Parallel and Distributed Sparse Optimization Algorithms

Part I

Ruoyu Li[1]

[1]Department of Computer Science and Engineering
University of Texas at Arlington

March 19, 2015

Outline

Outline

Paper We Present

The paper we present today:
Title: "*Parallel and distributed sparse optimization*."
Authors: Peng, Zhimin, Ming Yan, and Wotao Yin.
Publisher: Signals, Systems and Computers, 2013 Asilomar Conference on. IEEE, 2013.
Affiliation: Dept. Mathematics, UCLA.

Motivations

1. Data is big right now. Too big to process them in single workstation.
2. Distributed data. Due to the fact that data is usually collected and stored separately, it is natural to try to process them separately and locally
3. Current algorithms are not scalable enough and cannot well reduce the total processing time complexity.

Outline

Related Works

Single Thread Algorithm for Sparse Optimization Algorithms

- Subgradient descent. LP, SOCP[1], SDP
- Smoothing. approximate l_1 norm by
  - $l_{1+\epsilon}$ norm
  - $\sum_i \sqrt{x_i^2 + \epsilon}$
  - Huber-norm[2]

  and apply gradient descent methods or quasi-Newton methods
- Splitting: split smooth and non-smooth terms into several simple subproblems
  - operator splitting: GPSR/FPC/SpaRSA/FISTA/SPGL1/....
  - dual operator splitting: Bregman/ADMM/split Bregman[3]/....
- Greedy algorithms: OMP[4], CoSaMP.....

[1] Second-Order-Cone Programming

[2] http://www.seas.ucla.edu/ vandenbe/236C/lectures/smoothing.pdf

[3] Setzer, Simon, Gabriele Steidl, and Tanja Teuber. "Deblurring Poissonian images by split Bregman techniques." Journal of Visual Communication and Image Representation 21.3 (2010): 193-199.

[4] Orthogonal Matching Pursuit

Related Works

State-of-the-art Parallel Sparse Optimization Algorithms

- Distributed ADMM. The total time complexity is not actually reduced, because its iteration number increases with the number of distributed blocks. Ref[9-10]
- Parallel Coordinate Descent. Randomly, cyclically or greedily select part of blocks to update for each iteration. Ref[11-13]

The proposed algorithm has obvious advantage over distributed ADMM and have merit on efficiency over the distributed coordinate descent under some assumption of coordinate orthogonality conditions.

Contributions

In this paper, the authors proposed two parallel algorithms.

- One is motivated by the separable objective functions

$$f(x) = \sum_{s=1}^{S} f_s(\mathbf{x}_s), \quad or \tag{1}$$

$$f(x) = \sum_{s=1}^{S} f_s(\mathbf{x}) \quad (partially \quad separable),$$

  and it can parallel existing proximal-linear algorithms, e.g. ISTA, FISTA, FPC.

- One is based on the data orthogonality. Both block and variables of each block are selected by greedy means. We argue that greedy coordinate block selection can be also fast.

Outline

Problem Definition

In this talk, we try to solve the sparse optimization problem underlying some structure in the solution. The objective function:

$$min_{\mathbf{x}\in\mathbb{R}^n}\mathcal{F}(\mathbf{x}) = \lambda\mathcal{R}(\mathbf{x}) + \mathcal{L}(\mathbf{Ax}, \mathbf{b})). \tag{2}$$

where $\mathcal{R}(\mathbf{x})$ is a non-smooth regularizer. $\mathcal{L}(\mathbf{Ax}, \mathbf{b}))$ is the data fidelity or loss function, which is usually smooth.

Notice

When solving the problem, no matter what algorithm we utilize, many $\mathbf{Ax}$ and $\mathbf{A}^T\mathbf{y}$ calculation are necessary. Accelerating the messy calculation of these matrix operations is a big plus for algorithm speed.

The Separability of Objective Function

1. For many regularization term $\mathcal{R}(\mathbf{x})$, e.g. l_1, $l_{1,2}$, Huber function, and elastic net function. We can write $\mathcal{R}(\mathbf{x}) = \sum_{b=1}^{B} \mathcal{R}(\mathbf{x}_b)$
2. Loss functions $\mathcal{L}(\mathbf{Ax}, \mathbf{b})$, e.g. square, logistic and hinge loss function, are also feasible for separating: $\mathcal{L}(\mathbf{Ax}, \mathbf{b}) = \sum_{b=1}^{B} \mathcal{L}(\mathbf{A}_b\mathbf{x}, \mathbf{b}_b)$
3. Usually, we only require one of them to be separable.

Data Distribution Scenarios

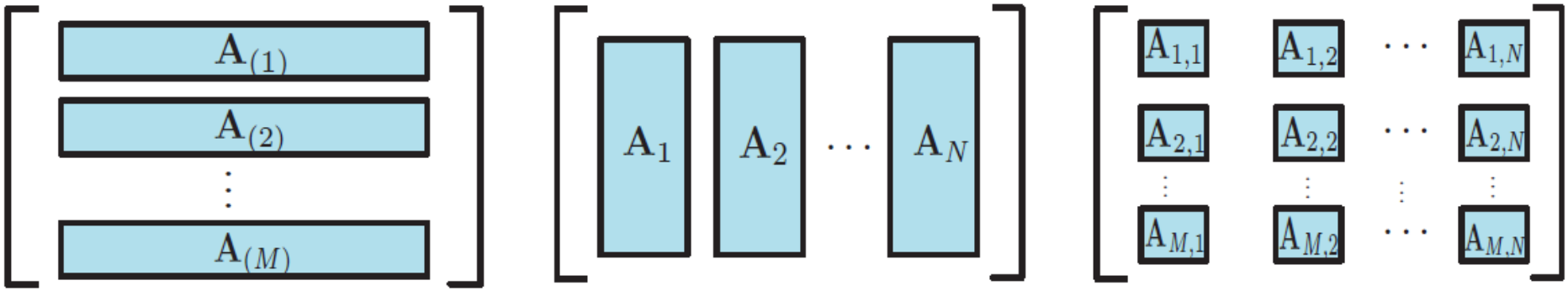

Figure: Data Distribution Scenarios for matrix A. a.Row blocks, b.column blocks and c.general blocks.

Benefits:

1. When A is very large, small store space for local computer.
2. Sometimes A is collected or generated separately, and it is easier to process and update locally.

Examples

In scenario a, computing Ax and $A^T y$:

$$Ax = \begin{bmatrix} A_{(1)}x \\ A_{(2)}x \\ \cdots \\ A_{(M)}x \end{bmatrix} \tag{3}$$

and $A^T y = \sum_{i=1}^{M} A_{(i)}^T y_i$, where y_i is ith block of y.

- For computing Ax, it is all right to independently store and process each block of A locally, but we need to broadcast x to each nodes, and update it every iteration.
- For computing $A^T y$, we need the *reduce* method to sum the solution from each node.

Examples-continued

In scenario b, computing Ax and $A^T y$:

$$A^T y = \begin{bmatrix} A_{(1)}^T y \\ A_{(2)}^T y \\ \cdots \\ A_{(M)}^T y \end{bmatrix} \tag{4}$$

and $Ax = \sum_{i=1}^{N} A_{(i)} x_i$, where x_i is ith block of x.

- For computing $A^T y$, we need to broadcast y to each nodes, and update it every iteration.
- For computing Ax, we need the *reduce* method to sum the solution from each node.

Examples-continued

In scenario c, computing either $A^T y$ or Ax requires a mixed use of broadcasting and reduce operations. Use Ax for example:

$$Ax = \begin{bmatrix} A_{(1)}x \\ A_{(2)}x \\ \cdots \\ A_{(M)}x \end{bmatrix} = \sum_{j=1}^{N} \begin{bmatrix} A_{1,j}x_j \\ A_{2,j}x_j \\ \cdots \\ A_{M,j}x_j \end{bmatrix} \tag{5}$$

1. broadcast x_j to nodes $(1,j),(2,j),\cdots,(M,j)$.
2. in parallel, calculate all $A_{i,j}x_j$ for i = 1,$\cdots$,M and j = 1,$\cdots$,N
3. reduce method is applied to nodes $(i,1),(i,2),\cdots,(i,N)$ and return $A_{(i)}x = \sum_{j=1}^{N} A_{i,j}x_j$.

Outline

Distributed Algorithm of Proximal Linear Algorithm

The proximal -linear algorithms, e.g. ISTA, FISTA, FPC, are iterations of gradient descent and proximal operator. When finding the gradient:

$$\nabla_x \mathcal{L}(\mathbf{A}\mathbf{x}^k, \mathbf{b}) = \mathbf{A}^T \nabla \mathcal{L}(\mathbf{A}\mathbf{x}^k, \mathbf{b}) \tag{6}$$

, we can see that the it could be accelerated by distributed computing of Ax and $A^T y$.

When we move to solving the proximal operator, we could utilize the separability of regularization $\mathcal{R}(\mathbf{x})$.And its solution is easy to obtain, e.g. $|\mathbf{x}|_1$ with close-form soft-thresholding solution.

Distributed Algorithm of Proximal Linear Algorithm

Using Taylor expansion, we could have the quadratic approximation of $\mathcal{L}(Ax, b)$ and apply Minorize-Maximization (MM) method, we have the following new problem for each iteration:

$$x^{k+1} \leftarrow arg\min_x \lambda\mathcal{R}(x) + < x, A^T\nabla\mathcal{L}(Ax^k, b) > + \frac{1}{2\delta_k}\|x - x^k\|_2^2 \quad (7)$$

and it could be given as proximal operator:

$$x^{k+1} = \mathbf{prox}_{\lambda\mathcal{R}}(x^k - \delta_k A^T\nabla\mathcal{L}(Ax^k, b)), \quad (8)$$

where $\mathbf{prox}_{\lambda\mathcal{R}}(t) = arg\min_x \lambda\mathcal{R}(x) + \frac{1}{2}\|x - t\|_2^2$.

Distributed Algorithm of Proximal Linear Algorithm

In scenario a,

1. Every node i keeps $\mathbf{A}_{(i)}$, $\mathbf{b}_i$ and current $\mathbf{x}^k$.
2. For every node i, we first compute $\mathbf{A}_j x_j^k$ and then compute $\nabla \mathcal{L}_i(\mathbf{A}_{(i)}\mathbf{x}, \mathbf{b}_i)$ in parallel manner. After that, we compute the $\mathbf{A}^T \nabla \mathcal{L}(\mathbf{A}\mathbf{x}, \mathbf{b}) = \sum_{i=1}^{M} \mathbf{A}_{(i)}^T \nabla \mathcal{L}_i(\mathbf{A}_{(\mathbf{i})}\mathbf{x}, \mathbf{b}_i)$ by reduce operation.
3. After that, we will easily solve the Eq(8)

To distribute the calculation, we require the separability of loss function $\mathcal{L}(Ax, b)$.

Two Examples

- Distributed LASSO.

$$min_x \lambda |x|_1 + \frac{1}{2}\|Ax - b\|_2^2 \tag{9}$$

  In this case, both $\mathcal{R}$ and $\mathcal{L}$ are separable.

- Distributed sparse logistic regression. Sparse logistic solver:

$$min_{w,c} \lambda \|w\|_1 + \frac{1}{m}\sum_{i=1}^{m} log(1 + exp(-b_i(w^T a_i + c))) \tag{10}$$

Parallel FISTA

Algorithm 1 P-FISTA: **scenario-b** distributed LASSO

1: node j keeps $\mathbf{A}_j$, $\mathbf{b}$, initializes $\mathbf{x}_j^0 = \mathbf{x}_j^1 = 0$;
2: **for** $k = 1, 2, \ldots, K$ **do**
3: $\quad \bar{\mathbf{x}}_j \leftarrow \mathbf{x}_j^k + \frac{k-2}{k+1}\left(\mathbf{x}_j^k - \mathbf{x}_j^{k-1}\right)$;
4: $\quad \mathbf{w} \leftarrow \sum_{j=1}^{N} \mathbf{A}_j \bar{\mathbf{x}}_j$ by MPI_Allreduce;
5: $\quad \mathbf{y} \leftarrow \nabla \mathcal{L}(\mathbf{w}; \mathbf{b})$;
6: $\quad \mathbf{g}_j \leftarrow \mathbf{A}_j^T \mathbf{y}$;
7: $\quad \mathbf{x}_j^{k+1} \leftarrow \mathbf{prox}_{\lambda\|\cdot\|_1}(\bar{\mathbf{x}}_j - \delta_k \mathbf{g}_j)$;
8: **end for**

Figure: Algorithm-1, p-FISTA for scenario b

Distributed Logistic Regression

Algorithm 2 diSLR: **scenario-a** distributed sparse logistic reg.

node i keeps $\mathbf{C}_{(i)}$, $\mathbf{b}_{(i)}$, sets $\mathbf{w}^0 = \mathbf{w}^1 = 0$, $c^0 = c^1 = 0$;
for $k = 1, 2, \ldots, K$ **do**
  $\bar{\mathbf{w}} \leftarrow \mathbf{w}^k + \frac{k-2}{k+1}\left(\mathbf{w}^k - \mathbf{w}^{k-1}\right)$;
  $\bar{c} \leftarrow c^k + \frac{k-2}{k+1}\left(c^k - c^{k-1}\right)$;
  $\mathbf{y}_{(i)} \leftarrow \nabla \mathcal{L}_i(\mathbf{C}_{(i)}\bar{\mathbf{w}} + \mathbf{b}_{(i)}\bar{c})$;
  $\mathbf{g}_{\mathbf{w}} \leftarrow \sum_{i=1}^{M} \mathbf{C}_{(i)}^T \mathbf{y}_{(i)}$ by MPI_Allreduce;
  $g_c \leftarrow \sum_{i=1}^{M} \mathbf{b}_{(i)}^T \mathbf{y}_{(i)}$ by MPI_Allreduce;
  $\mathbf{w}^{k+1} \leftarrow \mathbf{prox}_{\lambda\|\cdot\|_1}(\bar{\mathbf{w}} - \delta_k \mathbf{g}_{\mathbf{w}})$;
  $c^{k+1} \leftarrow \bar{c} - \delta_k g_c$;
end for

Figure: Algorithm-2, distributed logistic regression for scenario a

Outline

Parallel Greedy Coordinate Descent Method

Coordinate Descent (CD) algorithm– the objective function is optimized with respect to a chosen coordinate or a block of coordinates while the rest stay fixed.

How to decide the coordinates to update:

- Cyclic CD. Go through all coordinates iteratively.
- Random CD. Randomly select coordinates and analysis based on expectation of objective function.
- Greedy CD. Choose the coordinates with best merit value(decrease the objective function)
- Mixed CD.

Greedy CD is better for sparse problem

Under certain orthogonality conditions, such as RIP and incoherence conditions, certain greedy selection rules can guarantee to select the coordinates corresponding to non-zero value in the final solution.
The greedy selection of coordinates is below:

1. Recall Eq(7), and we only focus on one coordinate i, we define the potential of this coordinate as :

$$d_i = arg\min_d \lambda r(x_i + d) + g_i d + \frac{\beta}{2} d^2 \tag{11}$$

   where $\mathcal{R}(x) = \sum_{i=1}^{n} r(x_i)$, n is the number of coordinates.
2. Out of N blocks, the P blocks with coordinate of highest potential, are selected.
3. Update these best coordinates of each selected blocks.

GRock for scenario b

Algorithm 3 GRock(P) for scenario b

1: Initialize $\mathbf{x} = 0 \in \mathbb{R}^n$
2: **while** not converged **do**
3: $\quad \mathbf{d}_j \leftarrow (6)$ for each block j;
4: $\quad m_j, s_j \leftarrow (7)$ for each block j;
5: $\quad \mathcal{P} \leftarrow$ the indices of the P blocks with largest m_j;
6: $\quad x_{s_j} \leftarrow x_{s_j} + d_{s_j}$, for each $j \in \mathcal{P}$.
7: **end while**

Figure: Algorithm-3

Demonstration for GRock- divergence

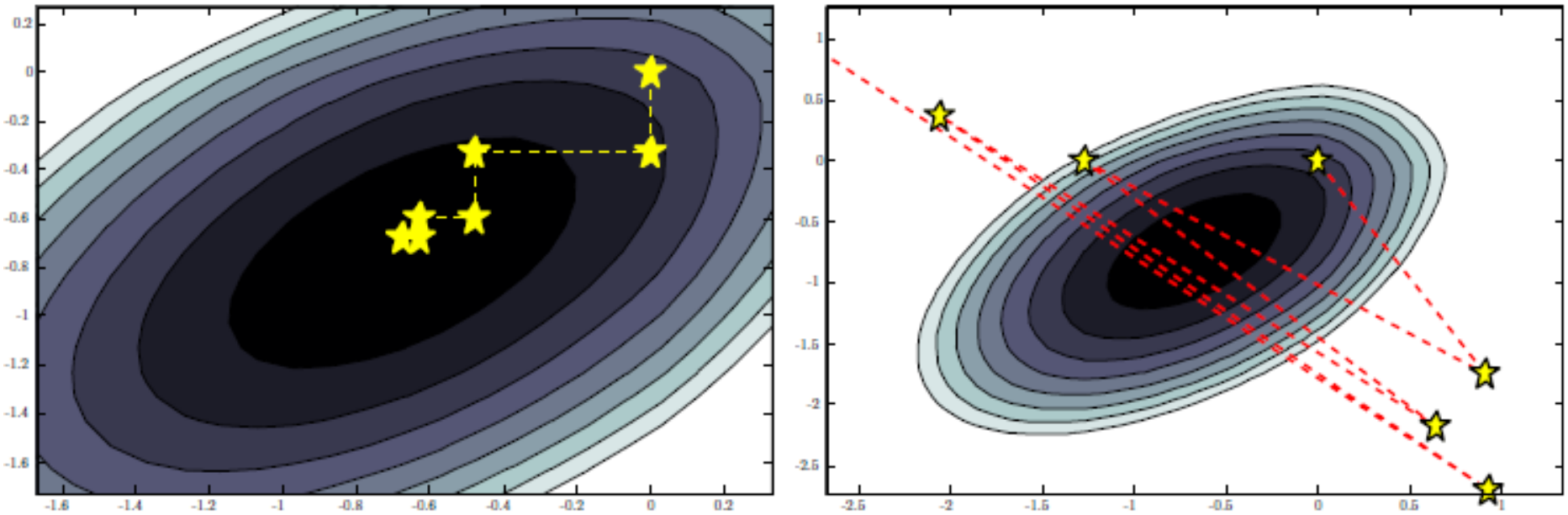

Figure: Contour is not aligned with the coordinate axis. Left: P = 1, optimize one coordinate each iteration; Right: P = 3.

Convergence

In this part, the paper mainly follow the derivations and theorems proposed in [5]. they define a block spectral radius :

$$\rho_P = \max_{M \in \mathcal{M}} \rho(M) \tag{12}$$

where $\mathcal{M}$ is the set of all $P \times P$ submatrices that we can obtain from $A^T A$ corresponding to selecting one column from each selected P blocks. $\rho(\star)$ is nothing but the maximal eigenvalue.

- The small $\rho(M)$ means the selected P columns from A (scenario b) are independent from each other.
- The higher P, the more likely to have a higher $\rho(M)$.

[5] Scherrer, Chad, et al. "Feature clustering for accelerating parallel coordinate descent." Advances in Neural Information Processing Systems. 2012.

Convergence

Lemma IV.1. *Assume $\mathcal{R}(\mathbf{x})$ is convex, $\mathcal{L}(\mathbf{Ax}, \mathbf{b})$ satisfies assumption (5), and $\mathbf{x}^k$ be the sequence generated by Alg. 3. If $\rho_P < 2$, then*

$$\mathcal{F}(\mathbf{x}^{k+1}) - \mathcal{F}(\mathbf{x}^k) \leq \frac{\rho_P - 2}{2}\beta\|\mathbf{x}^{k+1} - \mathbf{x}^k\|^2.$$

Theorem IV.2. *Let $\mathcal{R}(\mathbf{x}) = \|\mathbf{x}\|_1$, and $\mathbf{x}^*$ be a solution to (1), and $\mathbf{x}^k$ be the sequence generated by Alg. 3. Assume $\|\mathbf{x}^k - \mathbf{x}^*\|_2 \leq C$, $\mathcal{F}(\mathbf{x})$ satisfies the assumptions in Lemma IV.1, and the gradient of $\mathcal{L}(\mathbf{Ax}, \mathbf{b})$ satisfies the Lipschitz condition with Lipschitz constant L. Then, we have*

$$\mathcal{F}(\mathbf{x}^k) - \mathcal{F}(\mathbf{x}^*) \leq \frac{2\left(2CL + \beta C\sqrt{\frac{N}{P}} + 2\lambda\frac{N-P}{\sqrt{P}}\right)^2}{(2-\rho_P)\beta} \cdot \frac{1}{k}.$$

GRock for Sparse Optimization

- In CD algorithm usually cause more time to converge due to each iteration it only update few of coordinates.
- In sparse optimization problem, due to the nature that most entries of final solution is zero, we only need to focus on those non-zero coordinates.
- Greedy Rock algorithm always select coordinates corresponding to non-zero entries in final solution. It will need fewer iteration than other CD and proximal-linear algorithms.

GRock for Sparse Optimization

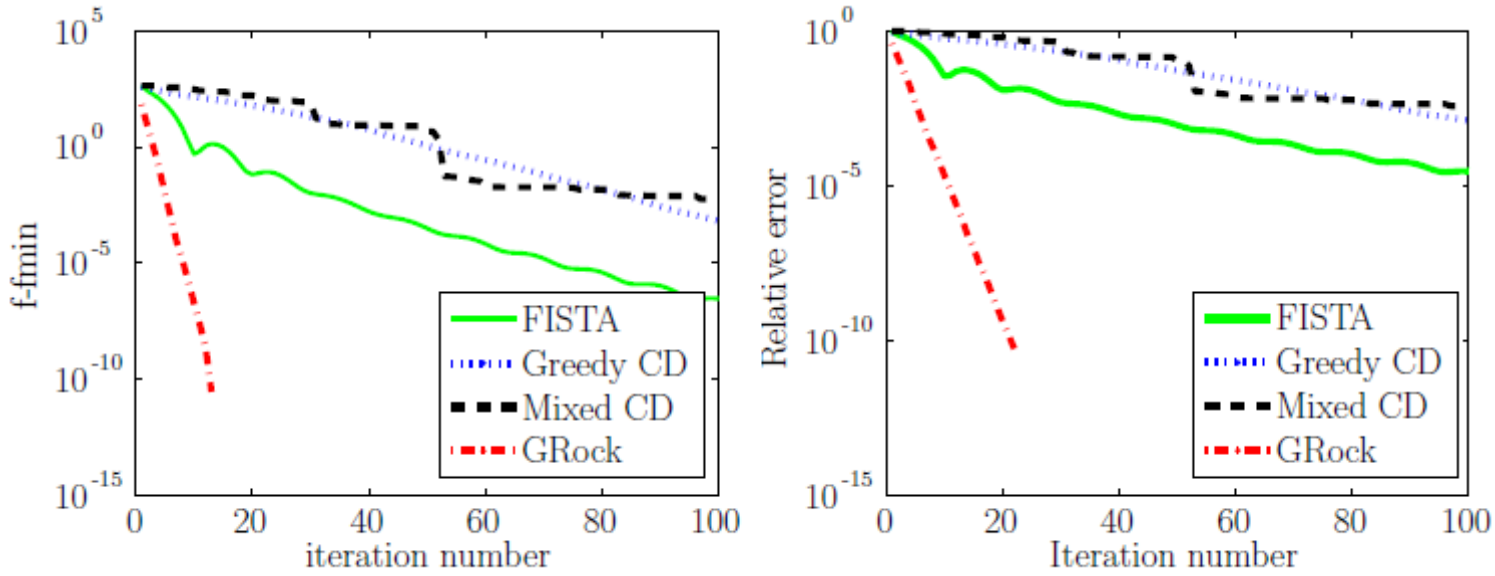

Figure: a comparison of different coordiante descent and FISTA on massive dataset with 0.4% non-zero entries.

Experiments on Cluster

Dataset I: 1024×2028; dataset II: 2048×4096.

Problem to solve: LASSO, Eq(9).

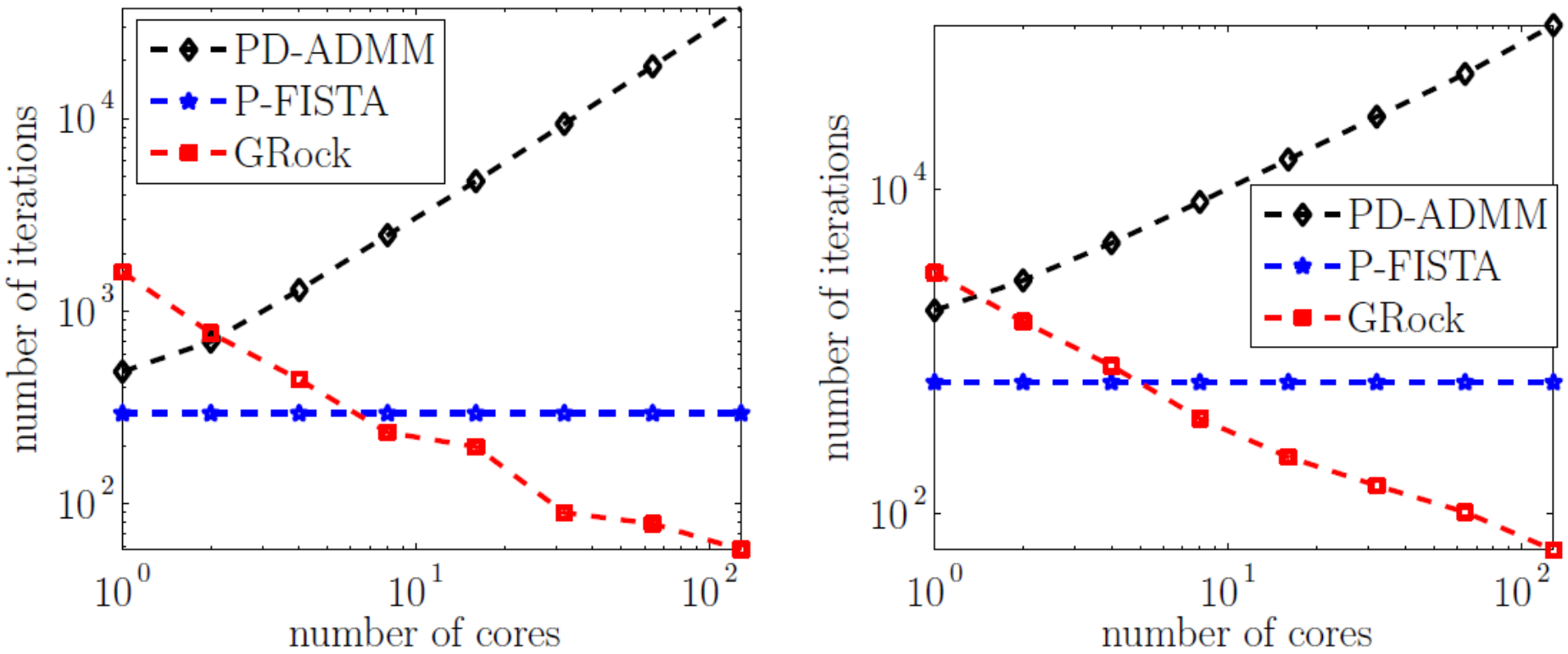

Figure: (a) Dataset I: cores vs iteration number; (b)Dataset II: cores vs iteration

Experiments on Cluster-continued

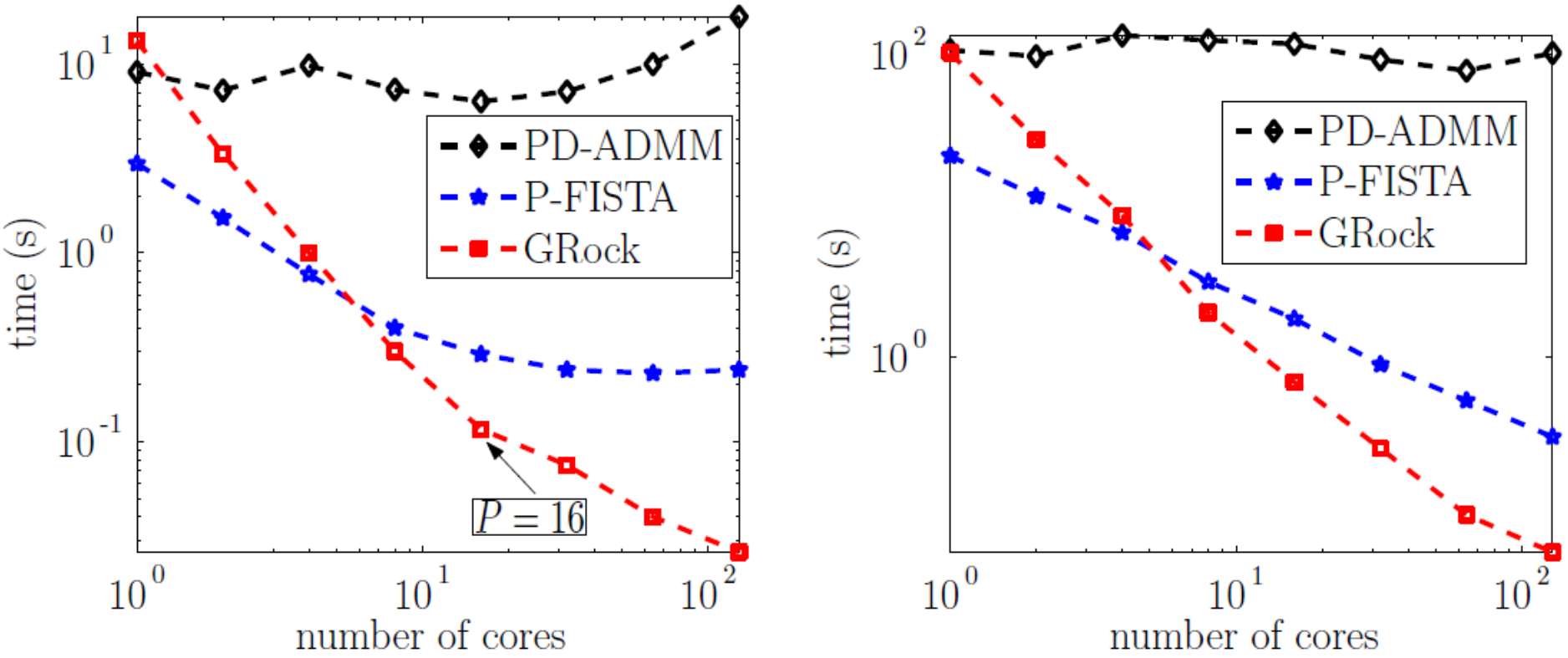

Figure: (a) Dataset I: cores vs time; (b)Dataset II: cores vs time

Complexity Analysis

- Each iteration of distributed prox-linear algorithm (Algorithm 1 and 2) takes $\mathcal{O}(mn/N)$, dominated by two matrix-vector multiplications, and $\mathcal{O}(nlogN)$ on communication for calling MPIAllreduce operator to assembly solutions from all nodes, $\sum_{i=1}^{N} A_i x_i$
- Each iteration of GRock takes $\mathcal{O}(mn/N + Pm)$ on computing, breaking down to $\mathcal{O}(mn/N)$ for one matrix-vector multiplication and $\mathcal{O}(Pm)$ for updating the residual $Ax - b$ as only P coordinates are updated.

Summary

- Decomposition of matrix A resulting in three scenarios, and it is the key to distribute proximal-linear algorithms over nodes.
- Greedy coordinate descent works perfect for sparse optimization problem, based on the strong orthogonality between selected column from A corresponding to selected coordinate.
- Based on above argument, a prerequisite condition for successfully having a converged solution by GRock is proposed.
- GRock is straight forward to process in parallel manner.