Multi-Block ADMM - Managing Randomness and Data Privacy in Optimization Algorithm Design

EUROPT 2021

JULY 7-9, 2021

Yinyu Ye

Stanford University

Today’s talk

- Introduction to Multiblock-ADMM
- **Benefit of Data Exchange in Multi-Block ADMM Algorithm for Regression Estimation**
- Applying RAC-ADMM to Mixed Integer Programming

Introduction to ADMM

- Consider the following convex optimization problem

$$\begin{aligned} & \min \ f(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \in \boldsymbol{X} \end{aligned}$$

- Where f is a convex function, and $\boldsymbol{X}$ the Cartesian product of possibly non-convex, real, closed, nonempty sets.

- The corresponding augmented Lagrangian function is

$$L(\boldsymbol{x},\ \boldsymbol{\lambda})_{\boldsymbol{x} \in \boldsymbol{X}} = f(\boldsymbol{x}) - \boldsymbol{\lambda}^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) + \frac{\rho}{2}||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||_2^2$$

where $\boldsymbol{\lambda}$ is the Lagrangian multipliers or dual variables, and $\rho > 0$ is the step size.

Two-block ADMM with separable variables

- Consider the following optimization problem

$$\begin{aligned} \min \quad & f(\boldsymbol{x}) + g(\boldsymbol{s}) \\ \text{s.t.} \quad & \boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{B}\boldsymbol{s} = \boldsymbol{b} \\ & \boldsymbol{x} \in \boldsymbol{X},\ \boldsymbol{s} \in \boldsymbol{S} \end{aligned}$$

- The corresponding augmented Lagrangian function is

$$L(\boldsymbol{x},\ \boldsymbol{s},\ \boldsymbol{\lambda})_{x \in X,\, s \in S} = f(\boldsymbol{x}) + g(\boldsymbol{s}) - \boldsymbol{\lambda}^T(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{s} - \boldsymbol{b}) + \frac{\rho}{2}||\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{s} - \boldsymbol{b}||_2^2$$

Two-block ADMM with separable variables

- Two-block ADMM updates as follows

$$\text{ADMM} = \begin{cases} x^{k+1} & = \arg\min_{x\in\mathcal{X}} \mathcal{L}_\beta(x, s^k; \lambda^k) \\ s^{k+1} & = \arg\min_{s\in\mathcal{X}} \mathcal{L}_\beta(x^{k+1}, s^{k+1}; \lambda^k) \\ \lambda^{k+1} & = \lambda^k - \beta(A[x; s]^{k+1} - b) \end{cases}$$

- The two-block ADMM with separable objective is guarantee to converge.

Multi-block cyclic ADMM algorithm

- We could also partition the variables into multiple blocks. Let $\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_b]$

- Direct extension of multi-block (cyclic) ADMM updates as follows

$$\text{ADMM} = \begin{cases} x_1^{k+1} & = \arg\min_{x_1 \in \mathcal{X}} \mathcal{L}_\beta(x_1, \dots, x_b^k; \lambda^k) \\ \dots & \\ x_b^{k+1} & = \arg\min_{x_l \in \mathcal{X}} \mathcal{L}_\beta(x_1^{k+1}, \dots, x_b; \lambda^k) \\ & \\ \lambda^{k+1} & = \lambda^k - \beta(Ax^{k+1} - b) \end{cases}$$

Direct extension of three-block ADMM does not converge

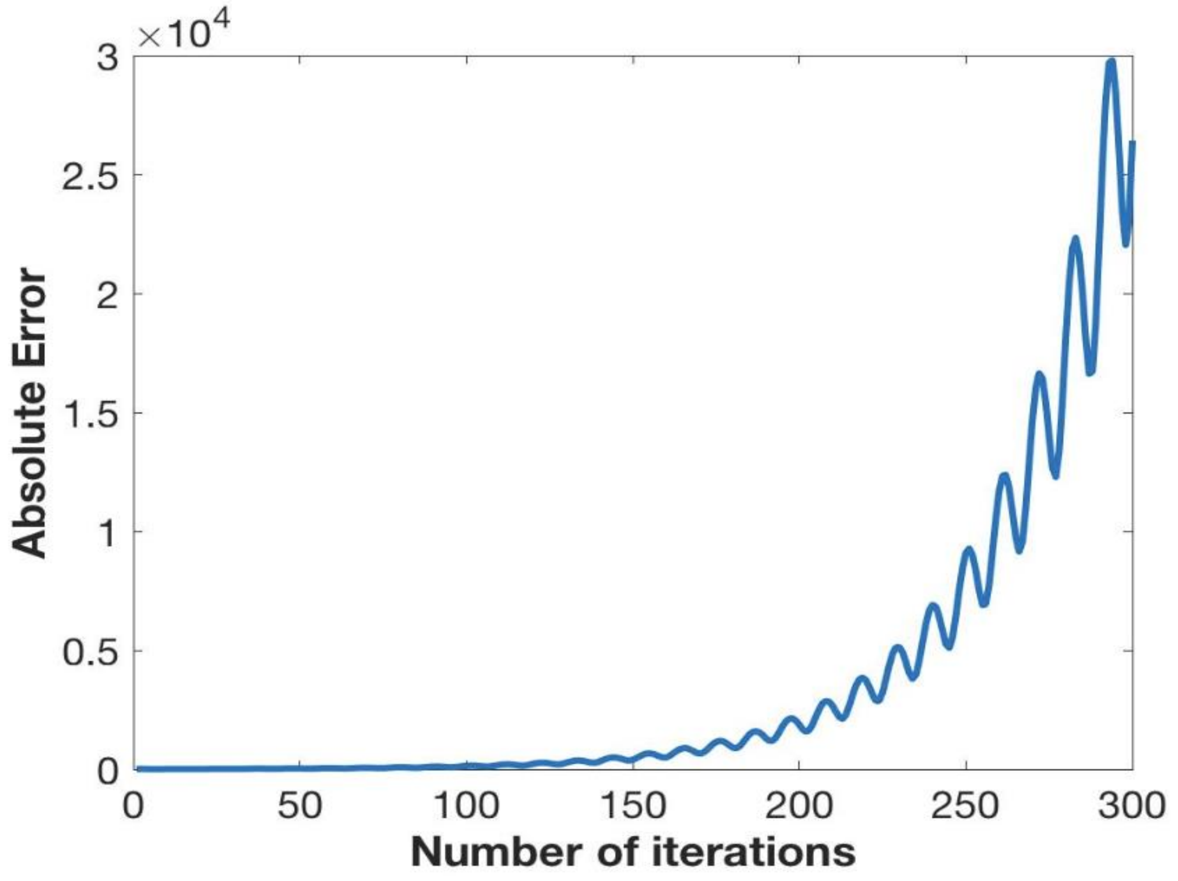


Figure 1: Non-singular system of square equations, three blocks (Chen et al. 2016)

Several variants of multi-block ADMM

- Consensus ADMM (Bertsekas 1982)
- Double sweep ADMM (Sun et al. 2015)
- Randomly Permuted ADMM (RP-ADMM) (Sun et al. 2015)
- Randomly Assembled Cyclic ADMM (RAC-ADMM) (Mihi´c et al. 2020)

Consensus or Variable-Splitting ADMM I

- Consensus ADMM introduces **auxiliaries** to each block of variables.

- Consider the following optimization problem with **separable** objective

$$\begin{aligned} &\min \ \Sigma_{i=1}^{b} f_i(\boldsymbol{x}_i) \\ \text{s.t.} \quad &\Sigma_{i=1}^{b} \boldsymbol{A}_i \boldsymbol{x}_i = \boldsymbol{b} \\ &\boldsymbol{x} \in \boldsymbol{X} \end{aligned}$$

- Primal Consensus ADMM reformulates the problem as

$$\begin{aligned} &\min \ \Sigma_{i=1}^{b} f_i(\boldsymbol{x}) \\ \text{s.t.} \quad &\Sigma_{i=1}^{b} \boldsymbol{y}_i = \boldsymbol{b} \\ &\boldsymbol{A}_i \boldsymbol{x}_i - \boldsymbol{y}_i = 0 \quad \forall i \\ &\boldsymbol{x} \in \boldsymbol{X} \end{aligned}$$

Consensus or Variable-Splitting ADMM II

- In each cycle of ADMM
  - Update all $\boldsymbol{x}_i$ (independently) as one block
  - Update all y$_i$ together as one block with a closed-form solution
  - Update the multipliers the same way

- Consensus ADMM is **guaranteed to converge** with any fixed step-size.

- However, it suffers from **slow** convergence.

Double-Sweep ADMM

In each cycle of ADMM

- Sequentially update $\boldsymbol{x}_1$,…, $\boldsymbol{x}_b$ followed by updating $\boldsymbol{x}_{b-1}$,…, $\boldsymbol{x}_1$ in the **reversed order**.
- Update the multipliers the same way.

Randomly Permuted - ADMM (RP - ADMM)

In each cycle of ADMM

- **Randomly generate a permutation** of 1,…,b; and following the permutation order to update $\boldsymbol{x}_i$ sequentially.
- Update the multipliers the same way.
- RP - ADMM is guaranteed to converge in expectation.

Randomly Assembled Cyclic - ADMM (RAC-ADMM)

- In each cycle, stead of cyclic updating, double sweep updating, or updating with block-wise random permutation, RAC-ADMM first **randomly assemble blocks of all primal variables**, then sequentially updates each of the blocks.

- Update the multipliers the same way.

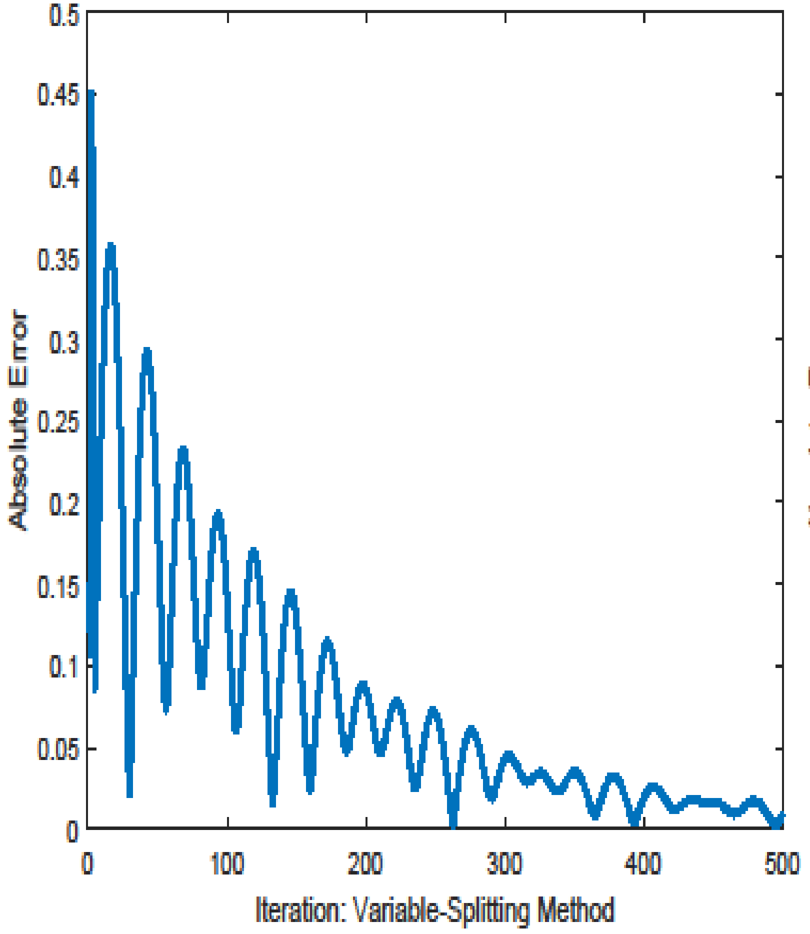
- RAC - ADMM is guaranteed to converge in **expectation**.

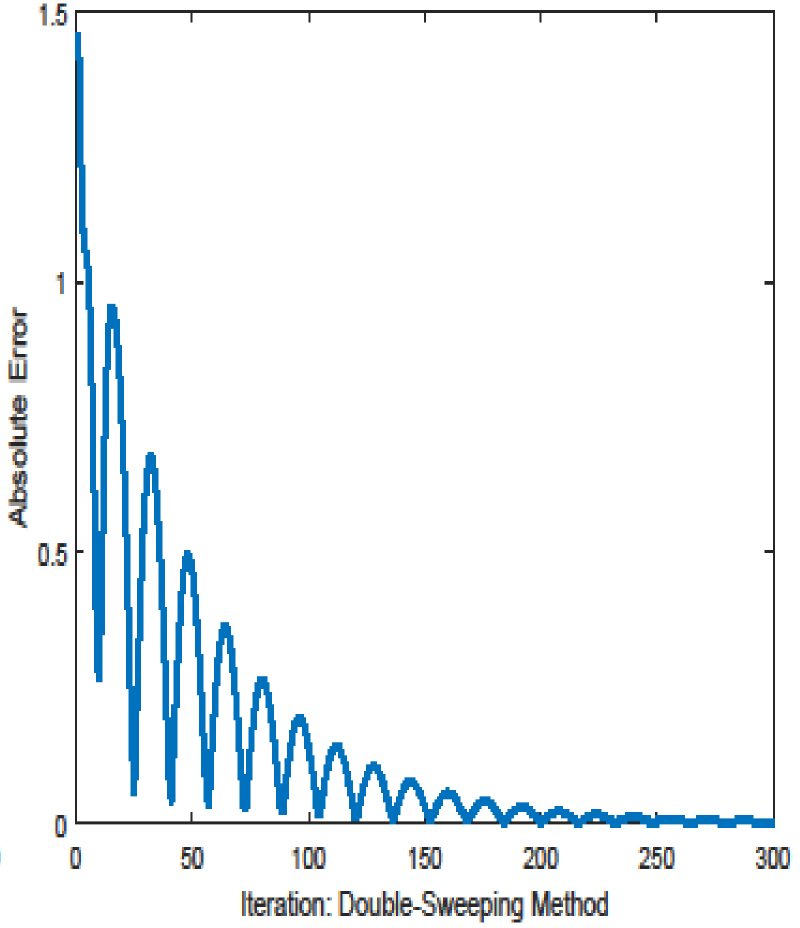
RAC-ADMM

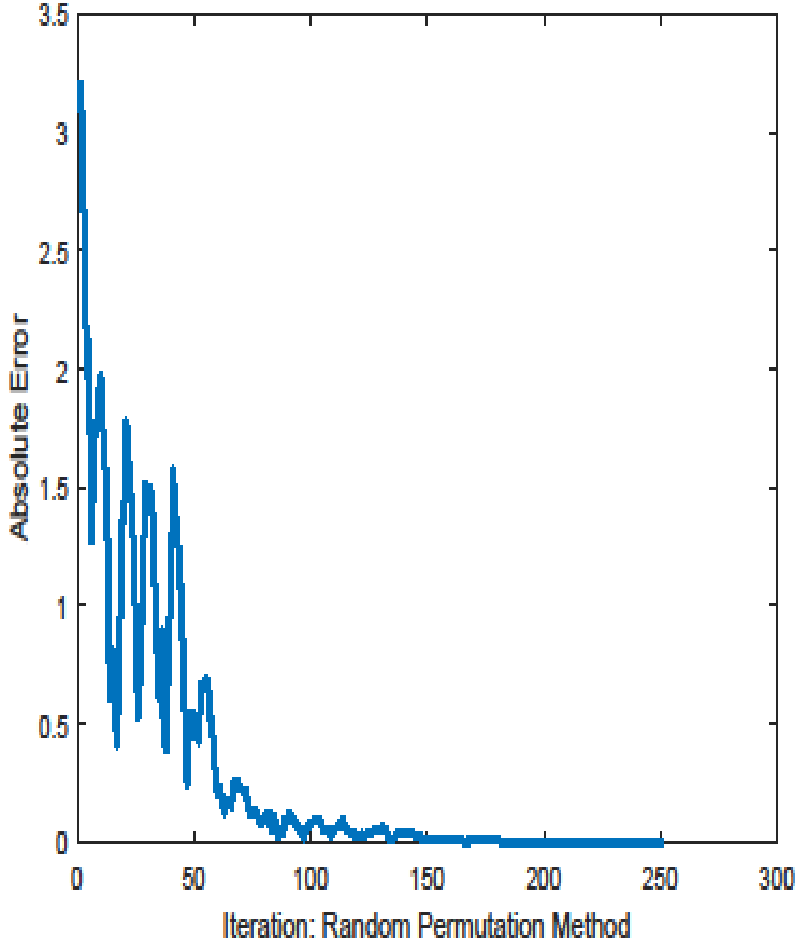
- RAC-ADMM can be viewed as introducing **double randomness** compared with RP-ADMM: permuting **block-wise order** + grouping **variables in blocks**

$$\text{RAC} = \begin{cases} \text{Randomly (without replacement) group primal variables into blocks } x_i \\ \text{For } i = 1, \dots, l, \text{ compute } x_i^{k+1} \text{ by:} \\ \qquad x_i^{k+1} = \arg\min_{x_i \in \mathcal{X}} \mathcal{L}_\beta(x_1^{k+1}, \dots, x_{i-1}^{k+1}, x_i^k, x_{i+1}^k, \dots, x_l^k; s^k; \lambda^k; \mu^k) \\ \lambda^{k+1} = \lambda^k - \beta(Ax^{k+1} - b) \end{cases}$$

Performance on the diverging example

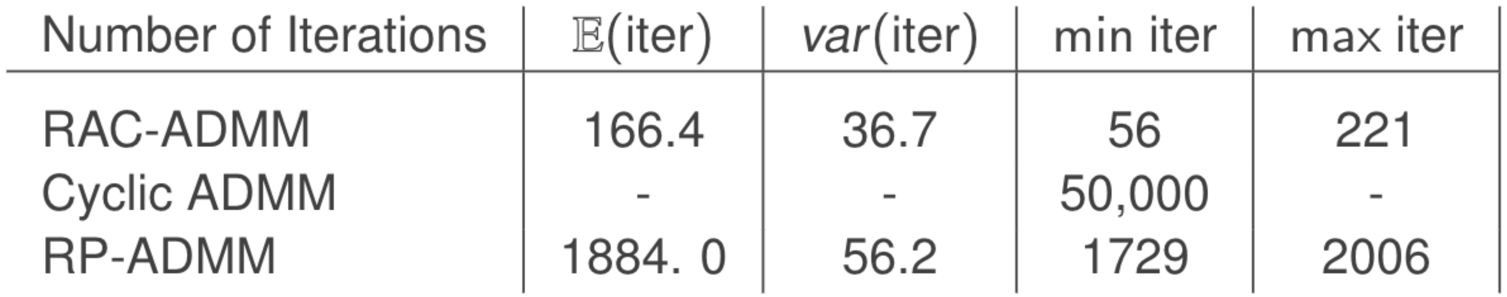






RAC – ADMM Performance

- RAC - ADMM **overcomes the slow convergence** issue of cyclic-ADMM and RP-ADMM



Number of Iterations	$\mathbb{E}$(iter)	*var*(iter)	min iter	max iter
RAC-ADMM	166.4	36.7	56	221
Cyclic ADMM	-	-	50,000	-
RP-ADMM	1884. 0	56.2	1729	2006

Table 2: $n = 3000$, $p = 50$, tolerance$= |Ax - b|_1 = 10^{-6}$

- RAC - ADMM is **guaranteed to converge in expectation**.
- However, RAC - ADMM is **not** guaranteed to converge **almost surely**, and there are problem instances that RAC - ADMM may diverge/oscillates.

Today’s talk

- **Introduction to Multiblock-ADMM**
- **Benefit of Data Exchange in Multi-Block ADMM Algorithm for Regression Estimation**
  - **Mingxi Zhu, Graduate School of Business, Stanford University**
  - **Yinyu Ye, Management Science & Engineering, Stanford University**
- **Applying RAC-ADMM to Mixed Integer Programming**

Statistical learning across decentralized data centers

- Decentralized Learning : Method that learns or trains an algorithm across multiple **decentralized centers** holding **local data**.
- Pros: Such method protects **data privacy** and **data security**.
- Cons: Many decentralized learning algorithms suffers from **slow convergence**.

Statistical Learning Model

- Each **center** i possess **model data matrix** $\boldsymbol{X}_i \in R^{s\times p}$ and **dependent variable** vector $\boldsymbol{y}_i \in R^{s\times 1}$.

- Let $\left(\boldsymbol{x}_{i,j}, y_{i,j}\right)$ be the j^{th} data pair of the i^{th} data **center.**

- The decision maker tries to find the **global estimator** $\boldsymbol{\beta} \in R^{p\times 1}$ that minimizes the following **loss function**

  - $\Sigma_{i=1}^{b} \Sigma_{j=1}^{s} f\left((\boldsymbol{x}_{i,j}, y_{i,j}); \boldsymbol{\beta}\right)$

where $f\left((\boldsymbol{x}_{i,j}, y_{i,j}); \boldsymbol{\beta}\right)$ is the **loss function**.

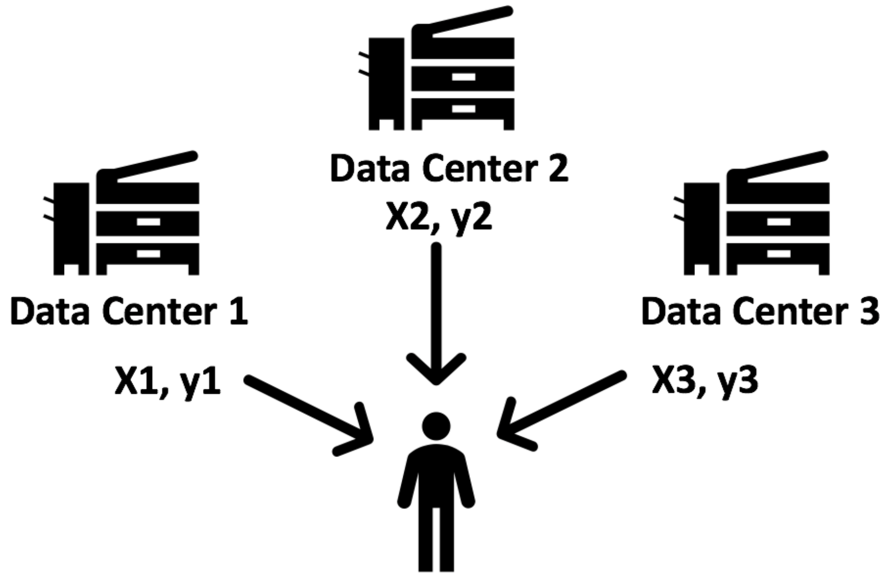
Commonly used loss functions

- Commonly used loss function are **convex** in $\boldsymbol{\beta}$, including
  - Least Square
    $$f((\boldsymbol{x}, \boldsymbol{y}); \boldsymbol{\beta}) = ||\boldsymbol{x}\boldsymbol{\beta} \text{-} y||_2^2$$
  - Ridge
    $$f((\boldsymbol{x}, \boldsymbol{y}); \boldsymbol{\beta}) = ||\boldsymbol{x}\boldsymbol{\beta} \text{-} y||_2^2 + \alpha||\boldsymbol{\beta}||_2^2$$
  - Lasso
    $$f((\boldsymbol{x}, \boldsymbol{y}); \boldsymbol{\beta}) = ||\boldsymbol{x}\boldsymbol{\beta} \text{-} y||_2^2 + \alpha||\boldsymbol{\beta}||_1$$
  - Elastic Net
    $$f((\boldsymbol{x}, \boldsymbol{y}); \boldsymbol{\beta}) = ||\boldsymbol{x}\boldsymbol{\beta} \text{-} \boldsymbol{y}||_2^2 + \alpha||\boldsymbol{\beta}||_1 + (1 - \alpha)\ ||\boldsymbol{\beta}||_2^2$$
  - Logistic
    $$f((\boldsymbol{x}, \boldsymbol{y}); \boldsymbol{\beta}) = log(1 - exp(-y\boldsymbol{x}\boldsymbol{\beta}))$$

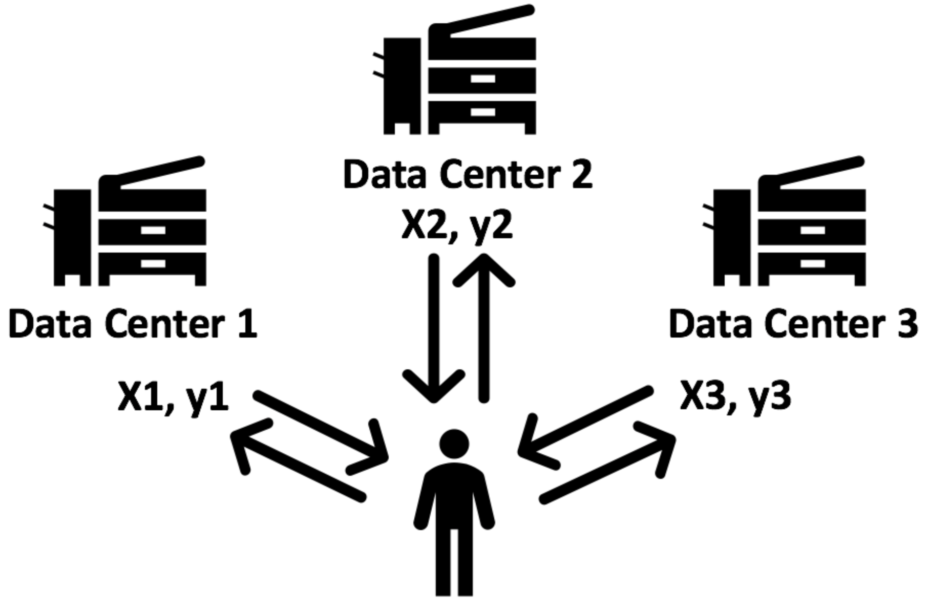
Statistical learning across decentralized data centers

- Centralized Learning
- All local data are uploaded to one

- Decentralized Learning
- Local data cannot be exchanged



Decision Maker Receives
X = [X1; X2; X3]
Y = [y1; y2; y3]
And trains in one server



Decision Maker trains local data in local servers, pools the training results and aggregates the results without accessing data

ADMM in decentralized learning

- Consensus/distributed ADMM is widely used in **decentralized learning**.
- Compared with another commonly used algorithm, Stochastic Gradient Descend (SGD), consensus ADMM is more robust in **step-size** choice, and it is guaranteed to converge for any choice of step-size.

Algorithm : Consensus ADMM

- Introducing **local estimators** $\boldsymbol{\beta}_i$ to each center and reformulate the problem as

$$\sum_{i=1}^{b} \sum_{j=1}^{s} f((\mathbf{x}_{i,j}, y_{i,j}); \boldsymbol{\beta}_i)$$

$$s.t. \quad \boldsymbol{\beta}_i - \boldsymbol{\beta} = 0 \quad \forall\, i = 1, \ldots, b$$

- Let λ_i be the dual with respect to the constraint $\boldsymbol{\beta}_i - \boldsymbol{\beta} = \mathbf{0}$, and ρ_p be the step-size to the primal consensus ADMM, the augmented Lagrangian is given by

$$L(\boldsymbol{\beta}_i, \boldsymbol{\beta}, \boldsymbol{\lambda}_i) = \sum_{i=1}^{b} \sum_{j=1}^{s} f((\mathbf{x}_{i,j}, y_{i,j}); \boldsymbol{\beta}_i) + \sum_{i=1}^{b} \boldsymbol{\lambda}_j^T (\boldsymbol{\beta}_i - \boldsymbol{\beta}) + \sum_{i=1}^{b} \frac{\rho_p}{2} (\boldsymbol{\beta}_i - \boldsymbol{\beta})^T (\boldsymbol{\beta}_i - \boldsymbol{\beta})$$

Least Square Regression

- Specifically, when $f((\boldsymbol{x}, y); \boldsymbol{\beta}) = ||\boldsymbol{x}\boldsymbol{\beta} - y||_2^2$, the problem becomes a linearly constrained quadratic optimization.

$$\min_{\boldsymbol{\beta}_i, \boldsymbol{\beta}} \sum_{i=1}^{b} \frac{1}{2}(\mathbf{X}_i\boldsymbol{\beta}_i - \mathbf{y}_i)^T(\mathbf{X}_i\boldsymbol{\beta}_i - \mathbf{y}_i)$$

$$s.t.\ \boldsymbol{\beta}_i - \boldsymbol{\beta} = 0 \qquad \forall\ i$$

- Let $\boldsymbol{D}_i = \boldsymbol{X}_i^T\boldsymbol{X}_i$ and $\boldsymbol{c}_i = -\boldsymbol{X}_i^T\boldsymbol{y}_i$, primal consensus ADMM becomes

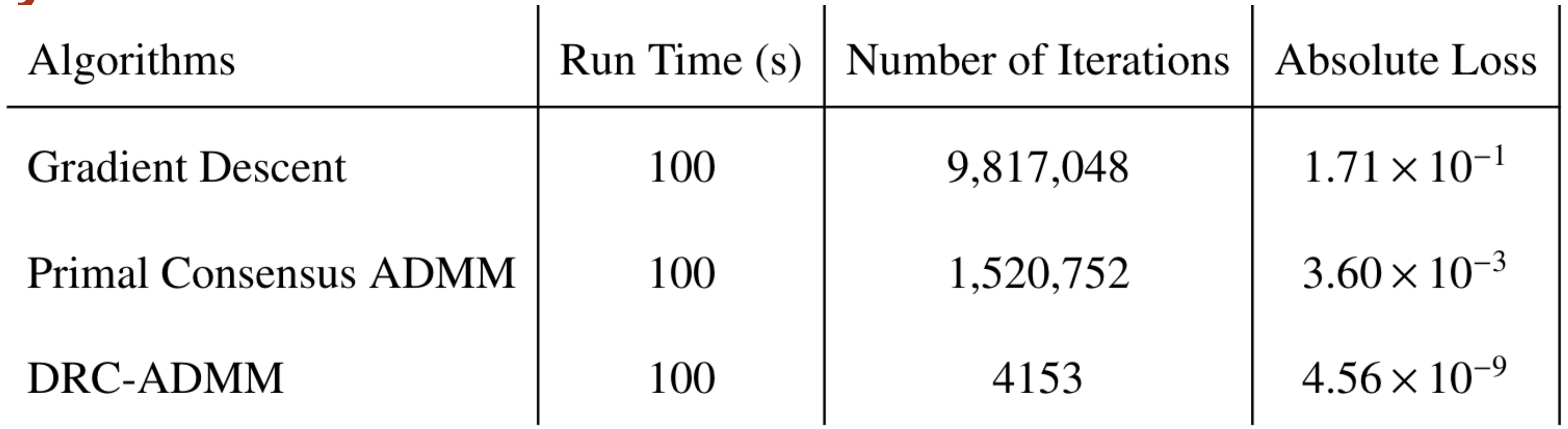
$$(\rho_p\mathbf{I} + \mathbf{D}_i)\boldsymbol{\beta}_i^{t+1} = \rho_p\boldsymbol{\beta}^t - \boldsymbol{\lambda}_i^t - \mathbf{c}_i$$

$$\boldsymbol{\beta}^{t+1} = \frac{1}{b}\sum_i \boldsymbol{\beta}_i^{t+1} + \frac{1}{b\rho_p}\sum_i \boldsymbol{\lambda}_i^t$$

$$\boldsymbol{\lambda}_i^{t+1} = \boldsymbol{\lambda}_i^t + \rho_p(\boldsymbol{\beta}_i^{t+1} - \boldsymbol{\beta}^{t+1})$$

Consensus ADMM suffers from slow convergence

- While consensus ADMM **does not exchange local data**, and enjoys benefit from **parallel computing**, it suffers from **slow convergence**.
- The following table reports the performance of GD, primal consensus ADMM, and Dual Randomly-Assembled Cyclic ADMM (**DRC-ADMM**), which we designed to carefully balance the trade-off between **data privacy** and **efficiency**.



Algorithms	Run Time (s)	Number of Iterations	Absolute Loss
Gradient Descent	100	9,817,048	1.71×10^{-1}
Primal Consensus ADMM	100	1,520,752	3.60×10^{-3}
DRC-ADMM	100	4153	4.56×10^{-9}

Table 1 : Algorithm Performances on UCI machine learning repository (Dua and Graff(2017)) regression data YearPrediction-MSD (Chang and Lin (2011)) with number of observations n=463,715, and number of features p=90, number of local data centers = 4. Table 1 report the number of iterations, and absolute L2 loss defined by $AL = ||\boldsymbol{\beta}^* - \boldsymbol{\beta}'||_2$. For ADMM method the step-size we set equals to 1 and for GD the step-size is optimally chosen.

Why data exchange helps convergence

- Consider the following example with $n = 4,\ p = 1,$ and number of data center $b = 2$.

$$\tilde{\mathbf{X}}_1 = \begin{bmatrix} 0.99 \\ 0.01 \end{bmatrix}, \quad \tilde{\mathbf{X}}_2 = \begin{bmatrix} 0.9 \\ 0.1 \end{bmatrix},$$

- Normalized the model matrix $\boldsymbol{X}$ by the Frobenius norm $||\boldsymbol{X}||_F$

$$\mathbf{X}_1 = \begin{bmatrix} 0.7379 \\ 0.0075 \end{bmatrix}, \quad \mathbf{X}_2 = \begin{bmatrix} 0.6708 \\ 0.0745 \end{bmatrix}$$

Why data exchange helps convergence

- The convergence rate under the previous data structure is **0.6661**, and one can show that if model matrix is normalized, with number of data centers equals to 2, the upper bound of convergence rate (worst case convergence rate) is **0.6667**.
- Convergence rate α: $\limsup_{k\to\infty} \frac{1}{k} \log \|x_k - \mathbf{1}_N \otimes x_\star\| = \log \alpha$
- However, if we apply **data exchange**

$$\mathbf{X}_1 = \begin{bmatrix} 0.7379 \\ 0.0075 \end{bmatrix}, \quad \mathbf{X}_2 = \begin{bmatrix} 0.6708 \\ 0.0745 \end{bmatrix}$$

- The convergence rate becomes **0.5264**, and one can show that the lower bound of convergence rate (best case of convergence rate) is **0.5000**.

Theory on worst case data structure

- Generally, under the following assumption

Assumption 1. *The regressor matrix* $\mathbf{X}$ *is normalized by its Frobenius norm* $\|\mathbf{X}\|_F$, *and the smallest and largest eigenvalue of* $\mathbf{X}^T\mathbf{X}$, $\underline{q}$ *and* $\bar{q}$ *are fixed, with* $\mathbf{X}_i^T\mathbf{X}_i > 0$ *for all* $i \in \{1, \ldots, b\}$.

- When step-size is relatively large, the worst-case data structure and the upper bound of convergence rate of consensus ADMM is given by

Theorem 1. *For* $\rho_p > \bar{q}$, *the convergence rate of distributed ADMM is upper bounded by* $\frac{b\rho_p}{b\rho_p+\underline{q}}$, *and the upper bound is achieved when* $\mathbf{D}_i = \mathbf{D}_j$ *for all* $i, j \in \{1, \ldots, b\}$.

Several Remarks

- The upper bound on convergence speed is increasing with respect to number of data centers and decreasing with smallest eigenvalue of covariance model matrix. This implies that **a greater number of data centers** and **ill conditioning of matrix** hurt convergence.

- Intuitively, for large step size, ADMM converges faster when each block **“differs”** from one another significantly.
  - When updating dual, ADMM takes average of all local estimators, which are essentially, the product of inverse matrix and vector. When each block differs from one another, it creates more momentum for dual updating .

- When applying **data augmentation** in machine learning with ADMM based optimization algorithm, one need to be careful as data augmentation are more likely to creates similar blocks.

Comparison between Gradient Descend

- Gradient Descend is also widely used in decentralized learning, with the bounds of distributed ADMM, we could compare performance between the two algorithms under different step-size.

Proposition 2. *For* $\rho_p \in (0, s_1) \cup (s_2, \infty)$, $\rho(M_p) < \rho(M_{GD})$, *where*

$$s_1 = \min\left(\frac{1}{\underline{q}} - \bar{q}, q_1\right), \qquad s_2 = \frac{2b - \bar{q}\underline{q} + \sqrt{4b^2 + (\bar{q}\underline{q})^2}}{2b\bar{q}}$$

- Consensus ADMM is indeed more robust in step-size choice, and for gradient descend method, there is only a small range of sweet spot of step-size that leads to faster convergence.

Introducing Dual Randomly-Assembled Cyclic ADMM

- Inspired by Mihi´c et al. (2020), we introduce the Dual Randomly-assembled Cyclic ADMM (**DRC-ADMM**)
- the least square regression problem is equivalent as

$$\min_{\zeta} \ \tfrac{1}{2}\zeta^T\zeta$$

$$s.t. \ \mathbf{X}\beta - \mathbf{y} = \zeta$$

- Let $\boldsymbol{t}$ be the dual variables, the dual is given by

$$\min_{\mathbf{t}} \ \tfrac{1}{2}\mathbf{t}^T\mathbf{t} + \mathbf{y}^T\mathbf{t}$$

$$s.t. \ \mathbf{X}^T\mathbf{t} = 0$$

- the dual variables serves as a label for each (potentially) exchanged data pair, and the randomization is more effective in the dual space.

Introducing Dual Randomly-Assembled Cyclic ADMM

Data Center 1

$(\boldsymbol{x}_{1,1}, y_{1,1}), (\boldsymbol{x}_{1,2}, y_{1,2}), (\boldsymbol{x}_{1,3}, y_{1,3});$

Local data

Data Center 2

$(\boldsymbol{x}_{2,1}, y_{2,1}), (\boldsymbol{x}_{2,2}, y_{2,2}), (\boldsymbol{x}_{2,3}, y_{2,3});$

Local data

Data Center 3

$(\boldsymbol{x}_{3,1}, y_{3,1}), (\boldsymbol{x}_{3,2}, y_{3,2}), (\boldsymbol{x}_{3,3}, y_{3,3});$

Local data

Global Data Pool

Introducing Dual Randomly-Assembled Cyclic ADMM

Data Center 1

$(\boldsymbol{x}_{1,1}, y_{1,1}), (\boldsymbol{x}_{1,2}, y_{1,2}),$

Local data

Data Center 2

$(\boldsymbol{x}_{2,1}, y_{2,1}),$ $(\boldsymbol{x}_{2,3}, y_{2,3});$

Local data

Data Center 3

$(\boldsymbol{x}_{3,2}, y_{3,2}), (\boldsymbol{x}_{3,3}, y_{3,3});$

Local data

$\longrightarrow$ Cyclic updating

$\longrightarrow$ Cyclic updating

Global Data Pool

$(\boldsymbol{x}_{1,3}, y_{1,3});$ $(\boldsymbol{x}_{2,2}, y_{2,2}),$ $(\boldsymbol{x}_{3,1}, y_{3,1}),$

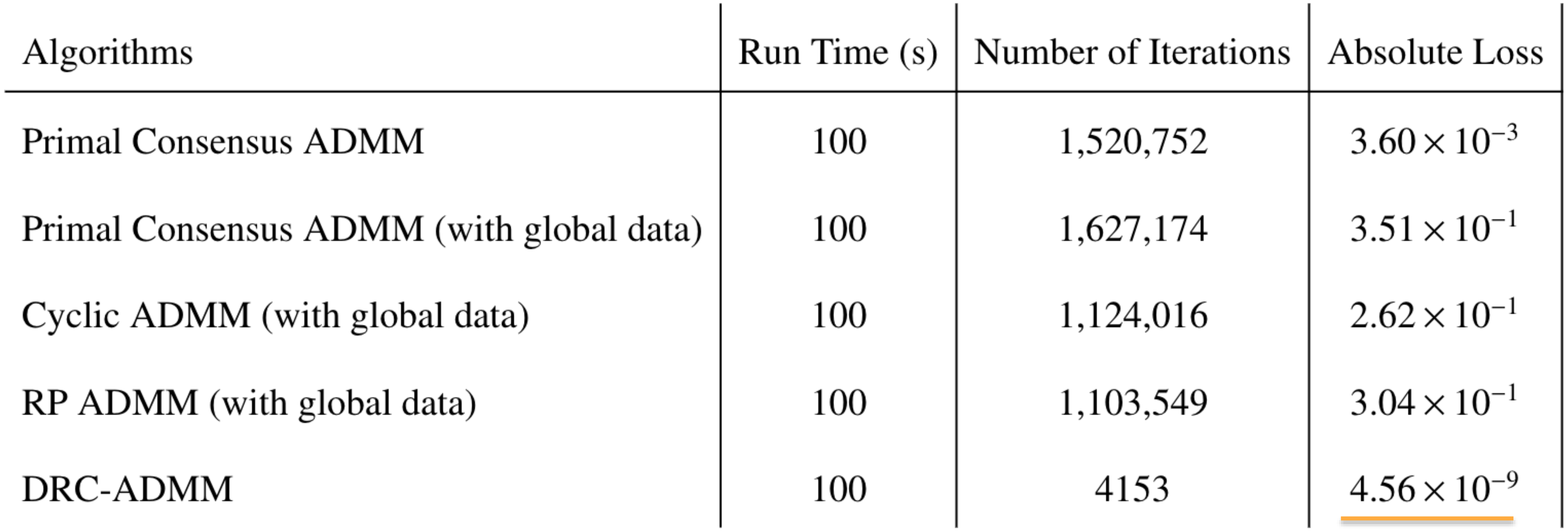
Introducing Dual Randomly-Assembled Cyclic ADMM

```
Algorithm 2 DRC-ADMM
Initialization: $t = 0$, step size $\rho_d \in \mathbb{R}^+$ $\mathbf{t}_t \in \mathbb{R}^n$, $\boldsymbol{\beta}_t \in \mathbb{R}^p$, and stopping rule $\tau$

Randomly select $\alpha\%$ of total observations. Let $\mathbf{r} = [r^1, \ldots, r^m]$ be the index of selected data ($m = \lfloor \alpha\% n \rfloor$).
Let $\mathbf{r}^i = \mathbf{B}^i \cap \mathbf{r}$ be the index of selected data belongs to data center $i$.
while $t \leq \tau$ do
    Random permute $\mathbf{r}$ to $\sigma_t(\mathbf{r})$, partition $\sigma_t(\mathbf{r}) = [\sigma_t^1(\mathbf{r}), \ldots, \sigma_t^b(\mathbf{r})]$ according to $|r^i|$ (size of $\mathbf{r}^i$).
    For $i = 1, \ldots, b$
        Let $\boldsymbol{\sigma}_t^i = (\mathbf{B}^i \cap \overline{\mathbf{r}^i}) \cup \sigma_t^i(\mathbf{r})$ Center $i$ updates $\mathbf{t}_{t+1}^{\boldsymbol{\sigma}_i^t} = \arg\min_{BFt \in \mathbb{R}^{|B_i|}} L(\mathbf{t}_{t+1}^{\boldsymbol{\sigma}_1^t}, \ldots, \mathbf{t}_{t+1}^{\boldsymbol{\sigma}_{i-1}^t}, \mathbf{t}, \mathbf{t}_t^{\boldsymbol{\sigma}_{i+1}^t}, \ldots, \mathbf{t}_t^{\boldsymbol{\sigma}_b^t}, \boldsymbol{\beta}^t)$
    Decision maker updates $\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \rho_d \mathbf{X}^T \mathbf{t}_{t+1}$
end
Output: $\boldsymbol{\beta}_\tau$ as global estimator
```

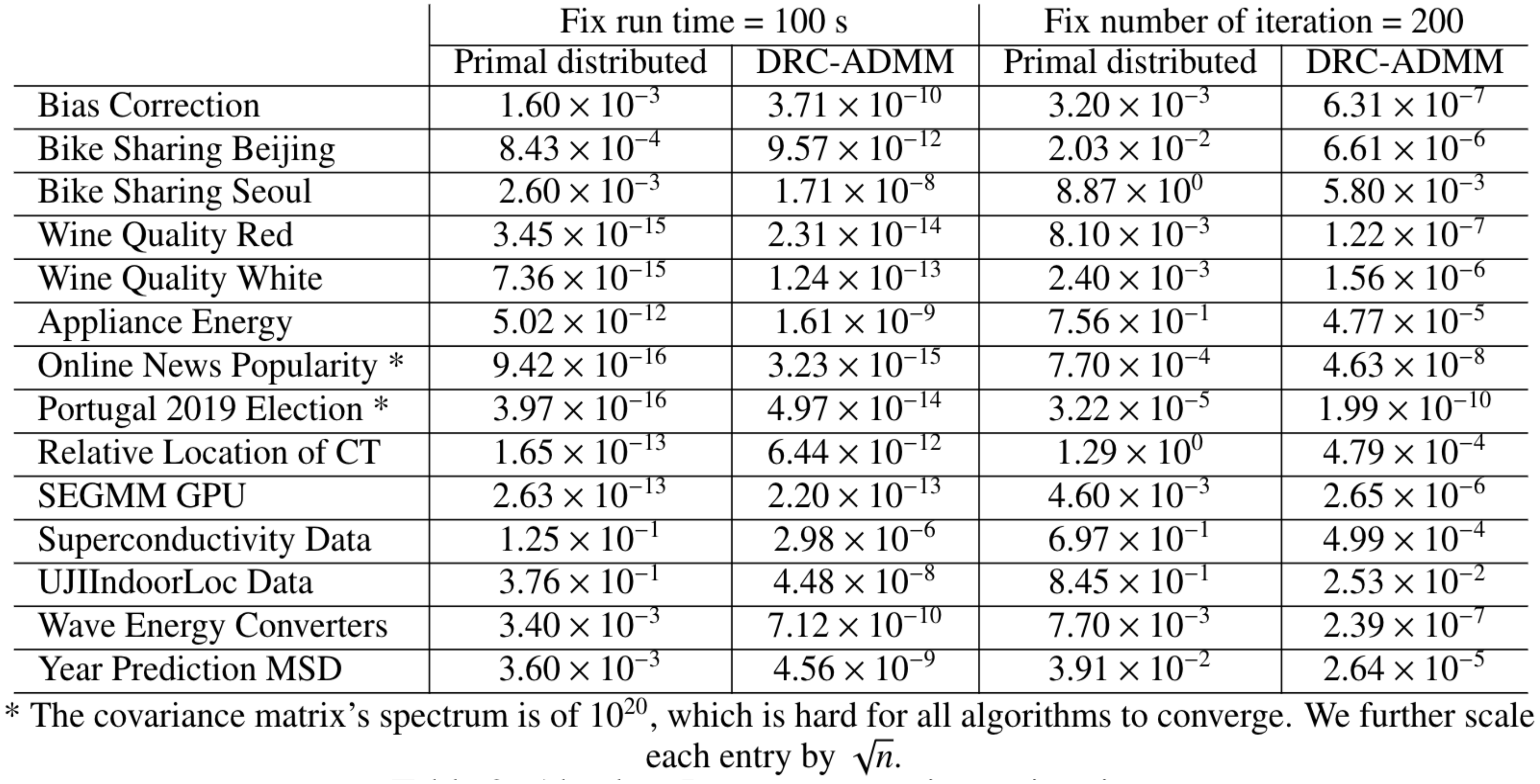
Data exchange is essential

- If each the time we directly add all global data to each of the block (here the data structure at each block is **fixed**), and compare
    - Distributed ADMM with global data
    - Cyclic ADMM with global data
    - Randomly Permuted ADMM with global data



Algorithms	Run Time (s)	Number of Iterations	Absolute Loss
Primal Consensus ADMM	100	1,520,752	3.60×10^{-3}
Primal Consensus ADMM (with global data)	100	1,627,174	3.51×10^{-1}
Cyclic ADMM (with global data)	100	1,124,016	2.62×10^{-1}
RP ADMM (with global data)	100	1,103,549	3.04×10^{-1}
DRC-ADMM	100	4153	4.56×10^{-9}

Numerical Results on UCI ML regression repository



	Fix run time = 100 s		Fix number of iteration = 200	
	Primal distributed	DRC-ADMM	Primal distributed	DRC-ADMM
Bias Correction	1.60×10^{-3}	3.71×10^{-10}	3.20×10^{-3}	6.31×10^{-7}
Bike Sharing Beijing	8.43×10^{-4}	9.57×10^{-12}	2.03×10^{-2}	6.61×10^{-6}
Bike Sharing Seoul	2.60×10^{-3}	1.71×10^{-8}	8.87×10^{0}	5.80×10^{-3}
Wine Quality Red	3.45×10^{-15}	2.31×10^{-14}	8.10×10^{-3}	1.22×10^{-7}
Wine Quality White	7.36×10^{-15}	1.24×10^{-13}	2.40×10^{-3}	1.56×10^{-6}
Appliance Energy	5.02×10^{-12}	1.61×10^{-9}	7.56×10^{-1}	4.77×10^{-5}
Online News Popularity *	9.42×10^{-16}	3.23×10^{-15}	7.70×10^{-4}	4.63×10^{-8}
Portugal 2019 Election *	3.97×10^{-16}	4.97×10^{-14}	3.22×10^{-5}	1.99×10^{-10}
Relative Location of CT	1.65×10^{-13}	6.44×10^{-12}	1.29×10^{0}	4.79×10^{-4}
SEGMM GPU	2.63×10^{-13}	2.20×10^{-13}	4.60×10^{-3}	2.65×10^{-6}
Superconductivity Data	1.25×10^{-1}	2.98×10^{-6}	6.97×10^{-1}	4.99×10^{-4}
UJIIndoorLoc Data	3.76×10^{-1}	4.48×10^{-8}	8.45×10^{-1}	2.53×10^{-2}
Wave Energy Converters	3.40×10^{-3}	7.12×10^{-10}	7.70×10^{-3}	2.39×10^{-7}
Year Prediction MSD	3.60×10^{-3}	4.56×10^{-9}	3.91×10^{-2}	2.64×10^{-5}

* The covariance matrix's spectrum is of 10^{20}, which is hard for all algorithms to converge. We further scale each entry by $\sqrt{n}$.

Numerical Results on UCI ML regression repository

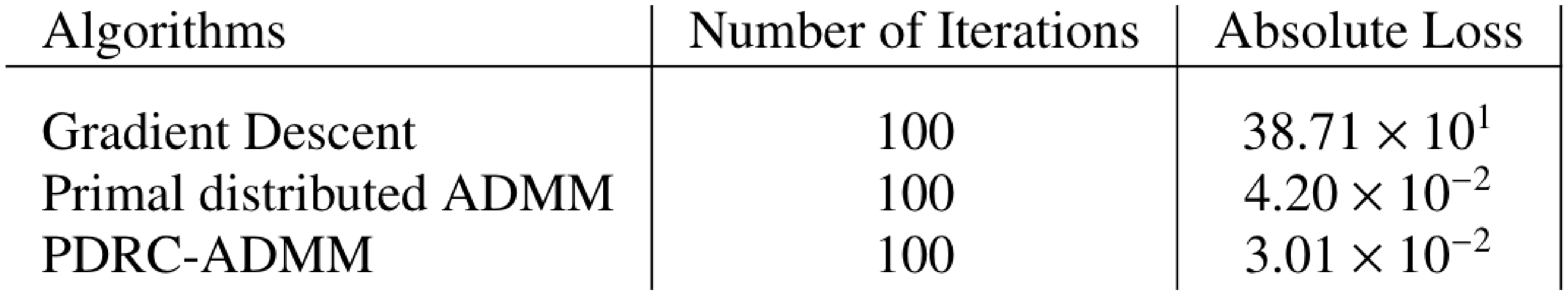
- With 5% of access to global data, DRC ADMM utilizes the benefit of data exchange, and outperforms primal distributed ADMM.

- Benefit of DRC-ADMM
  - Manage to get a good quality of solution **within fewer iteration**, which further reduces the communication load across centers
  - Manage to get a good quality of solution **within a fixed time**.

Privacy-Preserved DRC-ADMM

- Privacy would be a major concern when performing data exchange. The privacy-preserved DRC-ADMM (PDRC-ADMM) utilizes the random **Gaussian projection** for data required to be exchanged, which is known to be differentially private.

- Let $\boldsymbol{R} \in R^{k \times k}$ be a square matrix with entries i.i.d. sampled from normal Gaussian, and $k = \alpha n$, where α is the percentage of global data.



Algorithms	Number of Iterations	Absolute Loss
Gradient Descent	100	38.71×10^{1}
Primal distributed ADMM	100	4.20×10^{-2}
PDRC-ADMM	100	3.01×10^{-2}

Today’s talk

- Introduction to Multiblock-ADMM
- Benefit of Data Exchange in Multi-Block ADMM Algorithm for Regression Estimation
- Applying RAC-ADMM to Mixed Integer Programming
    - Kresimir Mihic, School of Mathematics, The University of Edinburgh
    - Yinyu Ye, Management Science & Engineering, Stanford University

Approaches to MIP

Using RACQP

$$\begin{array}{ll} \min\limits_{\mathbf{x}} & \frac{1}{2}\mathbf{x}^T\,\mathbf{H}\,\mathbf{x} + \mathbf{c}^T\,\mathbf{x} \\ \text{s.t.} & \mathbf{A}\,\mathbf{x} = \mathbf{b} \\ & \mathbf{l} \le \mathbf{x} \le \mathbf{u} \quad x_i \in \mathbb{Z}, x_j \in \mathbb{R},\ i = 1,\ldots,d,\ j = d+1,\ldots,n \end{array} \qquad (1)$$

(I) RACQP using external MIP solver for sub-problems

- RACQP is the upper-level engine, sub-problems solved to integrality using an (external) MIP solver
- Good solutions for large MIQP found fast
- Can not guarantee the optimal solution; feasibility often met, but can not be guaranteed

(II) Branch-and-bound solver based on RACQP (work in progress)

- RACQP is “internal”, low-level engine solving relaxed problems
- Optimal solution (within mipGap) or certificate of infeasibility returned

RACQP using external MIP solver for sub-problems

Solves the problem (1) using two-level approach. Lower-level finds a locally optimal solution given some initial point $\mathbf{x}^0$:

- Integrality constraints enforced by a sub-problem solver,

$$\mathbf{x}_i^{k+1} = \arg\min\{L_\beta(\mathbf{x}_1^{k+1}, \ldots, \mathbf{x}_i, \ldots, \mathbf{x}_p^k; \mathbf{y}^k) |\ \mathbf{l} \le \mathbf{x}_i \le \mathbf{u},\ \mathbf{x}_i \in \mathcal{X}_i\},$$

  where $(\mathcal{X}_i)_j = \mathbb{Z}$ or $\mathbb{R}$ for integer and continuous variables respectively.

Upper-level "escapes" from the local optimum by producing $\mathbf{x}^0_{\text{new}}$ in a neighborhood of the current best solution, $\mathbf{x}_{\text{best}}$:

- $\mathbf{x}^0_{\text{new}}$ found by perturbing values of randomly chosen components of $\mathbf{x}_{\text{best}}$
- Number of variables to change: chosen from a truncated exponential distribution (experimentally found to produce good results).
- Swap, exchange, permute,...

Improving the performance: grouping variables

Smart-grouping is a pre-processing method which uses a block structure of matrix $\mathbf{A}$ to pre-group certain variables as a single âsuper-variableâ (a group of variables which are always solved together).

$$\underbrace{\begin{bmatrix} \mathbf{V}_1 & 0 & \cdots & 0 \\ 0 & \vdots & & \vdots \\ \vdots & & \mathbf{V}_v & 0 \\ \mathbf{W}_1 & \cdots & \mathbf{W}_v & \mathbf{W}_{v+1} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_v \\ \mathbf{x}_{v+1} \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_v \\ \mathbf{b}_{v+1} \end{bmatrix}}_{\mathbf{b}} .$$

- One super-variable $\bar{\mathbf{x}}_i$ is made for each group $\mathbf{x}_i$, $i = 1, \dots, v$.
- Primal variables $\mathbf{x}_{v+1}$ stay shared and are randomly assigned to sub-problems to complement super-variables to which they are coupled with via block-matrices $\mathbf{W}_i$, $i = 1, \dots, v$.
- More than one super-variable can be assigned to a single sub-problem, dependent upon the maximum size of a sub-problem, if defined.

Improving the performance: partial augmented Lagrangian approach

Splitting the matrix $\mathbf{A}$ such that the block $\mathbf{W} = [\mathbf{W}_1, \ldots, \mathbf{W}_{v+1}]$ is admitted by the augmented Lagrangian while the rest of the constraints (blocks $\mathbf{V}_i$) are solved exactly as a part of a sub-problem,

$$\mathbf{x}_i^{k+1} = \arg\min\{L_{\mathcal{P}}(\mathbf{x}_1^{k+1}, \ldots, \mathbf{x}_i, \ldots, \mathbf{x}_p^k; \mathbf{y}^k) | \mathbf{V}_j \bar{\mathbf{b}}_j = \mathbf{b}_j, j \in \mathcal{J}, \mathbf{l} \leq \mathbf{x}_i \leq \mathbf{u}, \mathbf{x}_i \in \mathcal{X}_i\},$$

where $\mathcal{J}$ is a set of indices of super-variables $\bar{\mathbf{b}}_j$ constituting sub-problem i at any given iteration.

The partial augmented Lagrangian is defined with

$$L_{\mathcal{P}}(\mathbf{x}, \mathbf{y}) = \frac{1}{2} \mathbf{x}^T \mathbf{H} \mathbf{x} + \mathbf{c}^T \mathbf{x} - \mathbf{y}^T (\mathbf{W} \mathbf{x} - \mathbf{b}_{v+1}) + \frac{\beta}{2} || \mathbf{W} \mathbf{x} - \mathbf{b}_{v+1} ||^2.$$

The approach found to be useful for both continuous and mixed integer QP

For MIQP local constraints are sets of rules that relate integer variables, while constraints between continuous variables are left global. In the case of a problems where such straight separation does not exist, or when problems are purely integer, a problem structure is let to guide the local/global constraints decision.

Experiments: maximum bisection problem

A variant of the Max-Cut problem that involves partitioning the vertex set V of a graph $G = (V, E)$ into two disjoint sets V_1 and V_2 of equal cardinality (i.e. $V_1 \cap V_2 = \emptyset$, $V_1 \cup V_2 = V$, $|V_1| = |V_2|$) such that the total weight of the edges whose endpoints belong to different subsets is maximized.

A standard formulation can be re-formulated into mixed binary quadratic problem:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{x}^T \mathbf{H} \mathbf{x} \\ \text{s.t.} \quad & \mathbf{e}^T \mathbf{x} = \lfloor n/2 \rfloor \\ & \mathbf{x} \in \{0, 1\}^n. \end{aligned}$$

At each iteration we update the i^{th} block by solving

$$\begin{aligned} \min_{\mathbf{x}_i} \quad & \mathbf{x}_i^T \mathbf{H}_i \mathbf{x}_i - yr + \frac{\beta}{2} r^2 \\ \text{s.t.} \quad & \mathbf{e}^T \mathbf{x}_i - r = \mathbf{b}_i \\ & \mathbf{x}_i \in \{0, 1\}^{|\mathbf{x}_i|}, \ r \in [0, 1]. \end{aligned}$$

where $b_i = \lfloor n/2 \rfloor - \mathbf{e}^T \mathbf{x}_{-i}$ with $\mathbf{x}_{-i}$ being the sub-vector of $\mathbf{x}$ with indices of variables not in block i.

Experiments: maximum bisection problem

Gurobi used for solving RACQP sub-problems (called via Matlab interface).

The penalty parameter $\beta = 0.005$, number of blocks $p = 4$, Perturbation mechanism: swapping.

Instance name	Problem size	Density $(\mathbf{H})$	Best known Obj. val.	Gap[1]							
				run time = 5 min		run time = 10 min		run time = 30 min		run time = 60 min	
				Gurobi	RACQP	Gurobi	RACQP	Gurobi	RACQP	Gurobi	RACQP
G63	7000	$2{\cdot}10^{-3}$	26988	-0.263	-0.007	-0.160	-0.006	-0.160	-0.005	-0.037	-0.003
G67	10000	$5{\cdot}10^{-4}$	6938	-0.272	-0.016	-0.168	-0.014	-0.004	-0.011	-0.001	-0.010
G70	10000	$3{\cdot}10^{-4}$	9581	-0.009	-0.008	-0.007	-0.007	-0.006	-0.005	-0.003	-0.004
G77	14000	$3{\cdot}10^{-4}$	9918	-0.468	-0.015	-0.468	-0.013	-0.247	-0.012	-0.095	-0.010
G81	20000	$2{\cdot}10^{-4}$	14030	-0.280	-0.017	-0.280	-0.015	-0.253	-0.014	-0.214	-0.012
Average (across all G1-G81 instances):				-0.1228	-0.0101	-0.1046	-0.0088	-0.0735	-0.0073	-0.0513	-0.0065

Table 1: Max-Bisection, GSET instances [4]. Gap between best known results and RACQP/Gurobi objective values [6].

[1] $gap = (f(\mathbf{x}_S^*) - f(\mathbf{x}^*))/(1 + |f(\mathbf{x}^*)|)$ where $f(\mathbf{x}^*)$ and $f(\mathbf{x}_S^*)$ are objective values of the best known solution and of a solver, respectively

Experiments: quadratic assignment problem (QAP)

The objective of QAP is to assign n facilities to n locations in such a way that the assignment cost is minimized. The assignment cost is the sum, over all pairs, of a weight or flow between a pair of facilities multiplied by the distance between their assigned locations.

A standard formulation can be re-formulated into a binary quadratic problem:

$$\begin{array}{lll} \min\limits_{\mathbf{X}} & \mathrm{vec}(\mathbf{X})^T\,\mathbf{H}\,\mathrm{vec}(\mathbf{X}) & \\ \text{s.t.} & \sum_{i=1}^{r} x_{ij} = 1,\ \forall j = 1,\ldots r & \text{(a)} \\ & \sum_{j=1}^{r} x_{ij} = 1,\ \forall i = 1,\ldots r & \text{(b)} \\ & 0 \leq x_{ij},\ \forall i,j = 1,\ldots r & \end{array} \tag{2}$$

where x_{ij} is the entry of the permutation matrix $\mathbf{X} \in \mathbb{R}^{r\times r}$, and $\mathbf{H} = (\mathbf{A} \otimes \mathbf{B})$ with $\mathbf{A} \in \mathbb{R}^{r\times r}$ and $\mathbf{B} \in \mathbb{R}^{r\times r}$ being the “flow” and “distance” matrices respectively.

Experiments: quadratic assignment problem (QAP)

Variables grouped following the structure of constraints, which is dictated by the permutation matrix $\mathbf{X} \in \{0, 1\}^{r \times r}$: one super-variable, $\mathbf{x}_i$ for each row i of $\mathbf{X}$.

Sub-problem is defined with

$$\mathbf{x}_i^{k+1} = \arg\min\{L_{\mathcal{P}}(\cdot)|\ \mathbf{A}_{local}\, \mathbf{x}_i = \mathbf{1},\ \mathbf{x}_i \in \{0, 1\}^n\}.$$

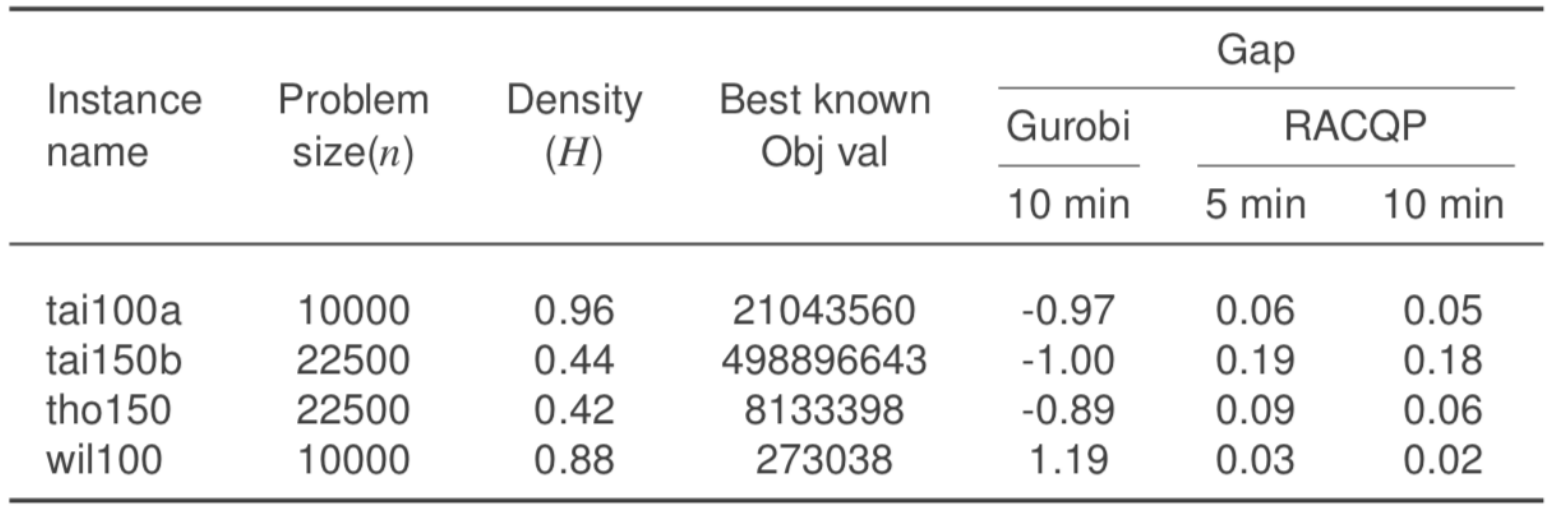
where $\mathbf{A}_{local}$ contains constraints described by (2 a), and the the partial Lagrangian is

$$L_{\mathcal{P}}(\mathbf{x}, y) = \frac{1}{2}\, \mathbf{x}^T\, \mathbf{H}\, \mathbf{x} - \mathbf{y}^T (\mathbf{A}_{global}\, \mathbf{x} - \mathbf{1}) + \frac{\beta}{2} \|\, \mathbf{A}_{global}\, \mathbf{x} - \mathbf{1}\, \|^2.$$

where $\mathbf{A}_{global}$ consists of constraints coming form (2 b).

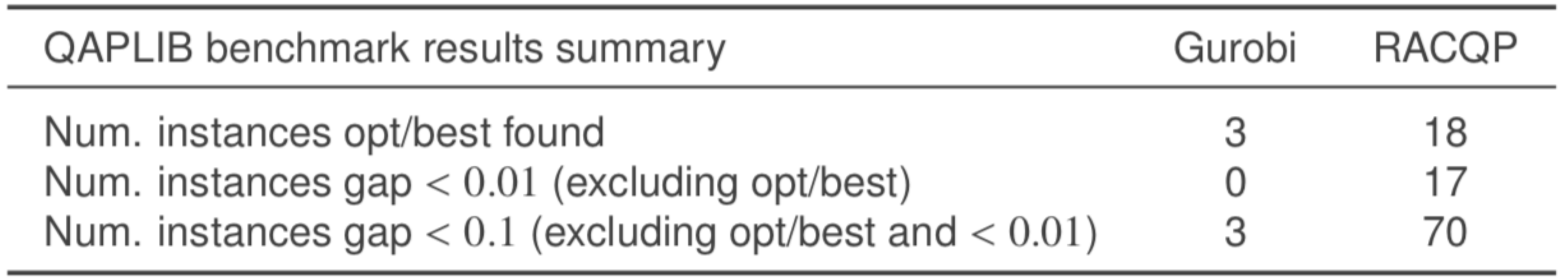
The initial point is a random feasible vector. The penalty parameter is a function of the problem size, $\beta = n,\ n = r^2$, while the number of sub-problems depends on the permutation matrix size, $p = \lceil r/2 \rceil$. Perturbation mechanism: swapping (of the super-variables). Gurobi used for solving sub-problems.

Experiments: quadratic assignment problem (QAP)



Instance name	Problem size(n)	Density (H)	Best known Obj val	Gap		
				Gurobi	RACQP	
				10 min	5 min	10 min
tai100a	10000	0.96	21043560	-0.97	0.06	0.05
tai150b	22500	0.44	498896643	-1.00	0.19	0.18
tho150	22500	0.42	8133398	-0.89	0.09	0.06
wil100	10000	0.88	273038	1.19	0.03	0.02

Table 2: QAPLIB [2], selected large problems. [6].



QAPLIB benchmark results summary	Gurobi	RACQP
Num. instances opt/best found	3	18
Num. instances gap < 0.01 (excluding opt/best)	0	17
Num. instances gap < 0.1 (excluding opt/best and < 0.01)	3	70

Table 3: Number of instances = 133. Max run-time: 10 min. [6]

Ongoing: Branch-and-bound solver based on RACQP

Extending RACQP by embedding it within a branch-and-bound method:

- Search strategy: depth first until first incumbent, then best bound
- Branching method: the variable closest to an integer
- Warm-start: initializing RACQP with primal and dual solutions from the parent nodes
- Pre-factorization for RD/ADMM modes: using the same factorization together with the parent node solution.

Current implementation is a proof a concept:

- Implementation done for mixed binary problems only.
- Basic branching algorithm used.

Next in pipeline: Branch-and-cut algorithm implementation. Much later in pipeline: port RACQP to C/C++.

Solving a node: convex relaxation

Relaxing integrality constraints of (1) to

$$\begin{array}{ll} \min\limits_{\mathbf{x}} & \frac{1}{2}\mathbf{x}^T\mathbf{H}\,\mathbf{x} + \mathbf{c}^T\mathbf{x} \\ \text{s.t.} & \mathbf{A}\,\mathbf{x} = \mathbf{b} \\ & \mathbf{x} - \bar{\mathbf{x}} = \mathbf{0} \\ & \mathbf{l} \le \bar{\mathbf{x}} \le \mathbf{u} \\ & \mathbf{x}, \bar{\mathbf{x}} \in \mathbb{R}^n \end{array} \tag{3}$$

The augmented Lagrangian of (3) is then

$$\begin{aligned} L_\beta(\mathbf{w};\mathbf{y};\mathbf{z}) := \quad & \frac{1}{2}\mathbf{x}^T\mathbf{H}\,\mathbf{x} + \mathbf{c}^T\mathbf{x} - \mathbf{y}^T(\mathbf{A}\,\mathbf{x} - \mathbf{b}) - \mathbf{z}^T(\mathbf{x} - \bar{\mathbf{x}}) \\ & + \frac{\beta}{2}(\|\,\mathbf{A}\,\mathbf{x} - \mathbf{b}\,\|^2 + \|\,\mathbf{x} - \bar{\mathbf{x}}\,\|^2) \end{aligned}$$

and the optimal $\bar{\mathbf{x}}$ has a closed form solution given by

$$\bar{\mathbf{x}} = \min(\max(\mathbf{l}, \mathbf{x} - \frac{1}{\beta}\mathbf{z}), \mathbf{u})$$

Solving a node: Certificates of primal and dual infeasibility

For some arbitrary $\mathbf{A}_i \in \mathbb{R}^{m_i \times n}$, Motzkin's theorem of the alternative states that exactly one of the following sets is nonempty

$$\begin{aligned} I) &\quad \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}_1\,\mathbf{x} > \mathbf{0},\ \mathbf{A}_2\,\mathbf{x} \geq \mathbf{0},\ \mathbf{A}_3\,\mathbf{x} = \mathbf{0}\} \\ II) &\quad \{\mathbf{y}_1, \mathbf{y}_2 \geq \mathbf{0}, \mathbf{y}_3 \in \mathbb{R}^{m_3} : \mathbf{A}_1^T\,\mathbf{y}_1 + \mathbf{A}_2^T\,\mathbf{y}_2 + \mathbf{A}_3^T\,\mathbf{y}_3 = \mathbf{0}, \mathbf{e}^T\,\mathbf{y}_1 > 0\} \end{aligned} \tag{4}$$

Substituting constraints of (3) into (4), and taking into account that $\bar{\mathbf{x}}$ is found by a projection, gives

$$\begin{aligned} \mathcal{P} = &\quad \{\mathbf{x} \in \mathbb{R}^n, \bar{\mathbf{x}} \in [\mathbf{l}, \mathbf{u}]^n : \mathbf{A}\,\mathbf{x} = \mathbf{b},\ \mathbf{x} - \bar{\mathbf{x}} = \mathbf{0}\} \\ \mathcal{D} = &\quad \{\mathbf{y} \in \mathbb{R}^m, \mathbf{z} \in \mathbb{R}^n : \mathbf{A}^T\,\mathbf{y} + \mathbf{z} = \mathbf{0}, \mathbf{b}^T\,\mathbf{y} > 0\},\ \text{or} \\ &\quad \{\mathbf{y} \in \mathbb{R}^m, \mathbf{z} \in \mathbb{R}^n : \mathbf{A}^T\,\mathbf{y} + \mathbf{z} = \mathbf{0}\}, \text{when}\ \ \mathbf{b} = \mathbf{0} \end{aligned}$$

meaning that any variable $\mathbf{y} \in \mathcal{D}$ serves as a certificate that problem (3) is primal infeasible.

Solving a node: Finding an integer feasible solution

Applying rounding heuristics to find a feasible solution after solving the relaxation at each node:

- Nearest-neighbor rounding
- Randomized rounding, $\text{Prob}(\text{round}(x_i) = 1) = x_i$
- Dependent randomized rounding:
  - Geometric rounding : $\sum_{i=1}^{N} x_i = 1,\ \mathbf{x} \in \{0,1\}^N$
  - Pipage random rounding [2]:

$$\sum_{i=1}^{N} y_{i,r} \le 1, \forall r \ \sum_{j=1}^{J} x_{j,t} \ge 1, \forall t \ \sum_{i=1}^{P_{r,t}} y_{i,r} - x_{j,t}, \forall r,t;\ \mathbf{y} \in \{0,1\}^N,\ \mathbf{x} \in \{0,1\}^M$$

  - additional solutions that utilize constraint structure will be implemented.

Followed by the optimization of the remaining continuous variables by running RACQP for a couple of iterations (root relaxation usually ran with more iterations that the rest).

Experiments: Randomly constructed problems

Created random MBQP with varying dimensions $n,\ m$ and number of binary variables r: Density of $\mathbf{H}$ is 0.1, with entries generated from the uniform distribution $U(0,1)$, the linear cost $\mathbf{c}$ from the normal distribution $N(0,1)$, and the constraints $\mathbf{A} \sim N(0,1)$.

For each set of parameters 5 random problems were constructed.

n	r	m	max. runtime [s]
500	50, 100	100, 200, 300	600
1000	50, 100	300, 500, 800	600
2000	50, 100	500, 1000	600
5000	50, 100	500, 1000	1800

Table 4: MBQP parameters

Presolve option was turned off for Gurobi. Runs limited to one thread.

Experiments: Randomly constructed problems

n	RACQP				Gurobi		
	Root LB time [s]	Root round time [s]	# incumbent after round	Cost per node [s]	Root LB time [s]	Time to first UB [s]	Cost per node [s]
500	0.14	0.88	25(30)	0.05	0.25	9.90	0.06
1000	2.17	3.95	23(30)	0.13	5.94	10.36	1.83
2000	9.76	6.94	20(20)	0.28	215.16	225.99	2.77
5000	48.52	7.48	13(20)	0.91	192.12	212.27	2.95

Table 1: Summary of the results. Average time given.

- Rounding scheme used by RACQP finds good incumbents very fast.
- Solutions found by RACQP match those of Gurobi, but satisfying mipGap requirement takes time. Gurobi produces good cuts that eliminate much of the tree.
- Cost per node is much lower for RACQP, making room for time for non-linear cut generation.
- Current RACQP implementation is Matlab, which greatly limits its performance. Migrating the implementation to C/C++.

References

[1] G. Banjac, P. Goulart, B. Stellato, and S. Boyd, *Infeasibility detection in the alternating direction method of multipliers for convex optimization*, Journal of Optimization Theory and Applications, 183 (2019), pp. 490–519.

[2] R. Gandhi, S. Khuller, S. Parthasarathy, and A. Srinivasan, *Dependent rounding and its applications to approximation algorithms*, Journal of the ACM (JACM), 53 (2006), pp. 324–360.

[3] Y. Liu, E. K. Ryu, and W. Yin, *A new use of douglas-rachford splitting and admm for identifying infeasible, unbounded, and pathological conic programs*, arXiv preprint arXiv:1706.02374, (2017).

[4] K. Mihić, M. Zhu, and Y. Ye, *Managing randomization in the multi-block alternating direction method of multipliers for quadratic optimization*, Mathematical Programming Computation, (2020), pp. 1–75.

[5] Y. Liu, E. K. Ryu, and W. Yin, *A new use of douglas-rachford splitting and admm for identifying infeasible, unbounded, and pathological conic programs*, arXiv preprint arXiv:1706.02374, (2017).

[6] K. Mihić, M. Zhu, and Y. Ye, *Managing randomization in the multi-block alternating direction method of multipliers for quadratic optimization*, Mathematical Programming Computation, (2020), pp. 1–75.

Thank you!

COMMENTS AND QUESTIONS?

KMIHIC@ALUMNI.STANFORD.EDU
MINGXIZ@STANFORD.EDU
YYYE@STANFORD.EDU