A Parallelizable Acceleration Framework for Packing Linear Programs

Palma London[1] Shai Vardi[1] Adam Wierman[1] Hanling Yi[2]
California Institute of Technology[1] The Chinese University of Hong Kong [2]

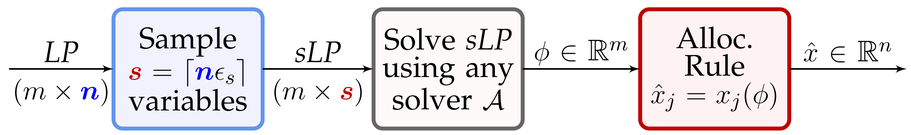
Goal: Black-Box Framework for Accelerating Linear Program Solvers

Speed up any LP solver $\mathcal{A}$ for a Packing Linear Program (LP)

$$\begin{array}{lll} \text{maximize} & \sum_{j=1}^n c_j x_j & \\ \text{subject to} & \sum_{j=1}^n a_{ij} x_j \le b_i & i \in [m] \\ & 0 \le x_j \le 1 & j \in [n] \end{array}$$

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

The Acceleration Framework



Acceleration Algorithm*

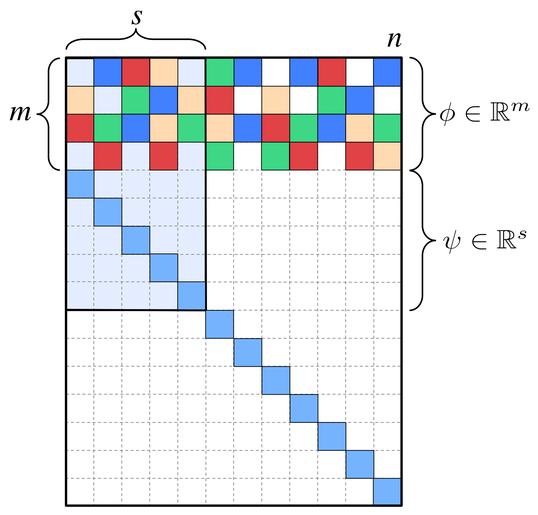
Input: Packing LP, Exact or approximate LP solver $\mathcal{A}$, $\epsilon_s > 0$, $\epsilon_f > 0$
Output: $\hat{x} \in \mathbb{R}^n$

1. Select $s = \lceil n\epsilon_s \rceil$ primal variables uniformly at random.
2. Use $\mathcal{A}$ to find the 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 $j \in [n]$.

*Inspired by Agrawal et al. [1]

Sampling Phase

Select a subset of the variables of size $s = \lceil n\epsilon_s \rceil$:



Define the following *sample linear program* (sLP):

$$\begin{array}{lll} \text{maximize} & \sum_{j=1}^s c_j x_j & \\ \text{subject to} & \sum_{j=1}^s a_{ij} x_j \le \frac{(1-\epsilon_f)\epsilon_s}{\alpha_d} b_i & i \in [m] \\ & x_j \in [0,1] & j \in [s] \end{array}$$

Parameters	
$\epsilon_s n$	sample size
ϵ_f	feasibility
α_d	solver $\mathcal{A}$

Allocation Rule

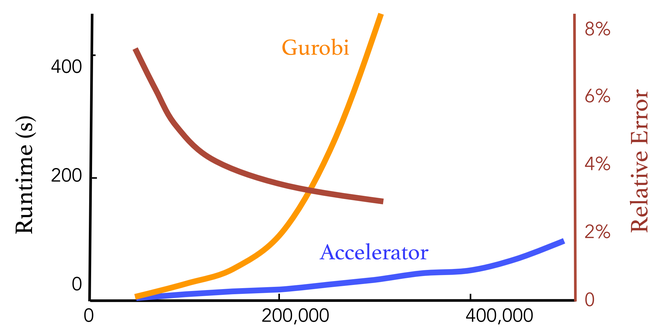
After obtaining the dual solution $\phi \in \mathbb{R}^m$ to (sLP), set variables $x_j \; \forall j \in [n]$:

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

$\sum_{i=1}^m a_{ij}\phi_i + \psi_j > c_j \implies \psi_j > 0 \overset{\text{C.S.}}{\implies} x_j = 1$ C.S.: $\psi_j(x_j - 1) = 0$

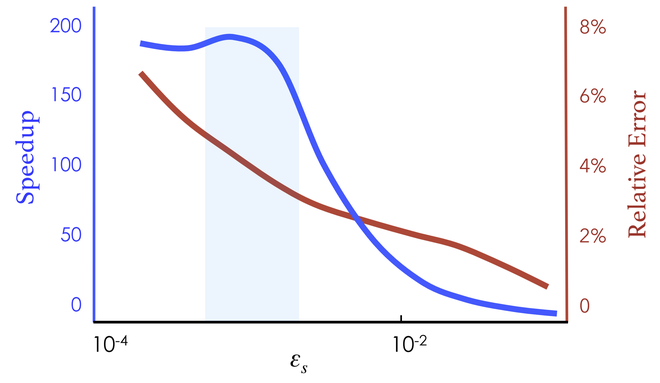
Accelerating Gurobi

Speed up by factor of 100
Error below 4%



n: number of variables

Gurobi's runtime grows until it runs into memory errors, while our algorithm's runtime grows slowly and can solve problems of much larger size. We set $n/m = 10^3$ and $\epsilon_s = 0.01$.



Relative error and runtime as the sample size varies, for fixed problem dimension $m = 10^2$ and $n = 10^6$. We set $\epsilon_f = 0$ and additively increase it.

Results

Theorem *Given an $(1, \alpha_d)$-approximation algorithm† $\mathcal{A}$ for packing LPs with runtime $f(n, m)$, for any*

$$\epsilon_s > 0, \qquad \epsilon_f \ge 3\sqrt{\tfrac{6(m+2)\log n}{\epsilon_s B}},$$

where $B := \min_{i \in [m]}\{b_i\}$, Algorithm (1) obtains a

1. *feasible $(1-\epsilon_f)/\alpha_d^2$-approximation to the optimal solution with probability at least $1 - \frac{1}{n}$,*
2. *runs in time $f(\epsilon_s n, m) + O(n)$.*

†An $(1, \alpha_d)$-approximation algorithm is one that produces approximate solutions guaranteeing that for $\alpha_d \ge 1$ and $i \in [m]$, if $y_i > 0$, then $b_i/\alpha_d \le \sum_{j=1}^n a_{ij}x_j \le b_i$.

Observations

1. The framework is well suited for problems for which $m \ll n$; the variable dimension is much larger than the number of data points.
2. The framework works for both Packing Linear Programs (LPs) and Packing Integer Linear Programs (ILPs).
3. Parameters ϵ_s and ϵ_f capture two fundamental tradeoffs:

Speed $\longleftarrow \epsilon_s \longrightarrow$ Quality of Solution
Quality of Solution $\longleftarrow \epsilon_f \longrightarrow$ Feasibility

References

[1] Shipra Agrawal, Zizhuo Wang, and Yinyu Ye. A dynamic near-optimal algorithm for online linear programming. *Oper. Res.*, 62(4):876–890, 2014.

[2] Karthik Mohan, Palma London, Maryam Fazel, Daniela Witten, and Su-In Lee. Node-based learning of multiple gaussian graphical models. *JMLR*, 15:445–488, 2014.

[3] Omer Reingold and Shai Vardi. New techniques and tighter bounds for local computation algorithms. *Journal of Computer and System Science*, 82(7):1180–1200, 2016.