The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

A Parallelizable Acceleration Framework for Packing Linear Programs*

Palma London
California Institute of Technology
plondon@caltech.edu

Shai Vardi
California Institute of Technology
svardi@caltech.edu

Adam Wierman
California Institute of Technology
adamw@caltech.edu

Hanling Yi
The Chinese University of Hong Kong
yh014@ie.cuhk.edu.hk

Abstract

This paper presents an acceleration framework for packing linear programming problems where the amount of data available is limited, i.e., where the number of constraints m is small compared to the variable dimension n. The framework can be used as a black box to speed up linear programming solvers dramatically, by two orders of magnitude in our experiments. We present worst-case guarantees on the quality of the solution and the speedup provided by the algorithm, showing that the framework provides an approximately optimal solution while running the original solver on a much smaller problem. The framework can be used to accelerate exact solvers, approximate solvers, and parallel/distributed solvers. Further, it can be used for both linear programs and integer linear programs.

1 Introduction

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

LP solvers are at the core of many learning and inference problems, and often the linear programs of interest fall into the category of *packing problems*. Packing problems are linear programs of the following form:

$$\begin{aligned} \text{maximize} \quad & \textstyle\sum_{j=1}^{n} c_j x_j && && \text{(1a)} \\ \text{subject to} \quad & \textstyle\sum_{j=1}^{n} a_{ij} x_j \le b_i && i \in [m] && \text{(1b)} \\ & 0 \le x_j \le 1 && j \in [n] && \text{(1c)} \end{aligned}$$

where $A \in [0,1]^{m \times n}$, $b \in \mathbb{R}^m_{\ge 0}$, $c \in \mathbb{R}^n_{\ge 0}$.

Packing problems arise in a wide variety of settings, including max cut (Trevisan 1998), zero-sum matrix games (Nesterov 2005), scheduling and graph embedding (Plotkin, Shmoys, and Tardos 1995), flow controls (Bartal, Byers, and Raz 2004), auction mechanisms (Zurel and Nisan 2001), wireless sensor networks (Byers and Nasser 2000), and many other areas. In machine learning specifically, they show up in an array of problems, e.g., in applications of graphical models (Ravikumar, Agarwal, and Wainwright 2010), associative Markov networks (Taskar, Chatalbashev, and Koller 2004), and in relaxations of maximum a posteriori (MAP) estimation problems (Sanghavi, Malioutov, and Willsky 2008), among others.

In all these settings, practical applications require LP solvers to work at extreme scales and, despite decades of work, commercial solvers such as Cplex and Gurobi do not scale as desired in many cases. Thus, despite a large literature, the development of fast, parallelizable solvers for packing LPs is still an active direction.

Our focus in this paper is on a specific class of packing LPs for which data is either very costly, or hard to obtain. In these situations $m \ll n$; i.e., the number of data points m available is much smaller than the number of variables, n. Such instances are 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; Bienstock and Iyengar 2006) and also more generally in convex optimization and compressed sensing (Candes, Romberg, and Tao 2006; Donoho 2006), low rank matrix recovery (Recht, Fazel, and Parrilo 2010; Candes and Plan 2011), and graphical models (Yuan and Lin 2007a; Mohan et al. 2014).

Contributions of this paper. We present a black-box acceleration framework for LP solvers. When given a packing LP and an algorithm $\mathcal{A}$, the framework works by sampling an ϵ_s-fraction of the variables and using $\mathcal{A}$ to solve LP (1) restricted to these variables. Then, the dual solution to this sampled LP is used to define a thresholding rule for the primal variables of the original LP; the variables are set to either 0 or 1 according to this rule. The framework has the following key properties:

1. It can be used to accelerate exact or approximate LP-solvers (subject to some mild assumptions which we discuss below).
2. Since the original algorithm $\mathcal{A}$ is run only on a (much smaller) LP with ϵ_s-fraction of the variables, the framework provides a dramatic speedup.

*This work was supported in part by NSF grants AitF-1637598, CNS-1518941, CPS-154471, the Linde Institute, and the International Teochew Doctors Association Zheng Hanming Visiting Scholar Award Scheme.

3. The threshold rule can be used to set the values of the variables in parallel. Therefore, if $\mathcal{A}$ is a parallel algorithm, the framework gives a faster parallel algorithm with negligible overhead.

4. Since the threshold rule sets the variables to integral values, the framework can be applied without modification to solve integer programs that have the same structure as LP (1), but with integer constraints replacing (1c).

There are two fundamental tradeoffs 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 tradeoff involves feasibility. In order to ensure that the output of the framework is feasible w.h.p. (and not just that each constraint is satisfied in expectation), the constraints of the sample LP 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 a $(1+\delta)$ approximation to the optimal solution of the dual, we prove that the acceleration framework guarantees a $(1-\epsilon_f)/(1+\delta)^2$-approximation to the optimal solution of LP (1), under some assumptions about the input and ϵ_f. We formally state the result as Theorem 3.1, and note here that the result shows that ϵ_f grows proportionally to $1/\sqrt{\epsilon_s}$, which highlights that the framework maintains a high-quality approximation even when sample size is small (and thus the speedup provided by the framework is large).

The technical requirements for ϵ_f in Theorem 3.1 impose some restrictions on both the family of LPs that can be provably solved using our framework and the algorithms that can be accelerated. In particular, Theorem 3.1 requires $\min_i b_i$ to be large and the algorithm $\mathcal{A}$ to satisfy approximate complementary slackness conditions (see Section 2). While the condition on the b_i is restrictive, the condition on the algorithms is not – it is satisfied by most common LP solvers, e.g., 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 LP-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, e.g., 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 on a wide array of randomly generated packing LPs and obtaining solutions with $< 4\%$ relative error and a more than $150\times$ speedup. Other experiments with other solvers are qualitatively similar and are not included.

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 the thresholding rule is then applied by each clone in parallel, asynchronously. This improves 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. Incorporating "cloning" into the acceleration framework triples the speedup obtained, while reducing the error by 12%.

Summary of related literature. The approach underlying our framework is motivated by recent work that uses ideas from online algorithms to make offline algorithms more scalable, e.g., (Mansour et al. 2012; London et al. 2017a). A specific inspiration for this work is (Agrawal, Wang, and Ye 2014), which introduces an online algorithm that uses a two step procedure: it solves an LP based on the first s stages and then uses the 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, Wang, and Ye 2014) relies on exactly solving the LP given by the first s stages; considering approximate solutions of the sampled problem (as we do) adds complexity to the algorithm and analysis. Additionally, we can leverage the offline setting to fine-tune ϵ_f in order to optimize our solution while ensuring feasibility.

The sampling phase of our framework is reminiscent of the method of *sketching* in which the data matrix is multiplied by a random matrix in order to compress the problem and thus reduce computation time by working on a smaller formulation, e.g., see (Woodruff 2014). However, sketching is designed for overdetermined linear regression problems, $m \gg n$; thus compression is desirable. In our case, we are concerned with underdetermined problems, $m \ll n$; thus compression is not appropriate. Rather, the goal of sampling the variables is to be able to approximately determine the thresholds in the second step of the framework. This difference means the approaches are distinct.

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 e.g., (Boyd and Vandenberghe 2004). Recent work (Riquelme, Johari, and Zhang 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 under-determined scenario. Additionally, instead of sampling constraints, we sample variables.

The second stage of our algorithm is a thresholding step and 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 ILP; however we use thresholding to "extend" the solution of a sampled LP to the full

LP. The scheme we use is a deterministic threshold based on the complementary slackness condition. It is inspired by (Agrawal, Wang, and Ye 2014), but adapted to hold for approximate solvers rather than exact solvers. In this sense, the most related recent work is (Sridhar et al. 2013), which proposes a scheme for rounding an approximate LP solution. However, (Sridhar et al. 2013) uses 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, e.g., (Yarmish and Slyke 2009; Notarstefano and Bullo 2011; Burger et al. 2012; Richert and Cortés 2015). More specifically, there is significant interest in distributed strategies for approximately solving covering and packing linear problems, such as the problems we consider here, e.g., (Luby and Nisan 1993; Young 2001; Bartal, Byers, and Raz 2004; Awerbuch and Khandekar 2008; Allen-Zhu and Orecchia 2015).

2 A Black-Box Acceleration Framework

In this section we formally introduce our acceleration framework. At a high level, the framework accelerates an LP solver by running the solver in a black-box manner on a small sample of variables and then using a deterministic thresholding scheme to set the variables in the original LP. The framework can be used to accelerate any LP solver that satisfies the approximate complementary slackness conditions. The solution of an approximation algorithm $\mathcal{A}$ for a family of linear programs $\mathcal{F}$ 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* for $\mathcal{F}$. This terminology is non-standard, but is instructive when describing our results. It stems from a foundational result which states that an algorithm $\mathcal{A}$ that satisfies the above conditions is an α-approximation algorithm for any LP in $\mathcal{F}$ for $\alpha = \alpha_p \alpha_d$ (Buchbinder and Naor 2009).

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

Algorithm 1: Core acceleration algorithm

Input: Packing LP $\mathcal{L}$, LP solver $\mathcal{A}$, $\epsilon_s > 0$, $\epsilon_f > 0$
Output: $\hat{x} \in \mathbb{R}^n$

1. Select $s = \lceil n\epsilon_s \rceil$ primal variables uniformly at random. Label this set S.
2. Use $\mathcal{A}$ to find an (approximate) dual solution $\tilde{y} = [\phi, \psi] \in [\mathbb{R}^m, \mathbb{R}^s]$ to the sample LP.
3. Set $\hat{x}_j = x_j(\phi)$ for all $j \in [n]$.
4. Return $\hat{x}$.

Given a $(1, \alpha_d)$-approximation algorithm $\mathcal{A}$, the acceleration framework works in two steps. The first step is to sample a subset of the variables, $S \subset [n]$, $|S| = s = \lceil \epsilon_s n \rceil$, and use $\mathcal{A}$ to solve the following *sample LP*, which we call LP (2). For clarity, we relabel the variables so that the sampled variables are labeled $1, \ldots, s$.

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

$$\text{subject to} \quad \sum_{j=1}^{s} a_{ij} x_j \leq \frac{(1-\epsilon_f)\epsilon_s}{\alpha_d} b_i \quad i \in [m] \tag{2b}$$

$$x_j \in [0, 1] \quad j \in [s] \tag{2c}$$

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 a parameter that determines the fraction of the primal variables that are be sampled. Our analytic results give insight for setting ϵ_f and ϵ_s but, for now, both should be thought of as close to zero. Similarly, while the results hold for any α_d, they are most interesting when α_d is close to 1 (i.e., $\alpha_d = 1 + \delta$ for small δ). There are many such algorithms, given the recent interest in designing approximation algorithms for LPs, e.g., (Sridhar et al. 2013; Allen-Zhu and Orecchia 2015).

The second step in our acceleration framework uses the dual prices from the sample LP in order to set a threshold for a deterministic thresholding procedure, which is used to build the solution of LP (1). Specifically, let $\phi \in \mathbb{R}^m$ and $\psi \in \mathbb{R}^s$ denote the dual variables corresponding to the constraints (2b) and (2c) in the sample LP, respectively. We define the allocation (thresholding) rule $x_j(\phi)$ as follows:

$$x_j(\phi) = \begin{cases} 1 & \text{if } \sum_{i=1}^{m} a_{ij}\phi_i < c_j \\ 0 & \text{otherwise} \end{cases}$$

We summarize the core algorithm of the acceleration framework described above in Algorithm 1. When implementing the acceleration framework it is desirable to search for the minimal ϵ_f that allows for feasibility. This additional step is included in the full pseudocode of the acceleration framework given in Algorithm 2.

It is useful to make a few remarks about the generality of this framework. First, since the allocation rule functions as a thresholding rule, the final solution output by the accelerator is integral. Thus, it can be viewed as an ILP solver based on relaxing the ILP to an LP, solving the LP, and rounding the result. The difference is that it does not solve the

Algorithm 2: Pseudocode for the full framework.

Input: Packing LP $\mathcal{L}$, LP solver $\mathcal{A}$, $\epsilon_s > 0$, $\epsilon_f > 0$
Output: $\hat{x} \in \mathbb{R}^n$

```
Set ε_f = 0.
while ε_f < 1 do
    x̂ = Algorithm 1(L,A,ε_s,ε_f).
    if x̂ is a feasible solution to L then
        Return x̂.
    else
        Increase ε_f.
```

full LP, but only a (much smaller) sample LP; so it provides a significant speedup over traditional approaches. Second, the framework is easily parallelizable. The thresholding step can be done independently and asynchronously for each variable and, further, the framework can easily integrate speculative execution. Specifically, the framework can start multiple *clones* speculatively, i.e., take multiple samples of variables, run the algorithm $\mathcal{A}$ on each sample, and then round each sample in parallel. This provides two benefits. First, it improves the quality of the solution because the output of the "clone" with the best solution can be chosen. Second, it improves the running time since it curbs the impact of *stragglers*, tasks that take much longer than expected due to contention or other issues. Stragglers are a significant source of slowdown in clusters, e.g., nearly one-fifth of all tasks can be categorized as stragglers in Facebook's Hadoop cluster (Ananthanarayanan et al. 2013). There has been considerable work designing systems that reduce the impact of stragglers, and these primarily rely on speculative execution, i.e., running multiple clones of tasks and choosing the first to complete (Ananthanarayanan et al. 2010; 2014; Ren et al. 2015). Running multiple clones in our acceleration framework has the same benefit. To achieve both the improvement in solution quality and running time, the framework runs K clones in parallel and chooses the best solution of the first $k < K$ to complete. We illustrate the benefit of this approach in our experimental results in Section 3.

3 Results

In this section we present our main technical result, a worst-case characterization of the quality of the solution provided by our acceleration framework. We then illustrate the speedup provided by the framework through experiments using Gurobi, a state-of-the-art commercial solver.

3.1 A Worst-case Performance Bound

The following theorem bounds the quality of the solution provided by the acceleration framework. Let $\mathcal{L}$ be a packing LP with n variables and m constraints, as in (1), and $B := \min_{i\in[m]}\{b_i\}$. For simplicity, take $\epsilon_s n$ to be integral.

Theorem 3.1. *Let $\mathcal{A}$ be an $(1, \alpha_d)$-approximation algorithm for packing LPs, with runtime $f(n, m)$. For any $\epsilon_s > 0$ and $\epsilon_f \geq 3\sqrt{\frac{6(m+2)\log n}{\epsilon_s B}}$, Algorithm 1 runs in time $f(\epsilon_s n, m) + O(n)$ and obtains a feasible $(1-\epsilon_f)/\alpha_d^2$-approximation to the optimal solution for $\mathcal{L}$ with probability at least $1 - \frac{1}{n}$.*

Proof. The approximation ratio follows from Lemmas 4.2 and 4.7 in Section 4, with a rescaling of ϵ_f by 1/3 in order to simplify the theorem statement. The runtime follows from the fact that $\mathcal{A}$ is executed on an LP with $\epsilon_s n$ variables and at most m constraints and that, after running $\mathcal{A}$, the thresholding step is used to set the value for all n variables. □

The key trade-off in the acceleration framework is between the size of the sample LP, 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 3.1 quantifies this trade-off: ϵ_f grows as $1/\sqrt{\epsilon_s}$. Thus, ϵ_s can be kept small without impacting the loss in solution quality too much. The bound on ϵ_f in the theorem also defines the class of problems for which the accelerator is guaranteed to perform well—problems where $m \ll n$ 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.

Theorem 3.1 considers the acceleration of $(1, \alpha_d)$-approximation algorithms. As we have already noted, many common approximation algorithms fall into this class. Further, any exact solver satisfies this condition. For exact solvers, Theorem 3.1 guarantees a $(1 - \epsilon_f)$-approximation (since $\alpha_d = 1$).

In addition to exact and approximate LP solvers, our framework can also be used to convert LP solvers into ILP solvers, since the solutions it provides are always integral; and it can be parallelized easily, since the thresholding step can be done in parallel. We emphasize these points below.

Corollary 3.2. *Let $\mathcal{A}$ be an $(1, \alpha_d)$-approximation algorithm for packing LPs, with runtime $f(n, m)$. Consider $\epsilon_s > 0$, and $\epsilon_f \geq 3\sqrt{\frac{6(m+2)\log n}{\epsilon_s B}}$.*

- *Let $\mathcal{IL}$ be an integer program similar to LP (1) but with integrality constraints on the variables. Running Algorithm 1 on LP (1) obtains a feasible $(1-\epsilon_f)/\alpha_d^2$-approximation to the optimal solution for $\mathcal{IL}$ with probability at least $1 - \frac{1}{n}$ with runtime $f(\epsilon_s n, m) + O(n)$.*
- *If $\mathcal{A}$ is a parallel algorithm, then executing Algorithm 1 on p processors in parallel obtains a feasible $(1-\epsilon_f)/\alpha_d^2$-approximation to the optimal solution for $\mathcal{L}$ or $\mathcal{IL}$ with probability at least $1 - \frac{1}{n}$ and runtime $f_p(\epsilon_s n, m) + O(n/p)$, where $f_p(\epsilon_s n, m)$ denotes $\mathcal{A}$'s runtime for the sample program on p processors.*

3.2 Accelerating Gurobi

We illustrate the speedup provided by our acceleration framework by using it to accelerate Gurobi, a state-of-the-art commercial solver. Due to limited space, we do not present results applying the accelerator to other, more specialized, LP solvers; however the improvements shown here provide a conservative estimate of the improvements using parallel

implementations since the thresholding phase of the framework has a linear speedup when parallelized. 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 of Theorem 3.1 about 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 Theorem 3.1. This is also true with respect to the assumptions on the algorithm being accelerated. While 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 it to gradient algorithm such as (Sridhar et al. 2013) with results that parallel those presented for Gurobi below.

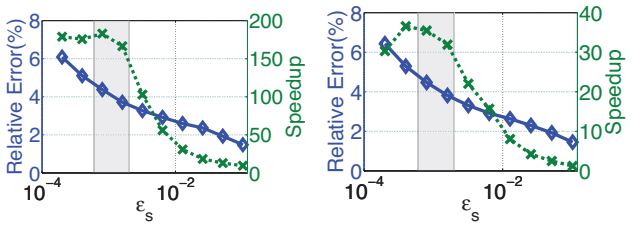
Experimental Setup. To illustrate the performance of our accelerator, we run Algorithm 2 on randomly generated LPs. Unless otherwise specified, the experiments use a matrix $A \in \mathbb{R}^{m \times n}$ of size $m = 10^2, n = 10^6$. Each element of A, denoted as a_{ij}, is first generated from $[0, 1]$ 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 $c \in \mathbb{R}^n$ is drawn i.i.d. from $[1, 100]$ uniformly. Each element of the vector $b \in \mathbb{R}^m$ 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 over 100 executions under different realizations of A, c.

To assess the quality of the solution, we measure the *relative error* and *speedup* of the accelerated algorithm as compared to the original algorithm. The relative error is defined as $(1 - Obj/OPT)$, where Obj is the objective value produced by our algorithm and OPT is the optimal objective value. The speedup is defined as the run time of the original LP solver divided by that of our algorithm.

We implement the accelerator in Matlab and use it to accelerate Gurobi. The experiments are run on a server with Intel E5-2623V3@3.0GHz 8 cores and 64GB RAM. 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).

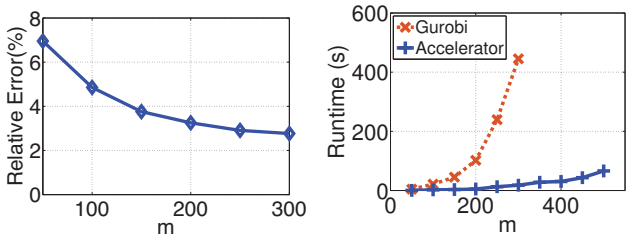
Experimental Results. Our experimental results highlight that our acceleration framework provides speedups of two orders of magnitude (over $150\times$), while maintaining high-quality solutions (relative errors of $< 4\%$).

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



(a) $p = 0.8$ (b) $p = 0.4$

Figure 1: Illustration of the relative error and speedup across sample sizes, ϵ_s. Two levels of sparsity, p, are shown.



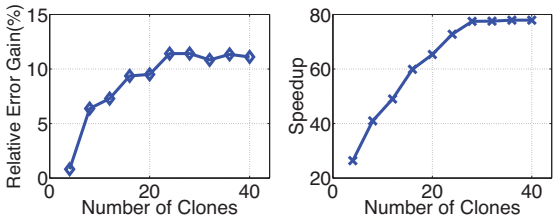
(a) Relative Error (b) Runtime

Figure 2: Illustration of the relative error and runtime as the problem size, m, grows.

small and the quality of the solution is still good, as indicated in the shaded region of Figure 1.

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 order of magnitude increase in the size of problems that can be solved. This is illustrated in Figure 2. The figure shows the runtime 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 (a) shows, one can choose ϵ_s more aggressively in large problems since 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 (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 runtime 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 8 GB 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%.

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



(a) Relative Error (b) Speedup

Figure 3: Illustration of the impact of cloning on solution quality as the number of clones grows.

clones (samples) in parallel and choosing the ones that finish the quickest. We illustrate the benefits associated with cloning in Figure 3. This figure shows the percentage gain in relative error and speedup associated with using different numbers of clones. In these experiments, 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 since our server only has 8 cores. The improvements would be larger than shown in Figure 3 in a system with more parallelism. Despite this conservative comparison, the improvements illustrated in Figure 3 are dramatic. Cloning reduces the relative error of the solution by 12% and triples the speedup. Note that these improvements are significant even though the solver we are accelerating is not a parallel solver.

4 Proofs

In this section we present the technical lemmas used to prove Theorem 3.1. The approach of the proof is inspired by the techniques in (Agrawal, Wang, and Ye 2014); however the analysis in our case is more involved. This is due to the fact that our result applies to approximate LP solvers while the techniques in (Agrawal, Wang, and Ye 2014) only apply to exact solvers. For example, this leads our framework to have three error parameters ($\epsilon_s, \epsilon_f, \alpha_d$) while (Agrawal, Wang, and Ye 2014) has a single error parameter.

The proof has two main steps: (1) show that the solution provided by Algorithm 1 is feasible with high probability (Lemma 4.2); and (2) show that the value of the solution is sufficiently close to optimal with high probability (Lemma 4.7). In both cases, we use the following concentration bound, e.g., (van der Vaart and Wellner 1996).

Theorem 4.1 (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 $\sigma_n^2 = \frac{1}{n}\sum_{j=1}^{n}(r_j - \sum_{j=1}^{n} r_j/n)^2$.*

Step 1: The solution is feasible

Lemma 4.2. *Let $\mathcal{A}$ be a (α_p, α_d)-approximation algorithm for packing LPs, $\alpha_p, \alpha_d \geq 1$. For any $\epsilon_s > 0$, $\epsilon_f \geq \sqrt{\frac{6(m+2)\log n}{\epsilon_s B}}$, the solution Algorithm 1 gives to LP (1) is feasible with probability at least $1 - 1/2n$, where the probability is over the choice of samples.*

Proof. Define a *price-realization*, $R(\phi)$, of a price vector ϕ as the set $\{r_{ij} = a_{ij}x_j(\phi), j \in [n], i \in [m]\}$ (note that $r_{ij} \in \{0, a_{ij}\}$) and denote, a "row" of $R(\phi)$ as $R_i(\phi) = \{r_{ij} = a_{ij}x_j(\phi), j \in [n]\}$. We say that $R_i(\phi)$ is *infeasible* if $\sum_{j\in[n]} r_{ij} > b_i$. The approach of this proof is to bound the probability that, for a given sample, the sample LP is feasible while there is some i for which $R_i(\phi)$ is not feasible in the original LP.

To begin, note that it naively seems that there are 2^n possible realizations of $R(\phi)$, over all possible price vectors ϕ, as $x_j \in \{0,1\}$. However, a classical result of combinatorial geometry (Orlik and Terao 1992) shows that there are only n^m possible realizations since each $R(\phi)$ is characterized by a separation of n points $(\{c_j, a_j\}_{j=1}^n)$ in an m-dimensional plane by a hyperplane, where a_j denotes the j-th column of A. The maximal number of such hyperplanes is n^m.

Next, we define a sample $S \subset [n]$, $|S| = \epsilon_s n$ as *R_i-good* if $\sum_{j\in S} r_{ij} \leq (1-\epsilon_f)\epsilon_s b_i$. Let $\tilde{x}$ be the solution to the sample LP for some sample S'. We say that S (possibly $S \neq S'$) is *$\tilde{x}_i$-good* if $\sum_{j\in S} a_{ij}\tilde{x}_j \leq \frac{(1-\epsilon_f)\epsilon_s}{\alpha_d} b_i$. The following claim relates these two definitions. Its proof is omitted due to space constraints, but can be found in the full version of this paper (London et al. 2017b).

Claim 4.3. *If a sample S is $\tilde{x}_i$-good then S is R_i-good.*

Next, fix the LP and $R(\phi)$. For the purpose of the proof, choose $i \in [n]$ uniformly at random. Next, we sample $\epsilon_s n$ elements without replacement from n variables taking the values $\{r_{ij}\}$. Call this sample S. Let $X = \sum_{j\in S} r_{ij}$ be the random variable denoting the sum of these random variables. Note that $\mathbb{E}[X] = \epsilon_s \sum_{j\in N} r_{ij}$, where the expectation is over the choice of S, and that the events $\sum_{j\in N} r_{ij} > b_i$ and $\mathbb{E}[X] > \epsilon_s b_i$ are equivalent. The probability that a sample is $\tilde{x}_i$-good and $R_i(\phi)$ is infeasible is

$$\Pr\left[\sum_{j\in S} a_{ij}\tilde{x}_j \leq \frac{(1-\epsilon_f)\epsilon_s}{\alpha_d} b_i \wedge \sum_{j\in N} r_{ij} > b_i\right]$$

$$\leq \Pr\left[\sum_{j\in S} r_{ij} \leq (1-\epsilon_f)\epsilon_s b_i \wedge \sum_{j\in N} r_{ij} > b_i\right] \quad (3)$$

$$\leq \Pr\left[\sum_{j\in S} r_{ij} \leq (1-\epsilon_f)\epsilon_s b_i \mid \sum_{j\in N} r_{ij} > b_i\right]$$

$$\leq \Pr\left[|X - \mathbb{E}[X]| > \epsilon_f \epsilon_s b_i\right]$$

$$\leq 2\exp\left(-\frac{\epsilon_f^2 \epsilon_s^2 b_i^2}{2\epsilon_s b_i + \epsilon_f \epsilon_s b_i}\right) \quad (4)$$

$$= 2\exp\left(-\frac{\epsilon_f^2 \epsilon_s b_i}{2+\epsilon_f}\right) \leq \frac{1}{2n^{m+2}}, \quad (5)$$

where (3) is due to Claim 4.3, (4) uses Theorem 4.1, and (5) uses the fact that $B \geq \frac{6(m+2)\log n}{\epsilon_s \epsilon_f^2}$.

To complete the proof, we now take a union bound over all possible realizations of R, which we bounded earlier by n^m, and values of i. □

Step 2: The solution is close to optimal

To prove that the solution is close to optimal we make two mild, technical assumptions.

Assumption 4.4. *For any dual prices $y = [\phi, \psi]$, there are at most m columns such that $\phi^T a_j = c_j$.*

Assumption 4.5. *Algorithm $\mathcal{A}$ maintains primal and dual solutions x and $y = [\phi, \psi]$ respectively with $\psi > 0$ only if $\sum_{j=1}^n a_{ij} x_j < c_j$.*

Assumption 4.4 does not always hold; however it can be enforced by perturbing each c_j by a small amount at random (see, e.g., (Devanur and Hayes 2009; Agrawal, Wang, and Ye 2014)). Assumption 4.5 holds for any "reasonable" $(1-\alpha_d)$-approximation dual ascent algorithm, and any algorithm that does not satisfy it can easily be modified to do so. These assumptions are used only to prove the following claim, which is used in the proof of the lemma that follows. The proof is omitted due to space restrictions, but can be found in the full version of this paper (London et al. 2017b).

Claim 4.6. *Let $\tilde{x}$ and $\tilde{y} = [\tilde{\phi}, \tilde{\psi}]$ be solutions of $\mathcal{A}$ to the sampled LP* (2)*. Then $\{x_j(\tilde{\phi})\}_{j\in[s]}$ and $\{\tilde{x}_j\}_{j\in[s]}$ differ on at most m values of j.*

Lemma 4.7. *Let $\mathcal{A}$ be a $(1, \alpha_d)$-approximation algorithm for packing LPs, $\alpha_d \geq 1$. For any $\epsilon_s > 0$, $\epsilon_f \geq \sqrt{\frac{6(m+2)\log n}{\epsilon_s B}}$, the solution Algorithm 1 gives to LP* (1) *is a $(1-3\epsilon_f)/\alpha_d^2$-approximation to the optimal solution with probability at least $1 - \frac{1}{2n}$, where the probability is over the choice of samples.*

Proof. Denote the primal and dual solutions to the sampled LP in (2) of Algorithm 1 by $\tilde{x}$, $\tilde{y} = [\tilde{\phi}, \tilde{\psi}]$. For purposes of the proof, we construct the following related LP.

$$\begin{aligned} \text{maximize} \quad & \textstyle\sum_{j=1}^n c_j x_j & & (6) \\ \text{subject to} \quad & \textstyle\sum_{j=1}^n a_{ij} x_j \leq \tilde{b}_i & i \in [m] \\ & x_j \in [0,1] & j \in [n], \end{aligned}$$

where

$$\tilde{b}_i = \begin{cases} \sum_{j=1}^n a_{ij} x_j(\tilde{\phi}) & \text{if } \tilde{\phi}_i > 0 \\ \max\{\sum_{j=1}^n a_{ij} x_j(\tilde{\phi}), b_i\} & \text{if } \tilde{\phi}_i = 0 \end{cases}$$

Note that $\tilde{b}$ has been set to guarantee that the LP is always feasible, and that $x(\tilde{\phi})$ and $y^* = [\tilde{\phi}, \psi^*]$ satisfy the (exact) complementary slackness conditions, where $\psi_j^* = c_j - \sum_{i=1}^m a_{ij}$ if $x_j(\tilde{\phi}) = 1$, and $\psi_j^* = 0$ if $x_j(\tilde{\phi}) \neq 1$. In particular, note that ψ^* preserves the exact complementary slackness condition, as ψ_j^* is set to zero when $x_j(\tilde{\phi}) \neq 1$. Therefore $x(\tilde{\phi})$ and $y^* = [\tilde{\phi}, \psi^*]$ are optimal solutions to LP (6).

A consequence of the approximate dual complementary slackness condition for the solution $\tilde{x}, \tilde{y}$ is that the i-th primal constraint of LP (2) is almost tight when $\tilde{\phi}_i > 0$:

$$\sum_{j\in S} a_{ij} \tilde{x}_j \geq \frac{(1-\epsilon_f)\epsilon_s}{(\alpha_d)^2} b_i.$$

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

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

where the first inequality follows from Claim 4.6 and the second follows from the fact that $B \geq \frac{m(\alpha_d)^2}{\epsilon_f \epsilon_s}$. Thus:

$$\begin{aligned} &\Pr\left[\sum_{j\in[s]} r_{ij} \geq \frac{(1-2\epsilon_f)\epsilon_s}{(\alpha_d)^2} b_i \wedge \sum_{j\in[n]} r_{ij} < \frac{1-3\epsilon_f}{(\alpha_d)^2} b_i\right] \\ &\leq \Pr\left[|X - \mathbb{E}[X]| > \frac{\epsilon_f \epsilon_s}{(\alpha_d)^2} b_i\right] \\ &\leq 2\exp\left(-\frac{\epsilon_f^2 \epsilon_s b_i}{2(\alpha_d)^4 + (\alpha_d)^2 \epsilon_f}\right) \leq \frac{1}{2n^{m+2}}. \end{aligned}$$

In the final step, we take α_d close to one, i.e., we assume $3 \geq 2(\alpha_d)^4 + (\alpha_d)^2 \epsilon_f$. The constant 6 in the lemma can be adjusted if application for larger α_d is desired.

Applying the union bound gives that, with probability at least $1 - \frac{1}{2n}$, it holds that $\tilde{b}_i \geq \sum_{j\in[n]} r_{ij} \geq \frac{(1-3\epsilon_f)}{(\alpha_d)^2} b_i$. It follows that, if x^* is an optimal solution to $\mathcal{L}$, then $\frac{(1-3\epsilon_f)}{(\alpha_d)^2} x^*$ is a feasible solution to LP (6). Thus, the optimal value of LP (6) is at least $\frac{(1-3\epsilon_f)}{(\alpha_d)^2} \sum_{j=1}^n c_j x_j^*$. □

References

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

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

Allen-Zhu, Z., and Orecchia, L. 2015. Using optimization to break the epsilon barrier: A faster and simpler width-independent algorithm for solving positive linear programs in parallel. In *Proc. of SODA*, 1439–1456.

Ananthanarayanan, G.; Kandula, S.; Greenberg, A. G.; Stoica, I.; Lu, Y.; Saha, B.; and Harris, E. 2010. Reining in the outliers in map-reduce clusters using mantri. In *Proc. of OSDI*.

Ananthanarayanan, G.; Ghodsi, A.; Shenker, S.; and Stoica, I. 2013. Effective straggler mitigation: Attack of the clones. In *Proc. of NSDI*, 185–198.

Ananthanarayanan, G.; Hung, M. C.-C.; Ren, X.; Stoica, I.; Wierman, A.; and Yu, M. 2014. Grass: Trimming stragglers in approximation analytics. In *Proc. of NSDI*, 289–302.

Awerbuch, B., and Khandekar, R. 2008. Stateless distributed gradient descent for positive linear programs. In *Proc. of STOC*, STOC '08, 691.

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

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

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

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

Bienstock, D., and Iyengar, G. 2006. Approximating fractional packings and coverings in o(1/epsilon) iterations. *SIAM J. Comp.* 35(4):825–854.

Boyd, S., and Vandenberghe, L. 2004. *Convex Optimization*. Cambridge University Press.

Buchbinder, N., and Naor, J. 2009. The design of competitive online algorithms via a primal-dual approach. *Found. and Trends in Theoretical Computer Science* 3(2-3):93–263.

Burger, M.; Notarstefano, G.; Bullo, F.; and Allgower, F. 2012. A distributed simplex algorithm for degenerate linear programs and multi-agent assignment. *Automatica* 48(9):2298–2304.

Byers, J., and Nasser, G. 2000. Utility-based decision-making in wireless sensor networks. In *Mobile and Ad Hoc Networking and Comp.*, 143–144.

Candes, E., and Plan, Y. 2011. Tight oracle inequalities for low-rank matrix recovery from a minimal number of noisy random measurements. *IEEE Trans. on Info. Theory* 57(4):2342–2359.

Candes, E.; Romberg, J.; and Tao, T. 2006. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Trans. Inform. Theory* 52(2):489 – 509.

Devanur, N. R., and Hayes, T. P. 2009. The adwords problem: online keyword matching with budgeted bidders under random permutations. In *Proc. of EC*, 71–78.

Donoho, D. L., and Tanner, J. 2005. Sparse nonnegative solution of underdetermined linear equations by linear programming. In *Proc. of the National Academy of Sciences of the USA*, 9446–9451.

Donoho, D. L. 2006. Compressed sensing. *IEEE Trans. Inform. Theory* 52:1289–1306.

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

Goemans, M. X., and Williamson, D. P. 1995. A general approximation technique for constrained forest problems. *SIAM J. on Comp.* 24(2):296–317.

London, P.; Chen, N.; Vardi, S.; and Wierman, A. 2017a. Distributed optimization via local computation algorithms. http://users.cms.caltech.edu/ plondon/loco.pdf.

London, P.; Vardi, S.; Wierman, A.; and Yi, H. 2017b. A parallelizable acceleration framework for packing linear programs. https://arxiv.org/abs/1711.06656.

Luby, M., and Nisan, N. 1993. A parallel approximation algorithm for positive linear programming. In *Proc. of STOC*, 448–457.

Mansour, Y.; Rubinstein, A.; Vardi, S.; and Xie, N. 2012. Converting online algorithms to local computation algorithms. In *Proc. of ICALP*, 653–664.

Mohan, K.; London, P.; Fazel, M.; Witten, D.; and Lee, S.-I. 2014. Node-based learning of multiple gaussian graphical models. *JMLR* 15:445–488.

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

Notarstefano, G., and Bullo, F. 2011. Distributed abstract optimization via constraints consensus: Theory and applications. *IEEE Trans. Autom. Control* 56(10):2247–2261.

Orlik, P., and Terao, H. 1992. *Arrangements of Hyperplanes*. Grundlehren der mathematischen Wissenschaften. Springer-Verlag Berlin Heidelberg.

Plotkin, S. A.; Shmoys, D. B.; and Tardos, E. 1995. Fast approximation algorithms for fractional packing and covering problems. *Math. of Oper. Res.* 20(2):257–301.

Ravikumar, P.; Agarwal, A.; and Wainwright, M. J. 2010. Message passing for graph-structured linear programs: Proximal methods and rounding schemes. *JMLR* 11:1043–1080.

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

Ren, X.; Ananthanarayanan, G.; Wierman, A.; and Yu, M. 2015. Hopper: Decentralized speculation-aware cluster scheduling at scale. In *Proc. of SIGCOMM*.

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

Riquelme, C.; Johari, R.; and Zhang, B. 2017. Online active linear regression via thresholding. In *Proc. of AAAI*.

Sanghavi, S.; Malioutov, D.; and Willsky, A. S. 2008. Linear programming analysis of loopy belief propagation for weighted matching. In *Proc. of NIPS*, 1273–1280.

Sridhar, S.; Wright, S. J.; Ré, C.; Liu, J.; Bittorf, V.; and Zhang, C. 2013. An approximate, efficient LP solver for LP rounding. In *Proc. of NIPS*, 2895–2903.

Taskar, B.; Chatalbashev, V.; and Koller, D. 2004. Learning associative markov networks. In *Proc. of ICML*.

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

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

Woodruff, D. P. 2014. Sketching as a tool for numerical linear algebra. *Found. and Trends in Theoretical Computer Science* 10(1-2):1–157.

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

Young, N. E. 2001. Sequential and parallel algorithms for mixed packing and covering. In *Proc. of FOCS*, 538–546.

Yuan, M., and Lin, Y. 2007a. Model selection and estimation in the gaussian graphical model. *Biometrika* 94(10):19–35.

Zurel, E., and Nisan, N. 2001. An efficient approximate allocation algorithm for combinatorial auctions. In *Proc. of EC*.