Let's Make Block Coordinate Descent Go Fast!

Julie Nutini, Issam Laradji, Mark Schmidt

University of British Columbia

November 15, 2018

Why Block Coordinate Descent?

- Block coordinate descent methods are key tools in large-scale optimization.
  - → Easy to implement.
  - → Low memory requirements.
  - → Cheap iteration costs.
  - → Adaptability to distributed settings.
  - → Ability to exploit problem structure.
  - → Good numerical performance.
- Used for almost two decades to solve LASSO and SVM problems.

→ **Any** improvements on convergence will affect many applications.

- This work: we propose several ways to make BCD much faster.

Block Coordinate Descent Framework

- We consider the basic optimization problem:

$$\min_{x \in \mathbb{R}^n} \; f(x),$$

  where f is differentiable and n is large.

- At each iteration of the BCD algorithm, we
  - Select a block $b_k \subseteq \{1, 2, \ldots, n\}$.
  - Update iterate according to

$$x^{k+1} = x^k + U_{b_k} d^k,$$

    where $d^k \in \mathbb{R}^M$ is a descent direction of the reduced dimensional subproblem,

$$\operatorname*{argmin}_{d \in \mathbb{R}^M} \; f(x^k + U_{b_k} d).$$

- There are many possible ways to define the blocks b_k and directions d^k.
  - E.g., take random τ variables and set $d^k = -\alpha_k \nabla_{b_k} f(x^k)$ for some $\alpha_k > 0$.

Why use coordinate descent?

- Theoretically, it is a provably bad algorithm:
  - The convergence rate is slower than gradient descent.
  - The iteration cost can be similar to gradient descent.
- But it is widely-used in practice:
  - Nothing seems to work better for certain problems.
  - Certain fields think it is the ‘ultimate’ algorithm.
- ???
- Renewed theoretical interest began with Nesterov [2010]:
  - Global convergence rate for randomized coordinate selection.
  - Faster than gradient descent if iterations are n times cheaper.

Problems Suitable for Coordinate Descent

- BCD most effective when updating all variables costs similar to gradient step.

$$\underbrace{\sum_{i=1}^{n} f_i(x_i)}_{\text{separable}} + \underbrace{\sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij}(x_i, x_j)}_{\text{pairwise separable}} + \underbrace{f(Ax)}_{\text{linear composition}}$$

  - f_i general convex functions (can be non-smooth).
  - f_{ij} and f are smooth.
  - A is a matrix and f is cheap.
- Key implementation ideas:
  - Separable part costs $O(1)$ for 1 partial derivative.
  - Pairwise part costs $O(n)$ for 1 partial derivative, instead of $O(n^2)$.
  - Linear compositions costs $O(1)$ for 1 partial derivative by tracking Ax.

Problems Suitable for Coordinate Descent

- Examples: least squares, logistic regression, lasso, SVMs.

$$\min_{x \in \mathbb{R}^n} \frac{1}{2}\|Ax - b\|^2 + \lambda \sum_{i=1}^{n} |x_i|,$$

- More examples: quadratics, graph-based label propagation, graphical models.

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} x^T A x + b^T x = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i x_j + \sum_{i=1}^{n} b_i x_i.$$

- BCD also allows group-separable variations (group L1-regularization).

- Fancier: tensor fact., log-det, convex extensions of submodular.

Cannonical Randomized BCD Algorithm

- Usual assumption: each block b is L_b-blockwise Lipschitz continuous,

$$\|\nabla_b f(x + U_b d) - \nabla_b f(x)\| \leq L_b \|d\|, \quad \text{for all } d,$$

  where for twice-differentiable functions this is equivalent to $\nabla^2_{bb} f(x) \preceq L_b I$.

- 3 ingredients of a “canonical” randomized BCD method:
  1. Partition the coordinates into n/τ blocks, using something like

$$\mathcal{B} = \{\{1, 2, \ldots, \tau\}, \{\tau+1, \tau+2, \ldots, 2\tau\}, \ldots, \{(n-\tau)+1, (n-\tau)+2, \ldots, n\}\}.$$

  2. Choose a block $b_k \in \mathcal{B}$, maybe uniformly at random.
  3. Take the step d_k, often a gradient step with $1/L_{b_k}$.
- This is not a competitive algorithm for many problems.
  - This talk: ways to make it go faster.

Analysis of Uniform Random BCD and Existing Improved Rules

- A blockwise version of the descent lemma is that

$$f(x^{k+1}) \leq f(x^k) + \langle \nabla f(x^k), x^{k+1} - x^k \rangle + \frac{L_{b_k}}{2} \|x^{k+1} - x^k\|^2.$$

- Plugging in our update gives the usual progress bound used for analysis,

$$f(x^{k+1}) \leq f(x^k) - \frac{1}{2L_{b_k}} \|\nabla_{b_k} f(x^k)\|_2^2.$$

- Taking expectation of b_k gives us a bound for uniform random sampling.

- Existing approaches to give tighter bounds (effecient for certain problems):
  - Lipschitz sampling: choosing b_k proportional to L_{b_k}.
  - Gauss-Southwell (GS): choosing b_k maximizing $\|\nabla_{b_k} f(x^k)\|$.

Gauss-Southwell???

- How is computing max(gradient) n-times cheaper than computing gradient?

- Consider a quadratic with a very-sparse Hessian.
  - Like 10 non-zeroes per column.
  - In this case, only 10 gradients change when you change one variable.
  - You can efficiently track the max using a max-heap structure.

- For pairwise objectives, need max-degree $\approx$ average-degree.
  - So it works for dense quadratics, too.
- For some problems, can approximate by nearest neighbours search.

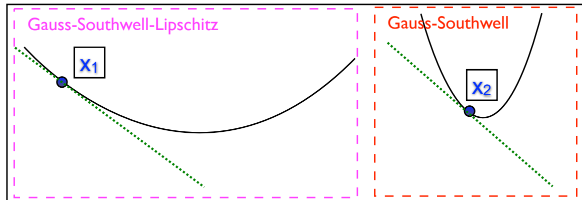
Gauss-Southwell-Lipschitz

- Consider maximizing the progress bound in terms of b_k,

$$f(x^{k+1}) \leq f(x^k) - \frac{1}{2L_{b_k}} \|\nabla_{b_k} f(x^k)\|_2^2.$$

- We call the rule that results the Gauss-Southwell-Lipschitz (GSL) rule:

$$b_k \in \underset{b \in \mathcal{B}}{\text{argmax}} \frac{\|\nabla_b f(x^k)\|_2}{\sqrt{L_b}},$$



- Prefers blocks with low Lipschitz constant if gradients are similar.

Fixed Blocks vs. Variable Blocks

- Fixed blocks (FB): partition the the coordinates into n/τ blocks:

$$\mathcal{B} = \{\{1, 2, \ldots, \tau\}, \{\tau + 1, \tau + 2, \ldots, 2\tau\}, \ldots, \{(n - \tau) + 1, (n - \tau) + 2, \ldots, n\}\}.$$

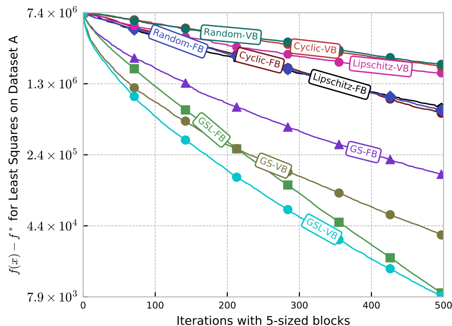
- Variable blocks (VB): $\mathcal{B}$ contains all possible blocks of size τ.
  - $\mathcal{B}$ is the set of b such that $|b| \leq \tau$.

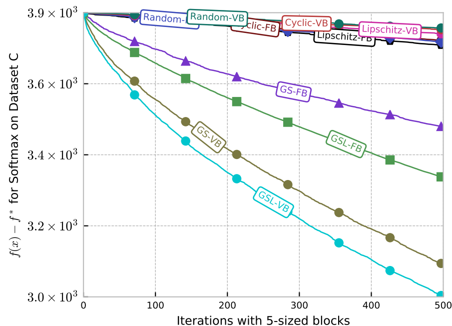
- With greedy rules, VB guarantees more progress.
  - Use VB if doesn't significantly increase the runtime.

- Although for some problems VB doesn't make sense:
  - Sparse multi-class logistic regression (VB is much more expensive).
  - Group L1-regularization (VB doesn't respect non-smooth group structure).
  - GSL rule (need an approximation to implement VB).

Greedy Rules with Gradient Updates

Performance on least sqaures (left) and 50-class logistic regression (right):





- Variable blocks give large improvement over fixed blocks for greedy rules.
- All methods worked better with line-search (not shown).
- As batch size increased:
  - Overall variance of methods decreased.
  - Benefit of line-search increased.
  - Benefit of GSL over GS increased.

Gauss-Southwell-Lipschitz vs. Maximum Improvement Rule

- The ideal rule is the maximum improvement (MI) rule:
  - The update of τ coordinates that maximally decreases f.
- GSL is equivalent to MI for quadratic functions when $\tau = 1$.
  - But not for $\tau > 1$.
- And should we really be doing gradient steps anyways?

Newton-Steps and Quadratic-Norms

- Assume that f is blockwise Lipschitz in a set of quadratic norms

$$\|\nabla_b f(x + U_b d) - \nabla_b f(x)\|_{H_b^{-1}} \leq \|d\|_{H_b} = \sqrt{d^T H_b d},$$

where the H_b are positive-definite matrices.

- This isn't a stronger assumption, just a change in how we measure.

- The descent lemma now becomes

$$f(x^{k+1}) \leq f(x^k) + \langle \nabla_{b_k} f(x^k), d^k \rangle + \frac{1}{2}\|d^k\|^2_{H_{b_k}},$$

and the optimal d^k is

$$d^k = -(H_{b_k})^{-1} \nabla_{b_k} f(x^k).$$

- This matrix update is similar to Newton, but using upper-bound on Hessian.

Gauss-Southwell-Quadratic Rule

- The optimal matrix update according to the progress bound is

$$b_k \in \underset{b \in \mathcal{B}}{\text{argmax}} \left\{ \|\nabla_b f(x^k)\|_{H_b^{-1}} \right\},$$

which we call the Gauss-Southwell-Quadratic (GSQ) rule.

  - Equivalent to MI rule for quadratics.
  - But memory/computationally expensive.

- A practical alternative is a diagonal approximation (GSD),

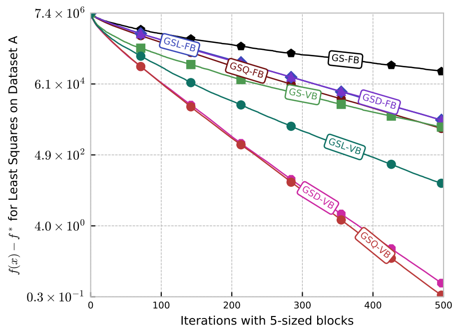
$$b_k \in \underset{b \in \mathcal{B}}{\text{argmax}} \left\{ \sum_{i \in b} \frac{|\nabla_i f(x^k)|^2}{D_{i,b}} \right\},$$

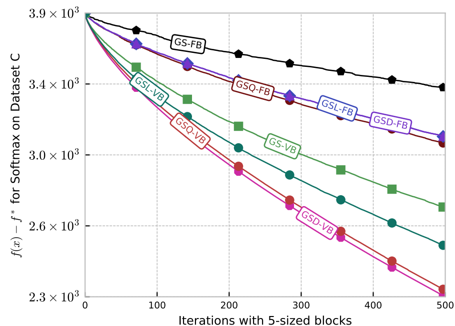
although we still use the full matrix update after selecting block.

  - Has same cost as GS under constraint that $D_{i,b} = d_i$ for set of d_i values.

Greedy Rules with Matrix Updates

Performance on least sqaures (left) and 50-class logistic regression (right):





- Here VB works much better than FB (difference larger for large batches).
- There wasn't a large advantage to using GSQ over simpler GSD.
  - "A little Lipschitz information is all that's needed" (here we use $d_i = L_i$).

Matrix vs. Newton Updates

- The matrix update updates the block b_k using

$$x_{b_k}^{k+1} = x_{b_k}^k - (H_{b_k})^{-1}\nabla_{b_k} f(x^k),$$

  based on an upper-bound H_{b_k}.

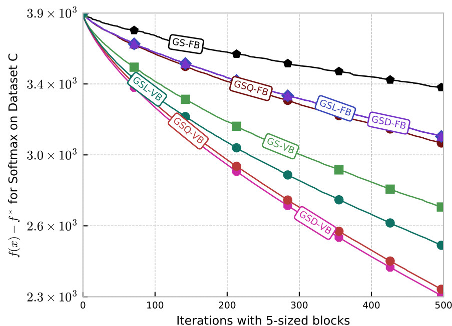
- For non-quadratic functions, Newton updates might make more progress,

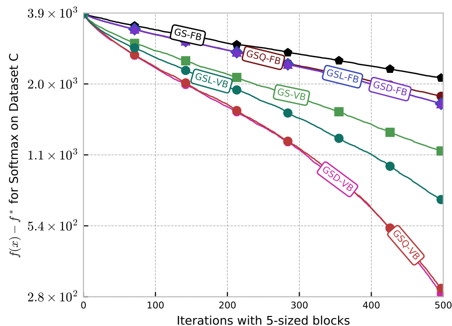
$$x_{b_k}^{k+1} = x_{b_k}^k - \alpha_k(\nabla^2_{b_k b_k} f(x^k))^{-1}\nabla_{b_k} f(x^k),$$

  for a step-size α_k.
- For example, we might have $\nabla^2_{b_k b_k} f(x^k) \prec\prec H_{b_k}$.
  - Requires a line-search, but this is usually cheap on the block.

Greedy Rules with Newton Updates

Performance on 50-class logistic regression with matrix updates (left) and Newton updates (right):





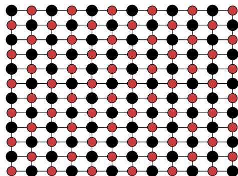
- Notice the difference in the y-axis.
- For variable blocks, the difference increases with the block size.

Cost of Higher-Order Updates

- Problem with matrix and Newton updates:
  - $O(\tau^3)$ cost to solve a linear system with τ variables.
- Can we do better for problems with sparse Hessians?
  - Gaussian elimination still requires $O(\tau^3)$ due to "fill-in".
  - Iterative solvers use sparsity but depend on condition number of block.
- An alternative approach: choose the blocks to guarantee no "fill-in".
  - Allows exact solution of Newton system in $O(\tau)$ to update "huge" blocks.

Graph-Colouring for Block Partitioning

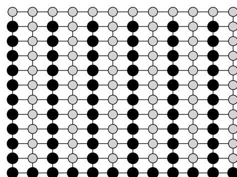
- Consider treating the non-zero pattern in $\nabla^2 f(x^k)$ as an adjacency matrix:



- A classic BCD approach is “red-black” ordering:
  - Partition the nodes via graph colouring.
  - Use the colouring as the blocks.
  - Guarantees that sub-Hessians $\nabla^2_{bb} f(x^k)$ are diagonal.
    - So Newton step costs $O(\tau)$.
- In the lattice example, we update blocks of size $n/2$ in $O(n)$.

Tree-Partitioning for Block Partitioning

- Diagonal matrices are not the only structure that allows $O(\tau)$ solutions.
- We considered forest-structured blocks:



- Allows dependencies within the block, but can be solved in $O(\tau)$.
- Key idea: define an arbitrary "root" of each tree and divide nodes into "levels".
  - Gaussian elimination starting from "leaves" guarantees no "fill-in".

Solving Forest-Structured Linear Systems in Linear Time

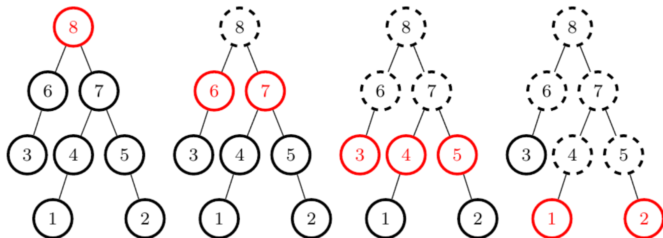
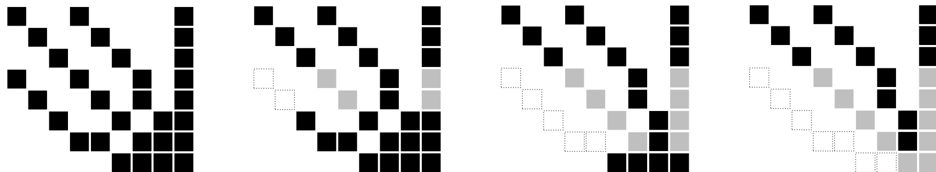


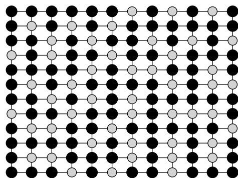
FIG. 1. *Process of partitioning nodes into level sets. For the above graph we have the following sets:* $L\{1\} = \{8\}, L\{2\} = \{6,7\}, L\{3\} = \{3,4,5\}$ *and* $L\{4\} = \{1,2\}$.

- Run Gaussian elimination from leaves to root:



Greedy Forest-Structured Blocks

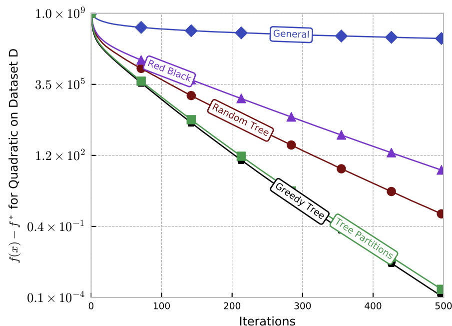
- Instead of partitioning nodes into forests ("fixed blocks"), we could find a new forest to update at each iteration ("variable blocks").

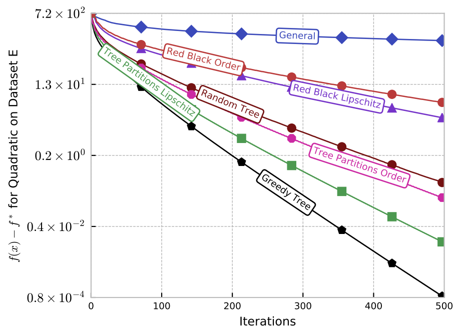


- We give an $O(n \log n + |E|)$-time algorithm to approximate GS forest.
  - Based on sorting and using two levels of hashing.
  - In the lattice example it tends to update $\approx 2n/3$ nodes.

Experiment: Sparse Quadratic Problem

- Comparing different methods with $O(n)$ cost on lattice (left) and label-prop (right):





- Huge structured blocks improve significantly over GS rule with smaller blocks.

Superlinear Convergence?

- When we think of Newton, we normally think superlinear convergence.
- Does BCD have superlinear convergence?
- No, not even with exact updates.
  - E.g., 2-variable non-separable quadratic

$\rightarrow$ Possible to get superlinear convergence for problems with certain structures.

Optimization with Bound Constraints

- Consider optimizing a smooth f plus a separable non-smooth g,

$$\operatorname*{argmin}_{x \in \mathbb{R}^d} f(x) + \sum_{i=1}^{n} g_i(x_i),$$

  which includes bound constraints and L1-regularization.
- In this context we can use proximal gradient steps,

$$x^{k+1} = \mathbf{prox}_{\alpha_k g_{b_k}} \left[x^k - \alpha_k U_{b_k} \nabla_{b_k} f(x^k) \right],$$

  and most issues are similar:
  - FB vs. VB, gradient updates vs Newton updates, random vs. greedy selection.

- Some differences:
  - There are 4 non-equivalent generalizations of the Gauss-Southwell.
  - The non-smoothness can lead to a faster convergence rate.

Manifold Identification Property

- Consider a problem with non-negative constraints,

$$\underset{x \in \mathbb{R}^d}{\text{argmin}}\, f(x) + \delta(x \geq 0).$$

- In this case proximal-gradient becomes projected-gradient:

$$x^{k+1} = \left[x^k - \alpha_k U_{b_k} \nabla_{b_k} f(x^k)\right]^+.$$

- The non-negative constraints mean that we often obtain a sparse solution.
    - E.g., non-negative matrix factorization.

- Manifold identification property:
    - For all k larger than some k', sparsity pattern of x^k is the same as optimal x^*.

- Once you have the manifold, algorithm converges faster on this subspace.

Manifold Identification Property

- Manifold identification for bound constraints requires a mild assumption:

$$\nabla_i f(x^*) \geq \delta > 0 \text{ if } x_i^* = 0.$$

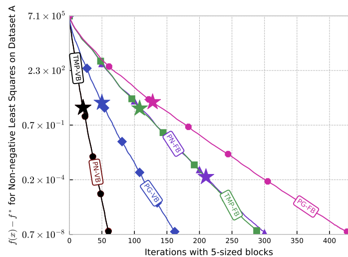
- We give a simple proof that greedy BCD finitely identifies manifold.
  - Previously known for cyclic and random BCD.
- We give upper bounds on number of iterations before manifold is identified:
  - For projected-gradient it happens after at most $\kappa \log(2L\|x^0 - x^*\|/\delta)$ iterations.
  - Bound is slightly more complicated for BCD methods.
- Similar results hold for other non-smooth g_i like L1-regularization.
  - Assumption changes to $\nabla_i f(x^*) \in \text{int } \partial g_i(x_i^*)$.

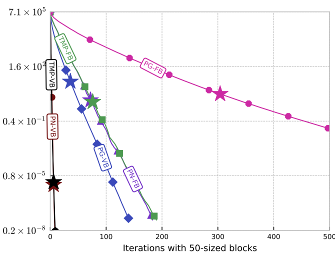
Superlinear Convergence and Proximal-Newton

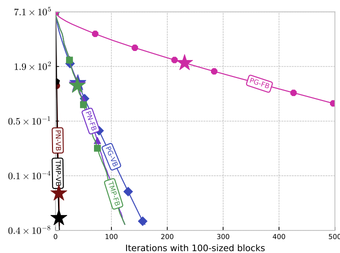
- Manifold property suggest an obvious strategy:
  - Run BCD for a fixed number iterations to identify manifold.
  - Then apply (unconstrained) Newton method on non-zero variables.
- But don’t know how logn to run BCD (just have upper bounds).
- Some alternatives to “switching”:
  - Hybrid methods: try BCD and Newton step, take best.
  - Proximal-Newton BCD steps (not too expensive if blocks are small).
    - Can be computed exactly for piecewise-linear g_i via homotopy methods.
  - Two-metric projection BCD steps (compromise between cost/progress).
- Superlinear convergence if using greedy rules with large-enough VB.

Superlinear Convergence and Proximal-Newton

- Experiments with L1-regularized least squares:







- Projected-Newton converges extremely quickly.
- Two-metric projection is equally fast but with cheaper iterations.
- For large greedy/variable blocks, both methods converge finitely.

Summary

- We proposed improved greedy rules for BCD methods.
  - Incorporate gradient and Lipschitz information.
  - Make substantially more progress on some problems.

- If you can afford to compute second-order information, you should.
  - Newton updates with line-search tend to outperform fixed-matrix updates.
  - Linear-time Newton steps for forest-structured blocks.
  - Two-metric projection can handle constrained/non-smooth cases.

- We give non-asymptotic bounds on number of iterations to reach manifolds.
  - Superlinear or finite convergence of BCD in some special cases.
  - Bounds are probably useful in other settings.