but we omit it for conciseness. The bounds we obtain for the linear cases are of the same order; $\Omega\left(\frac{m}{\epsilon_f^2\epsilon_s}\log(n)\right)$. Thus, despite applying to general convex programs in this work, we achieve the same order of bound as for linear programs.

As discussed in the introduction, our work is related to the literature on online algorithms for packing linear programs with optimal competitive ratio under a random permutation model. In this context, B is often referred to as the bid-to-budget ratio. In the case of Agrawal et al. (2014), the one-time learning algorithm achieves a competitive ratio of $1 - \epsilon$ if the bid-to-budget ratio is $\Omega\left(\frac{m}{\epsilon^3}\log\left(\frac{n}{\epsilon}\right)\right)$, which is similar to the assumption on B in Theorem 1. However, Molinaro and Ravi (2013) present a different analysis for the algorithm by Agrawal et al. (2014), in which they require a bid-to-budget ratio of $\Omega\left(\frac{m^2}{\epsilon^3}\log\left(\frac{m}{\epsilon}\right)\right)$. Removing the dependency on n, which is the time horizon of the online problem, is of theoretical importance in the online setting. However, in our setting, even though $m \ll n$, typically, m is comfortably larger than $\log(n)$; hence, a result similar to that of Agrawal et al. (2014) is more desirable.

In addition to exact and approximate LP solvers, our framework can be used solve integer linear programs. These are linear programs with binary decision variables:

$$\text{maximize} \quad \sum_{j=1}^{n} c_j x_j, \tag{14a}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad i \in [m], \tag{14b}$$

$$x_j \in \{0, 1\} \qquad j \in [n]. \tag{14c}$$

The Allocation Rule (12) produces binary solutions, and so naturally the solutions satisfy ILP integrality constraints. This gives rise to the following corollary to Theorem 1.

Corollary 1. *Let $\mathcal{C}$ be a packing ILP problem of Form (14), with n variables and m constraints, and define $B := \min_{i\in[m]}\{b_i\}$. Let $\mathcal{A}$ be a $(1, \alpha_d)$-approximation algorithm for $\mathcal{C}$, with*

run time $f(n,m)$. For any $\epsilon_s, \epsilon_f > 0$, if

$$B \geq \frac{20\left(m \log n + \log\left(\frac{m}{\epsilon_s}\right)\right)}{\epsilon_f^2 \epsilon_s},$$

then Algorithm 1 runs in time $f(\epsilon_s n, m) + O(n)$, and obtains a feasible $\frac{(1-\epsilon_f)}{\alpha_d^2}$-approximation to the optimal solution for $\mathcal{C}$ with probability at least $1 - 2\epsilon_s$.

Our framework can be parallelized; Step 2 of Algorithm 1 can be done in parallel. Additionally, if the solver $\mathcal{A}$ that is being accelerated is parallelizable, further parallelization can be achieved, giving the following corollary to Theorem 1. We only give the corollary for convex programs; the corollaries for LPs and ILPs are analogous.

Corollary 2. *Let $\mathcal{C}$ be a packing convex problem of* Form (1)*, with n variables and m constraints, and define $B := \min_{i \in [m]}\{b_i\}$. Let $\mathcal{A}$ be a parallel $(1, \alpha_d)$-approximation algorithm for $\mathcal{C}$, with run time $f(n,m)$. For any $\epsilon_s, \epsilon_f > 0$, if*

$$B \geq \frac{20\left(m \log n + \log\left(\frac{m}{\epsilon_s}\right)\right)}{\epsilon_f^2 \epsilon_s},$$

then executing Algorithm 1 on p processors in parallel obtains a feasible $\frac{(1-\epsilon_f)}{\alpha_d^2}$-approximation to the optimal solution with probability at least $1 - 2\epsilon_s$ and has run time $f_p(\epsilon_s n, m) + O(n/p)$, where $f_p(\epsilon_s n, m)$ denotes $\mathcal{A}$'s run time for the sample program on p processors.

4. Experiments

We illustrate the speedup provided by our acceleration framework by using it to accelerate state-of-the-art commercial solvers. We first run our acceleration framework on synthetic randomly generated problems and then provide a real data example. Additionally, we demonstrate the benefits of cloning.

We accelerate Gurobi and SCS in the case of linear and convex programs, respectively. Because of limited space, we do not present results for more specialized solvers; however, the improvements shown here provide a conservative estimate of the improvements possible with more specialized solvers. Similarly, the speedup provided by an exact solver (such as Gurobi) provides a conservative estimate of the improvements when applied to approximate solvers or when applied to solve ILPs.

Note that our experiments consider situations where the assumptions in Theorem 1 on B, m, and n do not hold. Thus, they highlight that the assumptions of the theorem are conservative, and the accelerator can perform well outside of the settings prescribed by the analysis. This is also true with respect to the assumptions on the algorithm being accelerated. Although our proof requires the algorithm to be a $(1, \alpha_d)$-approximation, the accelerator works well for other types of algorithms, too. For example, we have applied our framework to the gradient algorithm, such as Sridhar et al. (2013), in the linear case, with results that parallel those presented for Gurobi below. All experiments are run on a server with Intel E5-2623V3@3.0-GHz eight cores and 64 GB of RAM.

4.1. Linear Case: Accelerating Gurobi

In this section, we describe experiments done on synthetic randomly generated linear programs. We describe the speedup provided for Gurobi, a state-of-the-art commercial LP solver.

4.1.1. Experimental Setup. To illustrate the performance of our accelerator, we run Algorithm 1 on randomly generated LPs. Unless otherwise specified, the experiments use a matrix $A \in \mathbb{R}^{m \times n}$ of size $m = 10^2$, $n = 10^6$. Each element of A, denoted as a_{ij}, is first generated from $[0,1]$ uniformly at random and then set to zero with probability $1 - p$. Hence, p controls the sparsity of matrix A, and we vary p in the experiments. The vector $c \in \mathbb{R}^n_{\geq 0}$ is drawn independent and identically distributed (i.i.d.) from $[1,100]$ uniformly. Each element of the vector $b \in \mathbb{R}^m_{\geq 0}$ is fixed as $0.1n$. (Note that the results are qualitatively the same for other choices of b.) By default, the parameters of the accelerator are set as $\epsilon_s = 0.01$ and $\epsilon_f = 0$, though these are varied in some experiments. Each point in the presented figures is the average of more than 100 executions under different realizations of A and c.

In order to measure the quality of the solution, we define the *relative error* as $(1 - \hat{p}/p^*)$, where p^* is the optimal objective value and $\hat{p}$ is the objective value produced by our algorithm. To measure the acceleration, we define the *speedup* of the accelerated algorithm as the ratio of the run time of the original solver to the run time of our algorithm.

We implement the accelerator in Matlab and use it to accelerate Gurobi. We intentionally perform the experiments with a small degree of parallelism in order to obtain a conservative estimate of the acceleration provided by our framework. As the degree of parallelism increases, the speedup of the accelerator increases, and the quality of the solution remains unchanged (unless cloning is used, in which case it improves).

4.1.2. Experimental Results. Our experimental results highlight that our acceleration framework provides speedups of two orders of magnitude (over 150×), while maintaining high-quality solutions (relative errors of < 4%).

4.1.2.1. The Trade-Off Between Relative Error and Speed. The fundamental trade-off in the design of the accelerator is between the sample size, ϵ_s, and the

quality of the solution. The speedup of the framework comes from choosing ϵ_s small, but if it is chosen too small, then the quality of the solution suffers. For the algorithm to provide improvements in practice, it is important for there to be a sweet spot where ϵ_s is small and the quality of the solution is still good, as indicated in the shaded region of Figure 1.

4.1.2.2. Scalability.

In addition to speeding up LP solvers, our acceleration framework provides significantly improved scalability. Because the LP solver only needs to be run on a (small) sample LP, rather than the full LP, the accelerator provides an order-of-magnitude increase in the size of problems that can be solved. This is illustrated in Figure 2. The figure shows the run time and relative error of the accelerator. In these experiments, we have fixed $p = 0.8$ and $n/m = 10^3$ as we scale m. We have set $\epsilon_s = 0.01$ throughout. As Figure 2(a) shows, one can choose ϵ_s more aggressively in large problems because leaving ϵ_s fixed leads to improved accuracy for large-scale problems. Doing this would lead to larger speedups; thus, by keeping ϵ_s fixed, we provide a conservative estimate of the improved scalability provided by the accelerator. The results in Figure 2(b) illustrate the improvements in scalability provided by the accelerator. Gurobi's run time grows quickly until, finally, it runs into memory errors and cannot arrive at a solution. In contrast, the run time of the accelerator grows slowly and can (approximately) solve problems of much larger size. To emphasize the improvement in scalability, we run an experiment on a laptop with Intel Core i5 CPU and eight GB of RAM. For a problem with size $m = 10^2, n = 10^7$, Gurobi fails due to memory limits. In contrast, the accelerator produces a solution in 10 minutes with relative error less than 4%.

4.2. Convex Case: Accelerating SCS

In this section, we empirically demonstrate the speedup provided by our acceleration method for convex problems of Form (1). We accelerate SCS, a state-of-the-art convex program solver. As in the LP case, the experiments show that our method provides order-of-magnitude speedups, while producing near-optimal solutions. We consider several monotonically increasing objective functions, as described below.

4.2.1. Experimental Setup.

We implement the accelerator in python and use CVXPY (Diamond and Boyd 2016) to call SCS. We use randomly generated problems for our experiments, and each point in the figures is averaged over 50 executions under different realizations of A, b, and f_j for all $j \in [n]$.

To generate random instances, we generate matrices $A \in \mathbb{R}^{m \times n}$ as follows. Each element of A is drawn i.i.d. from the uniform distribution $U[0,1]$ and then set to zero with probability $1-p$. We vary p in the experiments, which controls the sparsity of matrix. We set $b = Ay + 0.1z$, where y and z are both random vectors drawn i.i.d. from $U[0,1]$. We consider concave objective functions $f_j(x_j) = c_j \log(x_j)$ and $f_j(x_j) = c_j x_j^{1/2}$, where c_j is drawn i.i.d. from $U[0,1]$. We note that the derivative does not exist at $x_j = 0$ for the function $\log(x_j)$. Instead, to satisfy our assumptions, we can solve an equivalent problem with objective function $\log(x_j + \delta) - \log(\delta)$.

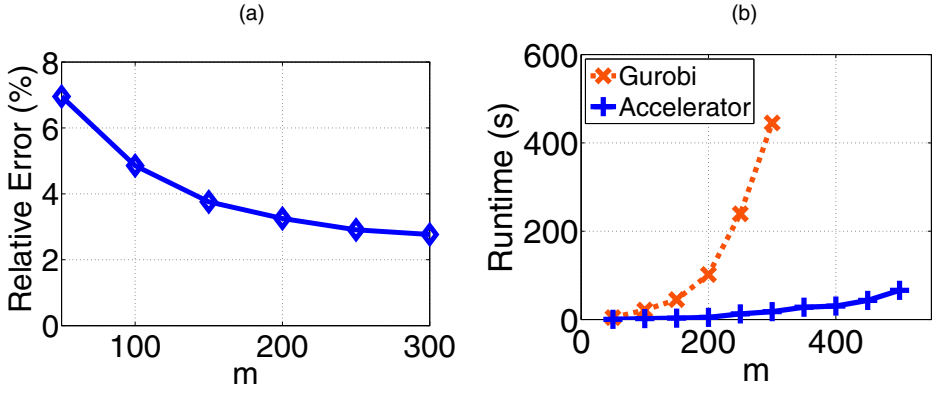
Unless otherwise specified, the problem dimensions are $m = 10^2, n = 10^6$, the parameters are set as $\epsilon_s = 0.01$ and $\epsilon_f = 0$, and ϵ_f is increased, as described in Algorithm 2, until feasibility is reached. The parameters are varied in experiments.

Note that these experiments consider settings in which the assumptions of Theorem 1 on B do not hold. When we set $b = Ay + 0.1z$ as described above, these values are smaller than the bound on B. However, our algorithm performs well, even in this setting, demonstrating that the assumptions of the theorem are conservative.

4.2.2. Experimental Results.

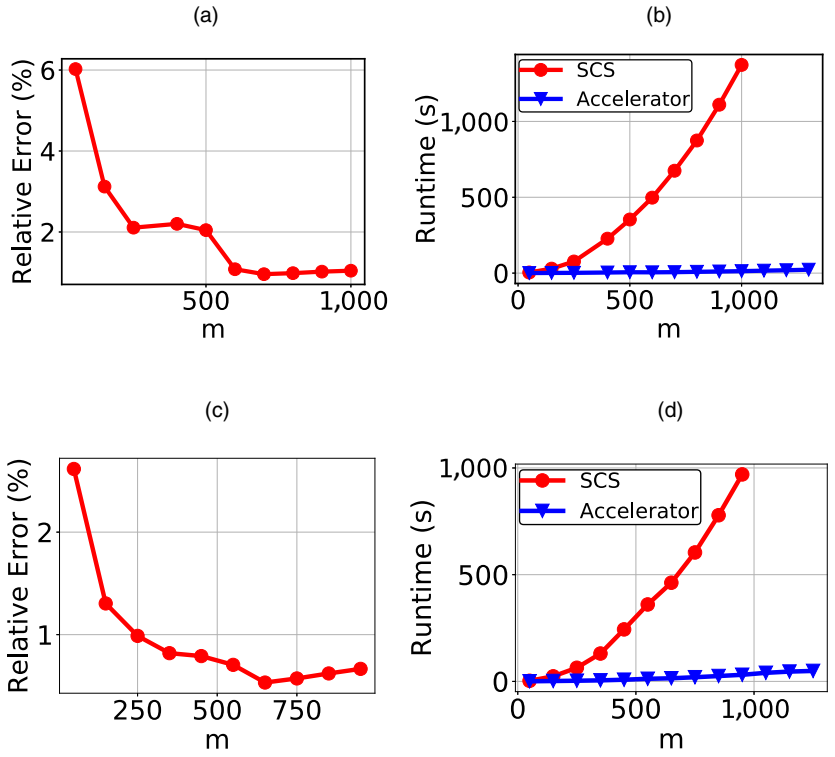
We see a remarkable speedup in our experiments, while maintaining high-quality solutions. Figure 3 demonstrates the relative

Figure 2. (Color online) Illustration of the Relative Error and Run Time as the Problem Size, m, Grows



Notes. Throughout, $n/m = 10^3$. In (b), Gurobi ran into memory errors for problems with $m > 300$. However, our accelerator algorithm was able to solve the problems in a fraction of the time, with low relative error. (a) Relative error. (b) Run time.

Figure 3. (Color online) The Relative Error and Run Time as the Problem Size, m, Grows, Where We Fix the Ratio $n/m = 10^3$



Notes. In (a) and (b), $f_j(x_j) = c_j \log(x_j)$, and in (c) and (d), $f_j(x_j) = c_j x_j^{1/2}$. (a) Relative error. (b) Run time. (c) Relative error. (d) Run time.

error and speedup as the size of the problems are varied. The ratio $n/m = 10^3$ is fixed as we vary both n and m. We set $p = 0.8$ and $\epsilon_s = 0.01$. Here, we see a speedup of multiple orders of magnitude compared with SCS. Additionally, the line corresponding to SCS stops at about $n = 1M$ variables. At this point, SCS runs into memory errors and is unable to proceed. However, our algorithm is able to solve the same problem in a factor-of-100 less time, while achieving a relative error of 1%. Our accelerator is additionally able to proceed with larger problems, ones that SCS cannot handle. Thus, our accelerator provides remarkable scalability: We can quickly solve problems with orders-of-magnitude more variables than SCS can handle.

Figure 4 demonstrates the relative error and speedup as the sample size ϵ_s is varied. We see that, as ϵ_s decreases, the speedup is larger, but problems become harder, and, thus, error increases. For larger ϵ_s, the speedup is less pronounced, but the accuracy is very high.

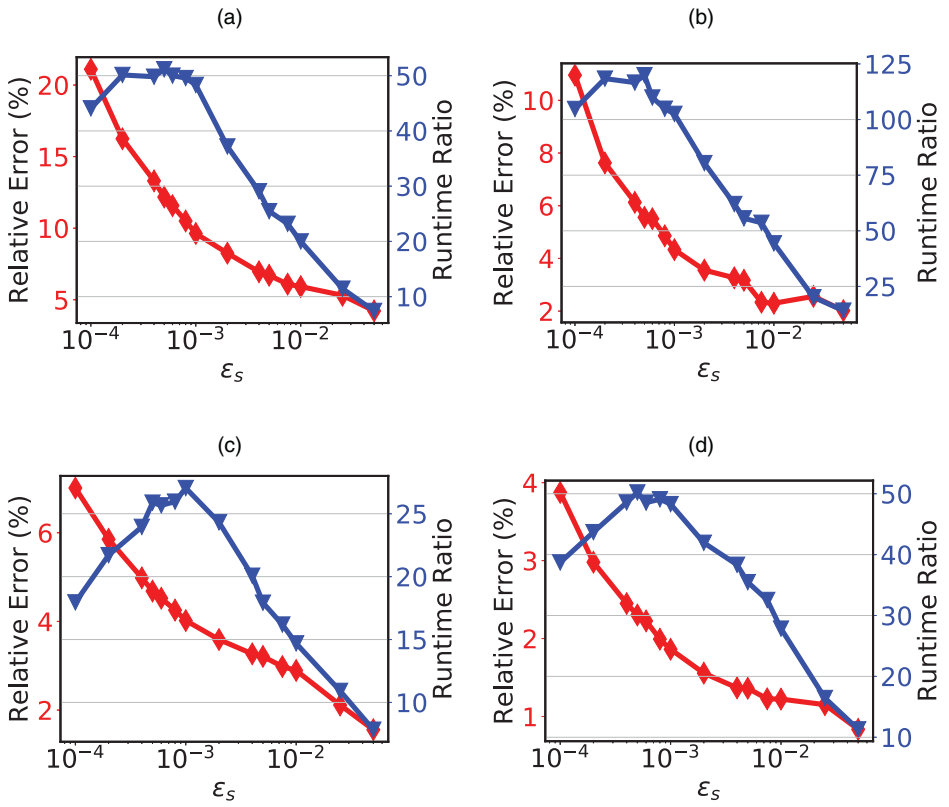
4.3. Setting ϵ_f and ϵ_s in Practice

Although Theorem 1 gives guidance on how to set ϵ_f and ϵ_s to ensure the worst-case guarantees, in practice, it is possible to be more aggressive. Figure 5 demonstrates the nature of the solution as ϵ_s and ϵ_f vary. We plot the optimal solution sorted by magnitude, as well as our approximate solution sorted in the same order as the optimal—that is, x_j^* and $x(\phi)_j$ appear on the same vertical line for the same j. First, we observe the effects of varying ϵ_f. In Figure 5, (a) and (c), we experiment with setting $\epsilon_f = 0$ and find that the solution produced is infeasible. However, in Figure 5, (b) and (d), we increase ϵ_f and find a feasible solution. Visually, when comparing Figure 5, (a) and (b), we can see that our solution shifts downward in Figure 5(b), which makes sense; a solution with smaller values is more likely to be feasible in a packing maximization problem. Secondly, we observe the effects of varying ϵ_s. When comparing Figure 5, (b) and (d), we demonstrate that increasing ϵ_s causes the approximate solution to tighten around the optimal; a larger sample size allows for a closer-to-optimal solution.

4.4. Comparison with the Online Algorithm of Agrawal et al. (2014)

Our framework extends the work of Agrawal et al. (2014) in three main ways: (i) We extend the results to convex programs; (ii) we allow approximate solvers; and (iii) we leverage the offline nature of our problem to introduce another parameter, ϵ_f, and fine-tune it. In this section, we illustrate the advantages of the third point by comparing our algorithm with that of the algorithm of Agrawal et al. (2014) adapted to the offline setting.

Figure 4. (Color online) The Relative Error and Run-Time Ratios as the Sample Size, ϵ_s, Varies, Where We Fix $n = 5 \times 10^5$, $m = 100$



Notes. In (a) and (b), $f_j(x_j) = c_j \log(x_j)$, and in (c) and (d), $f_j(x_j) = c_j x_j^{1/2}$. (a) $p = 0.4$. (b) $p = 0.8$. (c) $p = 0.4$. (d) $p = 0.8$.

The algorithm of Agrawal et al. (2014) can be summarized as follows: Observe s columns of A: $a_1, \ldots, a_s$, and set $x_1 = \cdots = x_s = 0$. Set the remaining variables sequentially, as follows. For $t \in \{s+1, \ldots, n\}$, if there is some $i \in [m]$ such that $a_{it} x_t + \sum_{j=1}^{t-1} a_{ij} x_j > b_i$, set $x_t = 0$; otherwise, set $x_t(\phi^S)$ according to Allocation Rule (12). We adapt this algorithm to the offline setting by allowing it to set $x_1, \ldots, x_s$ according to the above rule as well and denote the modified algorithm by OLS. We use a similar experimental setup to that of Section 4.1, with $m = 10^2, n = 10^5$, and $p = 0.4$, and the element of the vector b is fixed as $0.05n$. The vector c is drawn i.i.d. from the exponential distribution with mean 10. We fine-tune ϵ_f using binary search.

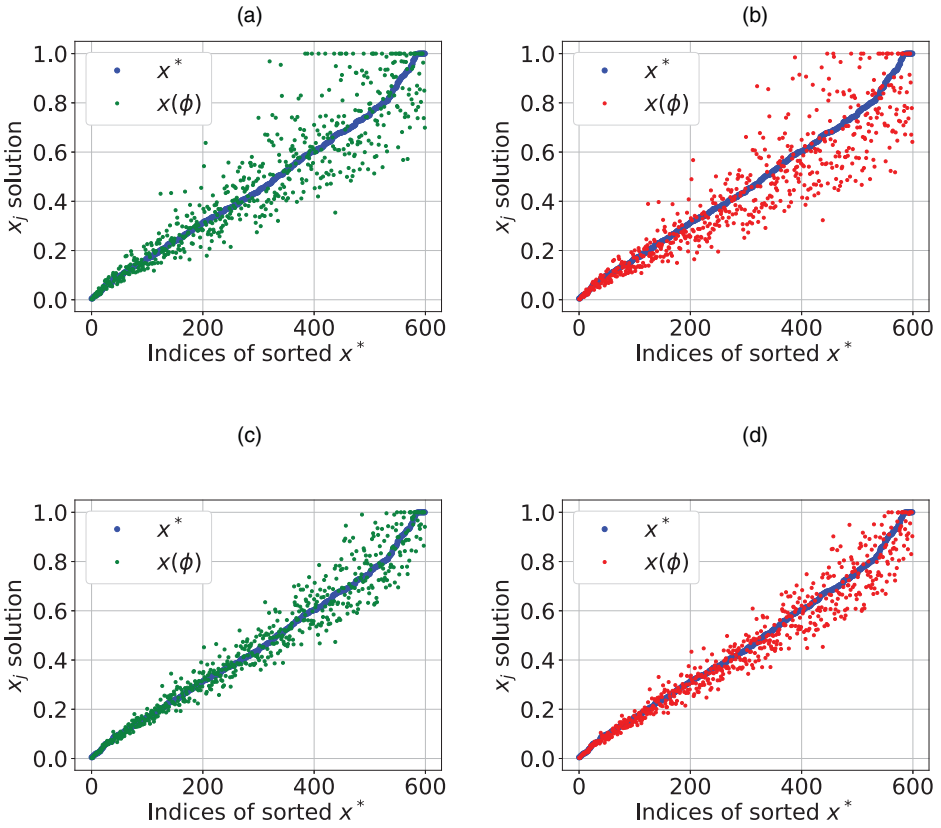
Figure 6(a) shows the ratio of the solution output by our algorithm and that of OLS, where ϵ_s is varied between 0.01 and 0.2. We can see that when $\epsilon_s \approx 0.07$, OLS performs almost as well as ours, but its performance degrades as ϵ_s increases or decreases. This is due to the inherent trade-off in the choice of ϵ_s: When ϵ_s is small, there is an increased probability that the resulting solution will be infeasible if all of the primal variables are assigned according to the Allocation Rule (12). If this is the case, many of the primal variables are set to zero, which adversely affects the solution (especially if their coefficient in the objective function is large). Figure 6(b) validates this observation. We can see that as ϵ_s decreases, a larger percentage of the primal variables are not assigned due to constraint violations. When ϵ_s is large, there is more slack in the constraints of the sample LP, which, again, leads to inefficiencies. In this case, an insufficient number of primal variables are assigned a positive value. The ability to fine-tune the value of ϵ_f allows us to maintain a high solution quality for a large range of ϵ_s.

We note that these results are not a criticism of the algorithm of Agrawal et al. (2014). That algorithm was designed for the online case, and the parameters used in our simulations do not necessarily satisfy their requirements. We include this comparison in order to illustrate the advantages afforded by our framework, which applies to the offline setting, rather than the online setting studied by Agrawal et al. (2014).

4.5. The Benefits of Cloning

Speculative execution is an important tool that parallel analytics frameworks use to combat the impact of stragglers. Our acceleration framework can implement speculative execution seamlessly by running multiple clones (samples) in parallel and choosing the ones that

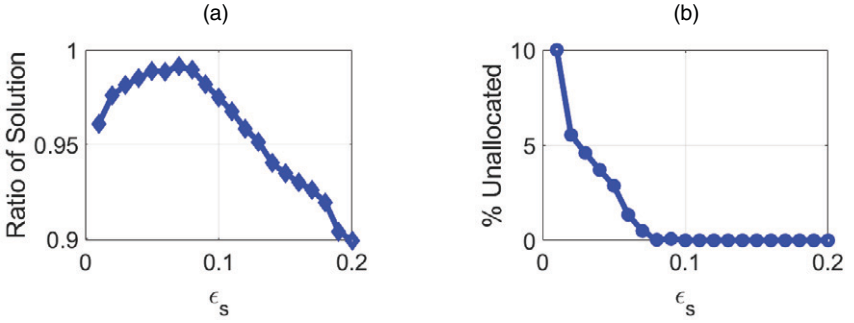
Figure 5. (Color online) Illustration of How the Optimal Solution x^* and our Approximate $x(\hat{\phi})$ Solution Differ



Notes. In particular, increasing ϵ_f affects the feasibility of our solution: Comparing (a) and (b), as ϵ_f is increased, the approximate solution $x(\hat{\phi})$ shifts downward; in (a), the approximate solution is infeasible, but setting $\epsilon_f = 0.1$ produces a feasible $x(\hat{\phi})$. Here, $m = 100$ and $n = 600$ and $f_j(x_j) = c_j \log(x_j)$. (a) $\epsilon_f = 0$, $\epsilon_s = 0.5$. (b) $\epsilon_f = 0.1$, $\epsilon_s = 0.5$. (c) $\epsilon_f = 0$, $\epsilon_s = 0.8$. (d) $\epsilon_f = 0.05$, $\epsilon_s = 0.8$.

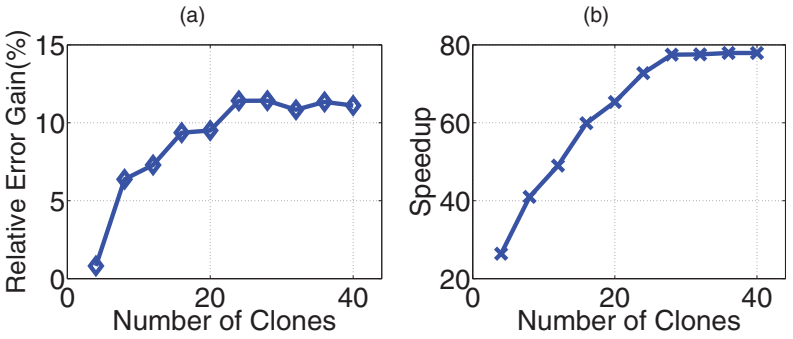
finish the quickest. We illustrate the benefits associated with cloning in Figure 7. This figure shows the percentage gain in relative error and speedup associated with using different numbers of clones. In these experiments, we consider LPs generated as described in Section 4.1.1. We fix $\epsilon = 0.002$ and $p = 0.8$. We vary the number of clones run, and the accelerator outputs a solution after the fastest four clones have finished. Note that the first four clones do not impact the speedup as long as they can be run in parallel. However, for larger numbers of clones, our experiments provide a conservative estimate of the value of cloning because our server only has eight cores. The improvements would be larger than shown in Figure 7 in a system with more parallelism. Despite this conservative comparison, the improvements illustrated in Figure 7 are dramatic. Cloning reduces the relative error of the solution by 12% and triples the speedup. Note that

Figure 6. (Color online) Comparison with the Algorithm of Agrawal et al. (2014) for $\epsilon_s \in [0.01, 0.2]$



Notes. (a) Ratio of the value of our solution and that of the algorithm of Agrawal et al. (2014). (b) Percent of primal variables whose allocation would violate a constraint in OLS.

Figure 7. (Color online) Illustration of the Impact of Cloning on Solution Quality as the Number of Clones Grows



Notes. (a) Relative error. (b) Speedup.

these improvements are significant, even though the solver we are accelerating is not a parallel solver.

4.6. Case Study: California Road Network Data Set

To illustrate the performance in a specific practical setting, we consider an example focused on optimal resource allocation in a network. We consider an LP that represents a multiconstraint knapsack problem associated with placing resources at intersections in a city transportation network. For example, we can place medical testing facilities, advertisements, or emergency supplies at intersections in order to maximize social welfare, but such that there never is a particularly high concentration of resources in any area.

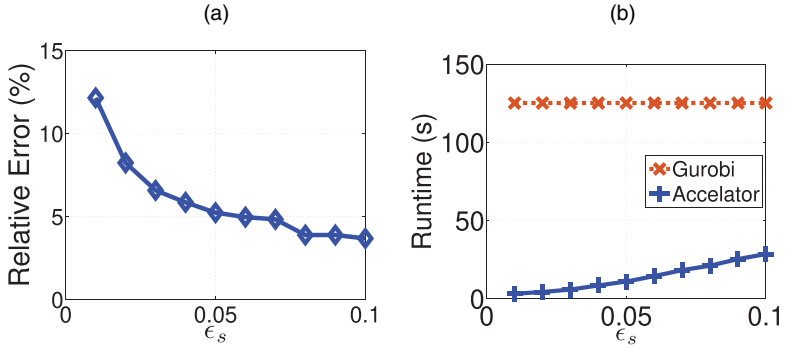
Specifically, we consider a subset of the California road network data set (Leskovec and Krevl 2014), consisting of 100,000 connected traffic intersections. We consider only a subset of a total of 1,965,206 intersections because Gurobi is unable to handle such a large data set when run on a laptop with Intel Core i5 CPU and eight GB of RAM. We choose 1,000 of the 100,000 intersections uniformly at random and defined for each of them a local vicinity of 20,000 neighboring intersections, allowing overlap between the vicinities. The goal is to place resources strategically at intersections, such that the allocation is not too dense within each local vicinity. Each intersection is associated with a binary variable, which represents a yes or no decision to place resources there. Resources are constrained such that the sum of the number of resource units placed in each local vicinity does not exceed 10,000.

Thus, the data set is as follows. Each element A_{ij} in the data matrix is a binary value representing whether the i-th intersection is part of the j-th local vicinity. There are 1,000 local vicinities and 100,000 intersections; hence, A is a (1,000 × 100,000) matrix. Within each local vicinity, there are no more than $b_j = 10,000$ resource units.

The placement of resources at particular locations has an associated utility, which is a quantifier of how beneficial it is to place resources at various locations. For example, the benefit of placing medical test supplies, advertisements, or emergency supplies at certain locations may be proportional to the population of the surrounding area. In this problem, we randomly draw the utilities from Unif[1, 10]. The objective value is the sum of the utilities at whose associated nodes resources are placed.

Figure 8 demonstrates the relative error and run time of the accelerator compared with Gurobi, as we vary the sample size ϵ_s. There is a speedup by a factor

Figure 8. (Color online) Illustration of the Relative Error and Run Time Across Sample Sizes, ϵ_s, for the Real Data Experiment on the California Road Network Data Set



Notes. (a) Relative error. (b) Run time.

of more than 30 when the approximation ratio is 0.9 or a speedup by a factor of about nine when the approximation ratio is 0.95.

5. Proofs

In this section, we present a proof of Theorem 1. The proof approach has two main steps: (1) show that the solution provided by Algorithm 1 is feasible with high probability (Lemma 5); and (2) show that the value of the solution is sufficiently close to optimal with high probability (Lemma 7). First, we present some preliminary results.

5.1. Preliminary Results

We make use of the following concentration bound—for example, as found in van der Vaart and Wellner (1996).

Theorem 2 (Hoeffding-Bernstein Inequality). *Let* $u_1, u_2 \ldots, u_s$ *be random samples without replacement from the real numbers* $r_1, \ldots, r_n$, *where* $r_j \in [0,1]$. *For* $t > 0$, $\Pr\left[\left|\sum_{j=1}^{s} u_j - \frac{s}{n}\sum_{j=1}^{n} r_j\right| \geq t\right] \leq 2\exp\left(\frac{-t^2}{2s\sigma_n^2 + t}\right)$, *where* $\sigma_n^2 = \frac{1}{n}\sum_{j=1}^{n}\left(r_j - \sum_{j=1}^{n} r_j/n\right)^2$.

Throughout this section, we use the following definition.

Definition 1. Denote the solution produced by Algorithm 1 by $x(\phi^S) \in \mathbb{R}^n$, the solution to the Sample Problem (7) by $x^S \in \mathbb{R}^s$, and the dual sample solutions by $\phi^S \in \mathbb{R}^m$ and $\psi^S \in \mathbb{R}^s$. Let the algorithm $\mathcal{A}$ being accelerated be a $(1, \alpha_d)$-approximation algorithm. Define a sample $S \subset [n]$, $|S| = \epsilon_s n$. Let $\mathcal{S}$ be the set of all samples, and $N = [n]$. Define events, associated with each constraint $i \in [m]$,

$$D_i = \left\{ S \in \mathcal{S} : \sum_{j \in S} a_{ij} x_j^S \leq \frac{(1-\epsilon_f)\epsilon_s}{\alpha_d} b_i \right\}$$

$$E_i = \left\{ S \in \mathcal{S} : \sum_{j \in S} a_{ij} x_j(\phi^S) \leq (1-\epsilon_f)\epsilon_s b_i \right\}$$

$$F_i = \left\{ S \in \mathcal{S} : \sum_{j \in N} a_{ij} x_j(\phi^S) \leq b_i \right\}.$$

For brevity, we drop $\mathcal{S}$ from our notation hereafter. The following claim describes the relation between the sample solution x^S to Problem (7) and the solution $x(\phi^S)$ produced by Algorithm 1.

We restate the dual complementary slackness condition as it applies to our setting:

- *Dual Approximate Complementary Slackness*: For $\alpha_d \geq 1$ and $i \in [m]$, if $\phi_i > 0$, then $b_i/\alpha_d \leq \sum_{j=1}^{n} a_{ij} x_j \leq b_i$; for $j \in [n]$, if $\psi_j > 0$, then $1/\alpha_d \leq x_j \leq 1$.

Claim 1. Let x^S be the sample solution to (7) and $x(\phi^S)$ be the approximate solution to (1). If Algorithm $\mathcal{A}$ is a $(1, \alpha_d)$-approximate solver, then $\alpha_d x_j^S \geq x_j(\phi^S)$ for all $j \in [s]$.

Proof. Recall that in the Allocation Rule (10), we set $x_j(\phi^S)$ based on the optimality conditions of the sample problem. Specifically, we consider (6), reproduced below,

$$f_j'(x_j^S) = a_j^T \phi^S + \psi_j^S \quad \forall j \in [s], \tag{15}$$

and the value of ψ_j^S to motivate the two branches of the General Allocation Rule (10) and its special cases, (11) and (12). Consider the following cases.

- $\psi_j^S > 0$: Evaluating (15), we see that $f_j'(x_j^S) > a_j^T \phi^S$. This is the condition in the first branch of each of the allocation rules, in which we set $x_j(\phi^S) = 1$. The approximate complementary slackness conditions imply that $\alpha_d x_j^S \geq 1$. Recalling that $\alpha_d \geq 1$, we see that $\alpha_d x_j^S \geq x(\phi^S)$. Note also that in the case of an exact solver ($\alpha_d = 1$), the exact complementary slackness conditions $\psi_j^S(1 - x_j^S) = 0$ imply that $x_j^S = 1$, and so $x_j^S = x_j(\phi^S) = 1$.
- $\psi_j^S = 0$: Evaluating (15), we see that $f_j'(x_j^S) = a_j^T \phi^S$, which corresponds to the second branch of the Allocation Rules (10), (11), and (12), which differ. Consider the following cases.

Case 1 (Concave f_j' Noninvertible). As described in the General Allocation Rule (10), we set $x_j(\phi^S)$ equal to the smallest value that satisfies $f_j'(x_j^S) = a_j^T \phi^S$. Thus, $x_j^S \geq x_j(\phi^S)$.

Case 2 (Strictly Concave). The sample solution and the result of the Strictly Concave Allocation Rule (11) are equal: $x_j^S = x_j(\phi^S) = f_j'^{-1}(a_j^T \phi^S)$.

Case 3 (Linear). The Linear Allocation Rule (12) sets $x_j(\phi^S) = 0$, and so $x_j^S \geq x_j(\phi^S)$. □

The following lemma relates the sample solution x^S to the solution $x(\phi^S)$ produced by Algorithm 1.

Lemma 1. *Let* D_i *and* E_i *be as in Definition 1. For all* $i \in [m]$, $D_i \subseteq E_i$.

Proof. We show that $S \in D_i \rightarrow S \in E_i$. For any $S \in D_i$, it holds that

$$\sum_{j \in S} a_{ij} x_j(\phi^S) \leq \sum_{j \in S} \alpha_d a_{ij} x_j^S \leq \alpha_d \frac{(1-\epsilon_f)\epsilon_s}{\alpha_d} b_i = (1-\epsilon_f)\epsilon_s b_i ,$$

implying that $S \in E_i$. The first inequality is due to Claim 1, and the second inequality is due to the definition of D_i. □

Finally, the following lemma will be used in the feasibility argument.

Lemma 2. *For all constraints $i \in [m]$,*

$$\Pr[D_i \cap \bar{F}_i] \le \Pr[E_i \cap \bar{F}_i] \le \Pr\left[\left|\sum_{j\in S} a_{ij}x_j(\phi^S) - \epsilon_s \sum_{j\in N} a_{ij}x_j(\phi^S)\right| > \epsilon_f \epsilon_s b_i\right].$$

Proof. The first inequality is immediate from Lemma 1. We evaluate $\Pr[E_i \cap \bar{F}_i]$:

$$\Pr[E_i \cap \bar{F}_i] = \Pr\left[\sum_{j\in S} a_{ij}x_j(\phi^S) \le (1-\epsilon_f)\epsilon_s b_i \cap \sum_{j\in N} a_{ij}x_j(\phi^S) > b_i\right] \le \Pr\left[\epsilon_s \sum_{j\in N} a_{ij}x_j(\phi^S) - \sum_{j\in S} a_{ij}x_j(\phi^S) > \epsilon_f \epsilon_s b_i\right] \le \Pr\left[\left|\sum_{j\in S} a_{ij}x_j(\phi^S) - \epsilon_s \sum_{j\in N} a_{ij}x_j(\phi^S)\right| > \epsilon_f \epsilon_s b_i\right]. \quad \square$$

5.2. Quantizing the Solution Space

We bound the number of possible solutions $x_j(\phi^S)$. Naively, it may seem that there are an infinite number of possible solutions, as $x_j \in [0,1]$. However, we can bound the number of possible solutions by considering classes of dual variables that result in approximately equivalent primal realizations $x_j(\phi^S)$. We discretize the primal solution space, $[0,1]$, with a grid size of q, and define the following:

Definition 2. Consider a discretization of the primal solution space $[0,1]$, with grid size $q \le 1$. Denote the value of f_j' evaluated at discrete values as $T_j = \{f_j'(0^+), f_j'(q), \ldots, f_j'(q\lfloor 1/q \rfloor) \cup \{f_j'(1^-)\}$.

Definition 3. Consider two dual solutions $\phi^{S_1}, \phi^{S_2} \in \mathbb{R}_+^m$ to Problem (7). Discretize the primal solution space as in Definition 2. Let $G_j = \{(a_j, \xi) \mid \xi \in T_j\}$ and $G = \cup_j G_j$. We say ϕ^{S_1} and ϕ^{S_2} are in the same equivalence class and write $\phi^{S_1} \sim \phi^{S_2}$ when:

$$\forall j,\ \forall (a_j, \xi) \in G_j : a_j^T \phi^{S_1} \ge \xi \iff a_j^T \phi^{S_2} \ge \xi. \tag{16}$$

We show that two dual variables in the same equivalence class, when applied to the Allocation Rule (10), result in two primal solutions, which are elementwise within a distance of q from each other.

Lemma 3. *Consider two dual variables in the same equivalence class. The corresponding primal solutions satisfy $|x_j(\phi^{S_1}) - x_j(\phi^{S_2})| \le q$ for all $j \in [n]$.*

Proof. Suppose that the interval $[0,1]$ is discretized with a grid size of q, as in Definition 2. Consider two dual variables in the same equivalence class as specified in Definition 3. Equation (16) and the Allocation Rule (10) together imply that if $\phi^{S_1} \sim \phi^{S_2}$, then one of the following cases occurs:

- When $a_j^T \phi^{S_1} < f_j'(1^-)$ and $a_j^T \phi^{S_2} < f_j'(1^-)$, then $x_j(\phi^{S_1}) = x_j(\phi^{S_2}) = 1$,
- When $a_j^T \phi^{S_1} \ge f_j'(0^+)$ and $a_j^T \phi^{S_2} \ge f_j'(0^+)$, then $x_j(\phi^{S_1}) = x_j(\phi^{S_2}) = 0$,
- When $f_j'(1^-) \le a_j^T \phi^{S_1} < f_j'(0^+)$ and $f_j'(1^-) \le a_j^T \phi^{S_2} < f_j'(0^+)$, then $|x_j(\phi^{S_1}) - x_j(\phi^{S_2})| \le q$.
- Thus, in general, $|x_j(\phi^{S_1}) - x_j(\phi^{S_2})| \le q$ for all j. $\square$

We employ a classical result of combinatorial geometry (Orlik and Terao 1992) to bound the number of possible primal solutions.

Lemma 4. *There are at most $\left(n\left(1+\frac{1}{q}\right)\right)^m$ possible primal solutions $x(\phi^S)$.*

Proof. We employ a classical result of combinatorial geometry (Orlik and Terao 1992). This result says that, given k points in m-dimensional space, the number of possible separations, or regions, created by an m-dimensional plane is k^m.

We characterize each primal solution by a separation of k points in an m-dimensional plane by a hyperplane. In this context, each point corresponds to a value that the primal solution can take on. The number of values that the primal solution $x_j(\phi^S)$ can take on is described by the size of the set T_j, as defined in Definition 2. Thus, $k = \sum_{j=1}^n |T_j|$.

In the linear case, note that $|T_j| = 1,\ \forall j$; thus, $k = n$. However, in general, the size of set T_j is determined by the number of quantized values on the $[0,1]$ interval, which is dependent on the grid size q. Recalling Definition 2, $|T_j| \le \left(1+\frac{1}{q}\right),\ \forall j$. There are n such sets, and so $k \le n\left(1+\frac{1}{q}\right)$.

Applying Orlik and Terao (1992), we find that the number of possible primal solutions is equal to the maximum number of regions created by the hyperplane, which is at most $\left(n\left(1+\frac{1}{q}\right)\right)^m$. $\square$

5.3. Feasibility

Now ,we turn to the feasibility portion of the proof; we show that the solution provided by Algorithm 1 is feasible with high probability.

Lemma 5. *Let $\mathcal{A}$ be a $(1, \alpha_d)$-approximation algorithm for packing problems of the Form (1) or (2), $\alpha_d \ge 1$. For any $\epsilon_s, \epsilon_f > 0$, if the Condition (13) on B holds, then the solution Algorithm 1 produces is feasible with probability at least $1 - 2\epsilon_s$.*

Proof. We bound the probability that for a given sample S, the sample solution x^S is feasible for the Sample Problem (7), whereas there is some constraint i for which the complete solution $x(\phi^S)$ is infeasible in the original Problem (1). Recall the events defined in Definition 1. Our goal is to bound $\Pr[D_i \cap \bar{F}_i]$. First, we relate the sample solution x^S to $x(\phi^S)$. By Lemma 2, $\Pr[D_i \cap \bar{F}_i] \le \Pr[E_i \cap \bar{F}_i]$. Suppose that the

primal solution space $[0,1]$ is discretized with a grid size of q. Let $q = \frac{\epsilon_f \min_i b_i}{4n}$. Consider two equivalent dual variables, ϕ^{S_1} and ϕ^{S_2}, as defined in Definition 3. Applying Lemma 3, the inequality $q \le \frac{\epsilon_f b_i}{4n}$, and recalling that $a_{ij} \in [0,1]$, we find:

$$\left|\sum_{j\in S} a_{ij}x_j(\phi^{S_1}) - \sum_{j\in S} a_{ij}x_j(\phi^{S_2})\right| \le \sum_{j\in S} q \le \epsilon_s n q \le \epsilon_s\epsilon_f b_i/4, \tag{17}$$

$$\left|\sum_{j\in N} a_{ij}x_j(\phi^{S_1}) - \sum_{j\in N} a_{ij}x_j(\phi^{S_2})\right| < \sum_{j\in N} q < nq < \epsilon_f b_i/4. \tag{18}$$

Now, for each equivalence class C of dual variables, and for each constraint i, we bound the probability of $\Pr[E_i \cap \bar{F}_i \cap \{\phi^S \in C\}]$.

In Line (19), we consider any dual variable $\phi^S \in C$ in a fixed equivalence class. In Line (20), we fix a particular dual variable Φ^C within the equivalence class C. Thus, Φ^C is not a random variable, and so $x_j(\Phi^C)$ is deterministic. This allows us to apply the Hoeffding-Bernstein Inequality (Theorem 2) in the next Line (21), where the randomness present is due only to the choice of samples, indexed by j.

$$\begin{aligned}
&\Pr[E_i \cap \bar{F}_i \cap \{\phi^S \in C\}] \\
&\le \Pr\left[\left|\sum_{j\in S} a_{ij}x_j(\phi^S) - \epsilon_s \sum_{j\in N} a_{ij}x_j(\phi^S)\right| > \epsilon_f\epsilon_s b_i \cap \{\phi^S \in C\}\right], && (19)\\
&\le \Pr\left[\left|\sum_{j\in S} a_{ij}x_j(\Phi^C) - \epsilon_s \sum_{j\in N} a_{ij}x_j(\Phi^C)\right| > \frac{\epsilon_f\epsilon_s b_i}{2}\right], && (20)\\
&\le \Pr\left[\left|\sum_{j\in S} a_{ij}x_j(\Phi^C) - \mathbb{E}_S\left[\sum_{j\in N} a_{ij}x_j(\Phi^C)\right]\right| > \frac{\epsilon_f\epsilon_s b_i}{2}\right], && (21)\\
&\le 2\exp\left(-\frac{\epsilon_f^2\epsilon_s^2 b_i^2}{2\epsilon_s b_i + \epsilon_f\epsilon_s b_i/2}\right)\\
&= 2\exp\left(-\frac{\epsilon_f^2\epsilon_s b_i}{8+2\epsilon_f}\right) \le 2\exp\left(-\frac{\epsilon_f^2\epsilon_s b_i}{8}\right), && (22)
\end{aligned}$$

where Line (19) is due to Lemma 2. Inequality (20) holds due to the fact that if

$$\left|\sum_{j\in S} a_{ij}x_j(\phi^S) - \epsilon_s \sum_{j\in N} a_{ij}x_j(\phi^S)\right| > \epsilon_f\epsilon_s b_i \text{ and } \phi^S \in C, \tag{23}$$

then

$$\begin{aligned}
&\left|\sum_{j\in S} a_{ij}x_j(\Phi^C) - \epsilon_s \sum_{j\in N} a_{ij}x_j(\Phi^C)\right|\\
&\ge \left|\sum_{j\in S} a_{ij}x_j(\phi^S) - \epsilon_s \sum_{j\in N} a_{ij}x_j(\Phi^C)\right| - \frac{\epsilon_f\epsilon_s b_i}{4} \text{ by (17)}\\
&\ge \left|\sum_{j\in S} a_{ij}x_j(\phi^S) - \epsilon_s \sum_{j\in N} a_{ij}x_j(\phi^S)\right| - \frac{\epsilon_f\epsilon_s b_i}{2} \text{ by (18)}\\
&> \frac{\epsilon_f\epsilon_s b_i}{2} \quad \text{by (23).}
\end{aligned}$$

In Line (22), we apply the Hoeffding-Bernstein Inequality (Theorem 2). To complete the proof, we take a union bound over all possible primal solutions and the values i of the m constraints. The number of possible primal solutions is described in Lemma 4. Setting $q \ge \frac{\epsilon_f b_i}{4n}$, we find,

$$2\left(n\left(1+\frac{4n}{\epsilon_f b_i}\right)\right)^m \exp\left(-\frac{\epsilon_f^2\epsilon_s b_i}{8}\right), \tag{24}$$

$$\le 2\left(n\left(1+\frac{4\epsilon_f\epsilon_s n}{m\log(n)}\right)\right)^m \exp\left(-\frac{\epsilon_f^2\epsilon_s b_i}{8}\right) \le \frac{2\epsilon_s}{m}. \tag{25}$$

In (24), b_i appears twice. The assumption on B (13) implies that $b_i > \frac{m\log(n)}{\epsilon_f^2\epsilon_s}$. Thus, (25) follows by first applying this weaker assumption on b_i and then the stronger Assumption (13). Finally, we take a union bound over all constraints: $P(\bigcup_{i=1}^m (D_i \cap \overline{F}_i)) \le m\frac{2\epsilon_s}{m} = 2\epsilon_s$. □

5.4. Optimality

The following lemma describes the relation between the approximate solution $x(\phi^S)$ and the approximation of the primal objective function.

Lemma 6. *Given the dual solution (ϕ^S, ψ^S) to the Sample Problem (7), if the solution produced by Algorithm 1, $\hat{x} = x(\phi^S)$, satisfies the dual approximate complementary slackness conditions for $r \le 1$:*

$$\forall i \quad \phi_i^S > 0 \Rightarrow r b_i \le (A\hat{x})_i \le b_i, \tag{26}$$

$$\forall j \quad \psi_j^S > 0 \Rightarrow r \le \hat{x}_j \le 1, \tag{27}$$

then $\hat{x}$ is an r-approximation of the optimal primal solution x^* Original Problem (1).

Proof. Consider any approximate solution $(\hat{x}, \phi^S, \psi^S)$ that satisfies the approximate complementary slackness conditions, as stated in (26) and (27).

Recall that (5) motivates the Allocation (10). Thus, (6) and (10) imply that $f_j'(\hat{x}_j) \ge a_j^T\phi^S$, $\forall j \in [n]$. Taking into account the concavity of the objective function and the assumption that $f_j(0) = 0$ for all j, we derive the following:

$$\forall j \quad f_j(\hat{x}_j) \ge \hat{x}_j f_j'(\hat{x}_j) \ge \hat{x}_j a_j^T\phi^S. \tag{28}$$

Recall that the dual to the Sample Problem (7) is:

$$\underset{\phi\in\mathbb{R}_+^m,\, \psi\in\mathbb{R}_+^s}{\text{minimize}} \quad b^T\phi - \sum_{j=1}^n f_j^\star(a_j^T\phi + \psi_j) + \mathbf{1}^T\psi,$$

where $f_j^\star(v) = \inf_{x\in\mathbb{R}_+} vx - f_j(x_j)$ is the concave conjugate

function. Thus,

$$\sum_{j=1}^{n} f_j^\star(a_j^T\phi^S + \psi_j^S) = \phi^S A\hat{x} + \psi^{S^T}\hat{x} - \sum_{j=1}^{n} f_j(\hat{x}_j). \tag{29}$$

So, the dual objective is,

$$\begin{aligned}\mathcal{D}(\phi^S) &:= \sum_{i=1}^{m} b_i {\phi^S}_i - \sum_{j=1}^{n} f_j^\star(a_j^T\phi^S + \psi_j^S) + \mathbf{1}^T\psi^S \\ &\le \frac{1}{r}\sum_{i=1}^{m}(A\hat{x})_i {\phi^S}_i - \sum_{j=1}^{n} f_j^\star(a_j^T\phi^S + \psi_j^S) + \psi^{S^T}\hat{x} \\ &= \frac{1}{r}\phi^{S^T}A\hat{x} - \phi^{S^T}A\hat{x} - \psi^{S^T}\hat{x} + \sum_{j=1}^{n} f_j(\hat{x}_j) + \psi^{S^T}\hat{x} \\ &= \left(\frac{1}{r} - 1\right)\phi^{S^T}A\hat{x} + \sum_{j=1}^{n} f_j(\hat{x}_j).\end{aligned}$$

The second line above follows from (26) and (27) and the third from (29). Thus, by (28), the primal objective is,

$$\mathcal{P}(\hat{x}) := \sum_{t=1}^{n} f_j(\hat{x}_j) \ge \phi^{S^T}A\hat{x},$$

which implies $\mathcal{D}(\phi^S) \le \frac{1}{r}\mathcal{P}(\hat{x})$, which, in turn, implies $\mathcal{P}(x^*) \le \frac{1}{r}\mathcal{P}(\hat{x})$. □

Next, we make a mild technical assumption.

Assumption 1. *For any dual solution ϕ^S, there are at most m columns a_j of A such that $a_j^T\phi^S = f_j'(x_j)$.*

Assumption 1 does not always hold; however, it can be enforced by perturbing each f_j by a small amount at random—for example, as described by Devanur and Hayes (2009) and Agrawal et al. (2014).

Claim 2. Let x^S and ϕ^S be solutions to the Sample Problem (7). Then, $\{x_j(\phi^S)\}_{j\in[s]}$ and $\{x_j^S\}_{j\in[s]}$ differ on at most m values of j.

Proof. When $f_j(x)$ is strictly concave, or, equivalently, $f_j'(x)$ is invertible, by Allocation Rule (11), the solutions are trivially equivalent $x_j(\phi^S) = x_j^S$ for all $j \in [s]$.

For instances in which $f_j(x)$ has piece-wise linear components, then $f_j'(x)$ may be noninvertible. In such cases, as described in the General Allocation Rule (10), we set $x_j(\phi^S)$ to be the smallest element such that $f_j'(x) \ge a_j^T\phi^S$. Thus, the resulting function, which we denote $f_j'^{-1}$, may have discontinuity points.

When $a_j^T\phi^S$ falls on a discontinuity point of $f_j'^{-1}$, it is possible that $x_j^S \ne x_j(\phi^S)$. Here, $f'^{-1}(x)$ is a nonincreasing function, and, thus, we can apply Froda's Theorem, which describes the set of discontinuities of a monotone real valued function. The set of discontinuities is countable and is thus of Lebesgue measure zero.

By Assumption 1, there are at most m values of j for which $a_j^T\phi^S = f_j'(x_j)$. If f_j' is noninvertible in these instances, then, by the above reasoning, we choose m values from a countable set of discontinuity points. Therefore, there are at most m cases in which $a_j^T\phi^S$ falls on a discontinuity point, which implies that there are at most m values of j for which $x_j^S \ne x_j(\phi^S)$. □

Now, we show that the solution is approximately optimal with high probability.

Lemma 7. *Let $\mathcal{A}$ be a $(1,\alpha_d)$-approximation algorithm for packing problems of the Form (1) or Linear (2), $\alpha_d \ge 1$. For any $\epsilon_s, \epsilon_f > 0$, if the Condition (13) on B holds, then the solution Algorithm 1 produces is a $(1-3\epsilon_f)/\alpha_d^2$-approximation with probability at least $1-2\epsilon_s$.*

Proof. To show that the solution is approximately optimal, we bound the probability that for a given sample, the sample solution x^S causes constraints i in the sample problem to be nearly tight, whereas the complete solution $x(\phi^S)$ does not cause those constraints to be nearly tight in the original problem. Define events,

$$M_i = \left\{S \in \mathcal{S} : \sum_{j\in S} a_{ij}x_j^S \ge \frac{(1-2\epsilon_f)\epsilon_s}{\alpha_d^2} b_i\right\}$$
$$N_i = \left\{S \in \mathcal{S} : \sum_{j\in N} a_{ij}x_j(\phi^S) < \frac{1-3\epsilon_f}{\alpha_d^2} b_i\right\}.$$

We want to bound $\Pr[M_i \cap \bar{N}_i]$. When $\phi_i^S > 0$, the approximate dual complementary slackness condition associated with the i-th primal constraint of Problem (7) is:

$$\sum_{j\in S} a_{ij}x_j^S \ge \frac{(1-\epsilon_f)\epsilon_s}{\alpha_d^2} b_i.$$

This allows us to bound $\sum_{j\in S} a_{ij}x_j(\phi^S)$ as follows.

$$\sum_{j\in S} a_{ij}x_j(\phi^S) \ge \sum_{j\in S} a_{ij}x_j^S - m \ge \frac{(1-2\epsilon_f)\epsilon_s}{\alpha_d^2} b_i,$$

where the first inequality follows from Claim 2, and the second follows from the fact that $B \ge \frac{m\alpha_d^2}{\epsilon_f\epsilon_s}$.

We discretize the values that the primal solution can take, as done in the feasibility argument in Lemma 5. However, we now let $q = \frac{\epsilon_f \min_i b_i}{4n\alpha_d}$. Consider two dual variables ϕ^{S_1} and ϕ^{S_2} in the same equivalence class, as defined in Definition 3. Similarly to Inequalities (17) and (18), we apply Lemma 3 to get

$$\left|\sum_{j\in S} a_{ij}x_j(\phi^{S_1}) - \sum_{j\in S} a_{ij}x_j(\phi^{S_2})\right| \le \sum_{j\in S} q \le \epsilon_s nq \le \frac{\epsilon_s\epsilon_f b_i}{4\alpha_d}, \tag{30}$$

$$\left|\sum_{j\in N} a_{ij}x_j(\phi^{S_1}) - \sum_{j\in N} a_{ij}x_j(\phi^{S_2})\right| < \sum_{j\in N} q < nq < \frac{\epsilon_f b_i}{4\alpha_d}. \tag{31}$$

For each equivalence class C of dual variables, and for each constraint i, we derive the following bound on

the probability of $\Pr[M_i \cap \bar{N}_i \cap \{\phi \in C\}]$. We employ the same approach used in the analogous argument in the feasibility proof of Lemma 5, Equations (19) and (20), where we fix a particular dual variable Φ^C within the equivalence class C. We bound

$$\begin{aligned}
&\Pr\left[\sum_{j\in S} a_{ij}x_j^S \geq \frac{(1-2\epsilon_f)\epsilon_s}{\alpha_d^2}b_i \cap \sum_{j\in N} a_{ij}x_j(\phi^S) < \frac{1-3\epsilon_f}{\alpha_d^2}b_i \cap \{\phi \in C\}\right] \\
&\leq \Pr\left[\sum_{j\in S} a_{ij}x_j(\Phi^C) \geq \frac{(1-2\epsilon_f)\epsilon_s}{\alpha_d^2}b_i \cap \sum_{j\in N} a_{ij}x_j(\Phi^C) < \frac{1-3\epsilon_f}{\alpha_d^2}b_i\right] \\
&\leq \Pr\left[\left|\sum_{j\in S} a_{ij}x_j(\Phi^C) - \mathbb{E}\left[\sum_{j\in N} a_{ij}x_j(\Phi^C)\right]\right| > \frac{\epsilon_f\epsilon_s}{2\alpha_d^2}b_i\right] \\
&\leq 2\exp\left(-\frac{\epsilon_f^2\epsilon_s b_i}{8\alpha_d^2 + 2\alpha_d\epsilon_f}\right).
\end{aligned}$$

Now, take α_d close to one. Concretely, we assume $10 \geq 8\alpha_d^2 + 2\alpha_d\epsilon_f$. We apply Lemma 4 for $q = \frac{\epsilon_f \min_i b_i}{4n\alpha_d}$ and find:

$$\begin{aligned}
&2\left(n\left(1+\frac{4n}{\epsilon_f b_i}\alpha_d\right)\right)^m \exp\left(-\frac{\epsilon_f^2\epsilon_s b_i}{10}\right) \\
&\leq 2\left(n\left(1+\frac{4\epsilon_f\epsilon_s n}{m\log(n)}\right)\right)^m \exp\left(-\frac{\epsilon_f^2\epsilon_s b_i}{10}\right) \leq \frac{2\epsilon_s}{m},
\end{aligned}$$

where the last line follows first because $b_i > \frac{m\log(n)}{\epsilon_f^2\epsilon_s\alpha_d}$, and then because of the assumption made on B (13). Taking the union bound over values of i, we find that $P(\cup_{i=1}^m (M_i \cap \bar{N}_i)) \leq 2\epsilon_s$. Finally, consider Lemma 6, for $r = \frac{1-3\epsilon_f}{\alpha_d^2}$. It follows that if x^* is an optimal solution to $\mathcal{L}$, then with probability at least $1-2\epsilon_s$, $\mathcal{P}(x(\phi^S)) \geq \frac{1-3\epsilon_f}{\alpha_d^2}\mathcal{P}(x^*)$. □

Finally, the proof of Theorem 1 follows from the above results.

Proof of Theorem 1. The solution generated by the allocation rule is feasible due to Lemma 5. The guarantee of a $(1-\epsilon_f)/\alpha_d^2$-approximation of the optimal solution follows from Lemma 7. For simplicity, we state the result of Lemma 7 with a rescaling of ϵ_f by 1/3. Concerning the run time, $\mathcal{A}$ is executed on a problem with $\epsilon_s n$ variables, and so it takes that fraction of the original run time. Then, the second step of the algorithm is n simple computations of $f_j'^{-1}(a_j^T\phi^S)$ for all $j \in [n]$. □

6. Conclusion

In this paper, we propose a framework for accelerating exact and approximate convex programming solvers for packing linear programming problems and a family of convex programming problems with linear constraints. Analytically, we provide worst-case guarantees on the run time and the quality of the solution produced. Numerically, we demonstrate that our framework speeds up Gurobi and SCS by two orders of magnitude, while maintaining a near-optimal solution.

Our framework works by subsampling columns of the data matrix and then defining a smaller sample problem defined on that subsampled matrix. We solve the dual of the sample problem using any given convex program solver in a black-box fashion. Finally, we set the values of the original primal variables based on the approximate dual solution of the sample problem.

Possible future areas of research include the following. In numerical experiments, we find that our algorithm can handle a larger family of problems than suggested by our theoretical bounds on B. Understanding this gap and improving the analysis is an area of interest. Additionally, our analysis relies partly on the fact that we are concerned with packing problems in this paper. It would be interesting to see what type of techniques are useful for more general problems.

Acknowledgments

The authors acknowledge Hanling Yi for work on the simulation study presented in Section 4.1, as well as Caltech undergraduates Irene Shen Wang and Maya Josyula for work on the simulation study presented in Section 4.2. This work was done in part while P. London was visiting Purdue University and while R. Eghbali was visiting the Simons Institute for the Theory of Computing.

References

Agrawal A, Klein P, Ravi R (1995) When trees collide: An approximation algorithm for the generalized Steiner problem on networks. *SIAM J. Comput.* 24(3):440–456.

Agrawal S, Devanur NR (2015) Fast algorithms for online stochastic convex programming. *SODA'15 Proc. 26th Annu. ACM-SIAM Sympos. Discrete Algorithms* (Society for Industrial and Applied Mathematics, Philadelphia), 1405–1424.

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

Allen-Zhu Z, Orecchia L (2015) Using optimization to break the epsilon barrier: A faster and simpler width-independent algorithm for solving positive linear programs in parallel. *SODA'15 Proc. 26th Annu. ACM-SIAM Sympos. Discrete Algorithms* (Society for Industrial and Applied Mathematics, Philadelphia), 1439–1456.

Andersen ED, Andersen KD (2000) The Mosek interior point optimizer for linear programming: An implementation of the homogeneous algorithm. Frenk H, Roos K, Terlaky T, Zhang S, eds. *High Performance Optimization, Applied Optimization*, vol. 33 (Springer US, Boston), 197–232.

Balakrishnan A, Magnanti TL, Wong RT (1989) A dual-ascent procedure for large-scale uncapacitated network design. *Oper. Res.* 37(5):716–740.

Bar-Yehuda R, Even S (1981) A linear-time approximation algorithm for the weighted vertex cover problem. *J. Algorithms* 2(2):198–203.

Barnhart C, Cohn A, Johnson E, Klabjan D, Nemhauser G, Vance P (2002) Airline crew scheduling. Hall RW, ed. *Handbook in Transportation Science, International Series in Operations Research & Management Science*, vol. 56 (Springer, Boston), 517–560.

Barnhart C, Johnson EL, Nemhauser GL, Savelsberg MWP, Vance P (2000) Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.* 48(3):318–326.

Bartal Y, Byers JW, Raz D (2004) Fast distributed approximation algorithms for positive linear programming with applications to flow control. *SIAM J. Comput.* 33(6):1261–1279.

Bertsimas D, Vohra R (1998) Rounding algorithms for covering problems. *Math. Program.* 80(1):63–89.

Borsos Z, Mutny M, Krause A (2020) Coresets via bilevel optimization for continual learning and streaming. *Adv. Neural Inform. Processing Systems* 33:34.

Borsos Z, Mutny M, Tagliasacchi M, Krause A (2021) Data summarization via bilevel optimization. Preprint, submitted September 26, https://arxiv.org/abs/2109.12534.

Boyd S, Vandenberghe L (2004) *Convex Optimization* (Cambridge University Press, Cambridge, UK).

Boyd S, Parikh N, Chu E, Peleato B, Eckstein J (2011) Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations Trends Machine Learning* 3(1):1–122.

Buchbinder N, Naor J (2009) The design of competitive online algorithms via a primal-dual approach. *Foundations Trends Theor. Comput. Sci.* 3(2-3):93–263.

Byers J, Nasser G (2000) Utility-based decision-making in wireless sensor networks. *First Annu. Workshop Mobile Ad Hoc Networking Comput. Mobihoc* (IEEE, Piscataway, NJ), 143–144.

Cevher V, Becker S, Schmidt M (2014) Convex optimization for big data: Scalable, randomized, and parallel algorithms for big data analytics. *IEEE Signal Processing Magazine* 31(5):32–43.

Chi Y, Lu YM, Chen Y (2019) Nonconvex optimization meets low-rank matrix factorization: An overview. *IEEE Trans. Signal Processing* 67(20):5239–5269.

Chowdhury A, London P, Avron H, Drineas P (2020) Faster randomized infeasible interior point methods for tall/wide linear programs. *Adv. Neural Inform. Processing Systems* 33:8704–8715.

Desaulniers G, Desrosiers J, Solomon M (2005) *Column Generation* (Springer, New York).

Devanur NR, Hayes TP (2009) The adwords problem: Online keyword matching with budgeted bidders under random permutations. *EC'09 Proc. 10th ACM Conf. Electronic Commerce* (Association for Computing Machinery, New York), 71–78.

Diamond S, Boyd S (2016) CVXPY: A Python-embedded modeling language for convex optimization. *J. Machine Learning Res.* 17(83):1–5.

Domahidi A, Chu E, Boyd S (2013) ECOS: An SOCP solver for embedded systems. *2013 Eur. Control Conf. ECC* (IEEE, Piscataway, NJ), 3071–3076.

Donoho DL (2006) Compressed sensing. *IEEE Trans. Inform. Theory* 52:1289–1306.

Donoho DL, Tanner J (2005) Sparse nonnegative solution of underdetermined linear equations by linear programming. *Proc. Natl. Acad. Sci. USA* 102(27):9446–9451.

Erlenkotter D (1978) A dual-based procedure for uncapacitated facility location. *Oper. Res.* 26(6):992–1009.

Esser E, Zhang X, Chan T (2010) A general framework for a class of first order primal-dual algorithms for convex optimization in imaging science. *SIAM J. Imaging Sci.* 3(4):1015–1046.

Gabay D, Mercier B (1976) A dual algorithm for the solution of nonlinear variational problems via finite element approximations. *Comput. Math. Appl.* 2(1):17–40.

Gilmore P, Gomory R (1963) A linear programming approach to the cutting-stock problem. *Oper. Res.* 11(6):863–888.

Goemans MX, Williamson DP (1995) A general approximation technique for constrained forest problems. *SIAM J. Comput.* 24(2): 296–317.

Gopalakrishnan B, Johnson E (2005) Airline crew scheduling: State-of-the-art. *Ann. Oper. Res.* 140:305–337.

Gupta A, Molinaro M (2016) How the experts algorithm can help solve LPs online. *Math. Oper. Res.* 41(4):1404–1431.

Ho-Nguyen N, Kilinc-Karzan F (2018) Online first-order framework for robust convex optimization. *Oper. Res.* 66(6):1670–1692.

Jalali A (2020) Persistent reductions in regularized loss minimization for variable selection. Preprint, submitted November 30, https://arxiv.org/abs/2011.14549.

Johansson M, Sternad M (2005) Resource allocation under uncertainty using the maximum entropy principle. *IEEE Trans. Inform. Theory* 51(12):4103–4117.

Kesselheim T, Tönnis A, Radke K, Vöcking B (2014) Primal beats dual on online packing LPs in the random-order model. *STOC'14 Proc. 46th Annu. ACM Sympos. Theory Comput.* (Association for Computing Machinery, New York), 303–312.

Kumar S, Ying J, de Miranda Cardoso JV, Palomar D (2019) Structured graph learning via Laplacian spectral constraints. *Adv. Neural Inform. Processing Systems* 32. https://papers.nips.cc/paper/2019/hash/90cc440b1b8caa520c562ac4e4bbcb51-Abstract.html.

Kyng R, Wang D, Zhang P (2020) Packing LPs are hard to solve accurately, assuming linear equations are hard. *Proc. 14th Annu. ACM-SIAM Sympos. Discrete Algorithms* (Society for Industrial and Applied Mathematics, Philadelphia), 279–296.

Leskovec J, Krevl A (2014) SNAP Datasets: Stanford large network data set collection. Accessed January 20, 2020, http://snap.stanford.edu/data.

London P, Chen N, Vardi S, Wierman A (2017) Distributed optimization via local computation algorithms. *ACM SIGMETRICS Performance Evaluation Rev.* 45(2):30–32.

London P, Vardi S, Wierman A (2019) Logarithmic communication for distributed optimization in multi-agent systems. *Proc. ACM Measurement Anal. Comput. Systems* 3(3):1–29.

London P, Vardi S, Wierman A, Yi H (2018) A parallelizable acceleration framework for packing linear programs. McIlraith SA, Weinberger KQ, eds. *AAAI'18/IAAI'18/EAAI'18 Proc. 32nd AAAI Conf. Artificial Intelligence 30th Innovative Appl. Artificial Intelligence Conf. Eighth AAAI Sympos. Ed. Adv. Artificial Intelligence* (AAAI Press, Palo Alto, CA), 3706–3713.

Luby M, Nisan N (1993) A parallel approximation algorithm for positive linear programming. *STOC'93 Proc. 25th Annu. ACM Sympos. Theory Comput.* (Association for Computing Machinery, New York), 448–457.

Mansour Y, Rubinstein A, Vardi S, Xie N (2012) Converting online algorithms to local computation algorithms. Czumaj A, Mehlhorn K, Pitts A, Wattenhofer R, eds. *ICALP 2012 Automata Languages Program., Lecture Notes in Computer Science*, vol. 7391 (Springer, Berlin), 653–664.

Martino AD, Martino DD (2018) An introduction to the maximum entropy approach and its application to inference problems in biology. *Heliyon.* 4(4): e00596.

Mohan K, London P, Fazel M, Witten D, Lee SI (2014) Node-based learning of multiple Gaussian graphical models. *J. Machine Learning Res.* 15:445–488.

Molinaro M, Ravi R (2013) The geometry of online packing linear programs. *Math. Oper. Res.* 39(1):46–59.

Nair V, Bartunov S, Gimeno F, von Glehn I, Lichocki P, Lobov I, O'Donoghue B, et al (2020) Solving mixed integer programs using neural networks. Preprint, submitted December 23, https://arxiv.org/abs/2012.13349.

Nedić A, Olshevsky A, Rabbat MG (2018) Network topology and communication-computation tradeoffs in decentralized optimization. *Proc. IEEE* 106(5):953–976.

Nemhauser GL (2012) Column generation for linear and integer programming. *Documenta Math.* Extra Volume ISMP: 65–73.

Nesterov Y (2005) Smooth minimization of non-smooth functions. *Math. Program.* 103(1):127–152.

O'Donoghue B, Chu E, Parikh N, Boyd S (2016) Conic optimization via operator splitting and homogeneous self-dual embedding. *J. Optim. Theory Appl.* 169:1042–1068.

Orlik P, Terao H (1992) *Arrangements of Hyperplanes, Grundlehren der Mathematischen Wissenschaften*, vol. 300 (Springer-Verlag, Berlin).

Pesnea P, Sadykov R, Vanderbeck F (2012) Feasibility pump heuristics for column generation approaches. Klasing R, ed. *Experimental Algorithms SEA 2012, Lecture Notes in Computer Science*, vol. 7276 (Springer, Berlin), 332–343.

Plotkin SA, Shmoys DB, Tardos E (1995) Fast approximation algorithms for fractional packing and covering problems. *Math. Oper. Res.* 20(2):257–301.

Ravikumar P, Agarwal A, Wainwright MJ (2010) Message passing for graph-structured linear programs: Proximal methods and rounding schemes. *J. Machine Learning Res.* 11:1043–1080.

Recht B, Fazel M, Parrilo PA (2010) Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM Rev.* 52(3):471–501.

Richert D, Cortés J (2015) Robust distributed linear programming. *IEEE Trans. Automatic Control.* 60(10):2567–2582.

Riquelme C, Johari R, Zhang B (2017) Online active linear regression via thresholding. AAAI'17 Proc. 31st AAAI Conf. Artificial Intelligence (AAAI Press, Palo Alto, CA), 2506–2512.

Sadykov R, Vanderbeck F (2013) Column generation for extended formulations. *EURO J. Comput. Optim.* 1(1–2):81–115.

Sanghavi S, Malioutov D, Willsky AS (2008) Linear programming analysis of loopy belief propagation for weighted matching. Platt JC, Koller D, Singer Y, Rowels ST, eds. *NIPS'07 Proc. 20th Internat. Conf. Neural Inform. Processing Systems* (Curran Associates, Red Hook, NY), 1273–1280.

Shi Y, Zhang J, O'Donoghue B, Letaief KB (2015) Large-scale convex optimization for dense wireless cooperative networks. *IEEE Trans. Signal Processing* 63(18):4729–4743.

Shim Jh, Kong K, Kang SJ (2021) Core-set sampling for efficient neural architecture search. *Proc. of ICML*.

Sridhar S, Wright SJ, Ré C, Liu J, Bittorf V, Zhang C (2013) An approximate, efficient LP solver for LP rounding. Burges CJC, Bottou L, Welling M, Ghahramani Z, Weinberger KQ, eds. *NIPS'13 Proc. 26th Internat. Conf. Neural Inform. Processing Systems*, vol. 2 (Curran Associates, Red Hook, NY), 2895–2903.

Sturm JF (1999) Using Sedumi 1.02, a Matlab toolbox for optimization over symmetric cones. *Optim. Methods Software* 1(1–4): 625–653.

Sun L, Fan Z, Fu X, Huang Y, Ding X, Paisley J (2019) A deep information sharing network for multi-contrast compressed sensing MRI reconstruction. *IEEE Trans. Image Processing* 28(12):6141–6153.

Taskar B, Chatalbashev V, Koller D (2004) Learning associative Markov networks. *ICML'04 Proc. 21st Internat. Conf. Machine Learning* (Association for Computing Machinery, New York), 102–112.

Toh KC, Todd MJ, Tutuncu RH (1999) SDPT3—a MATLAB software package for semidefinite programming, version 1.3. *Optim. Methods Software* 11(1–4):625–581.

Trevisan L (1998) Parallel approximation algorithms by positive linear programming. *Algorithmica* 21(1):72–88.

Tukan M, Maalouf A, Feldman D (2020) Coresets for near-convex functions. *Adv. Neural Inform. Processing Systems* 34:997–1009.

van der Vaart A, Wellner J (1996) *Weak Convergence and Empirical Processes with Applications to Statistics*, Springer Series in Statistics (Springer-Verlag, New York).

Vera A, Banerjee S (2021) The Bayesian prophet: A low-regret framework for online decision making. *Management Sci.* 67(3): 1368–1391.

Woodruff DP (2014) Sketching as a tool for numerical linear algebra. *Foundations Trends Theor. Comput. Sci.* 10(1–2):1–157.

Yarmish G, Slyke R (2009) A distributed, scalable simplex method. *J. Supercomputing* 49(3):373–381.

Young NE (2001) Sequential and parallel algorithms for mixed packing and covering. *Proc. 42nd IEEE Sympos. Foundations Comput. Sci.* (IEEE, Piscataway, NJ), 538–546.

Zurel E, Nisan N (2001) An efficient approximate allocation algorithm for combinatorial auctions. *EC'01 Proc. 3rd ACM Conf. Electronic Commerce* (Association for Computer Machinery, New York), 125–136.

Palma London received her PhD and MSc in Computer Science at Caltech. She is currently a postdoctoral researcher at Cornell. Her research broadly spans convex optimization, machine learning, and distributed algorithms.

Shai Vardi is an assistant professor of Management Information Systems at the Krannert School of Management at Purdue University. Previously, he was a Linde Postdoctoral Fellow at SISL at Caltech.

Reza Eghbali is a data science health innovation fellow at the University of California, San Francisco, and the University of California, Berkeley. He received his PhD in Electrical Engineering at the University of Washington. His research interest lies in the intersection of machine learning, neuroscience, and optimization algorithms.

Adam Wierman is a professor in the Department of Computing and Mathematical Sciences at Caltech. His research strives to make the networked systems that govern our world sustainable and resilient. He is best known for his work on the design of algorithms for sustainable data centers. He is a recipient of multiple awards, including the ACM Sigmetrics Rising Star award, the ACM Sigmetrics Test of Time award, and the IEEE Communications Society William R. Bennett Prize.