Graphical Newton for Huge-Block Coordinate Descent on Sparse Graphs

Issam Laradji, Julie Nutini and Mark Schmidt

Summary

- **Motivation**:
  - Block coordinate descent is widely-used in machine learning.
    - Easy to implement, cheap iteration cost, low memory requirements.
    - Especially useful for problems with sparse dependencies between variables.
  - Recent works use Newton updates on the blocks.
    - Makes *more progress per iteration* than gradient steps.
    - This is the *optimal update* for quadratic objectives.
  - But this costs $O(|b|^3)$ even for sparse problems.
    - So standard methods are forced to use small blocks.
- **Contribution**:
  - For sparse problems we propose to use forest-structured blocks.
    - Allows us to implement the Newton step in $O(|b|)$.
  - We propose random and greedy rules for selecting forests.
    - Results in more progress per iteration than standard rules.

Example: Sparse Quadratic Functions and Newton's Method

- Consider a quadratic objective,

$$\operatorname*{argmin}_{x\in\mathbb{R}^n} \frac{1}{2}x^T Ax - c^T x.$$

- The optimal update for block b is given by the solution of the linear system

$$A_{bb}x_b = \tilde{c}_b,$$

for $\tilde{c} = c_b - A_{b\bar{b}}x_{\bar{b}}$.
- If A_{bb} is unstructured solving this linear system costs $O(|b|^3)$.
- Classic approach ("red-black ordering") chooses b so that A_{bb} is diagonal.
  - Because of sparsity pattern we can solve the linear system in $O(|b|)$.

- We consider more general case where A_{bb} has a forest structure.
  - We can still solve it in $O(|b|)$ update, but it allows dependencies within the block.
- For non-quadratic updates, we need to solve the Newton system,

$$\nabla^2_{bb} f(x^k)d = -\nabla_b f(x^k).$$

  - If we pick b so $\nabla^2_{bb} f(x^k)$ forms a forest, we can solve it in $O(|b|)$ instead of $O(|b|^3)$.
    - We can update "huge" blocks.

Message Passing for Forest-Structured Linear Systems

- Let G be the graph representing the sparsity pattern of the matrix $\nabla^2 f(x^k)$.
  - And let G_b be the graph representing the sparsity pattern of $\nabla^2_{bb} f(x^k)$.
- If G_b forms a forest, we can solve the linear system in linear time:
  - Divide the nodes in each tree in b into sets $L\{1\}, L\{2\}, \ldots, L\{T\}$.
    - $L\{1\}$ is an arbitrary node in the graph chosen as the "root" node
    - $L\{2\}$ is the set of the root node neighbours
    - $L\{3\}$ is the set of the $L\{2\}$ neighbours - excluding parent nodes
    - The process continues until all nodes are assigned to a set

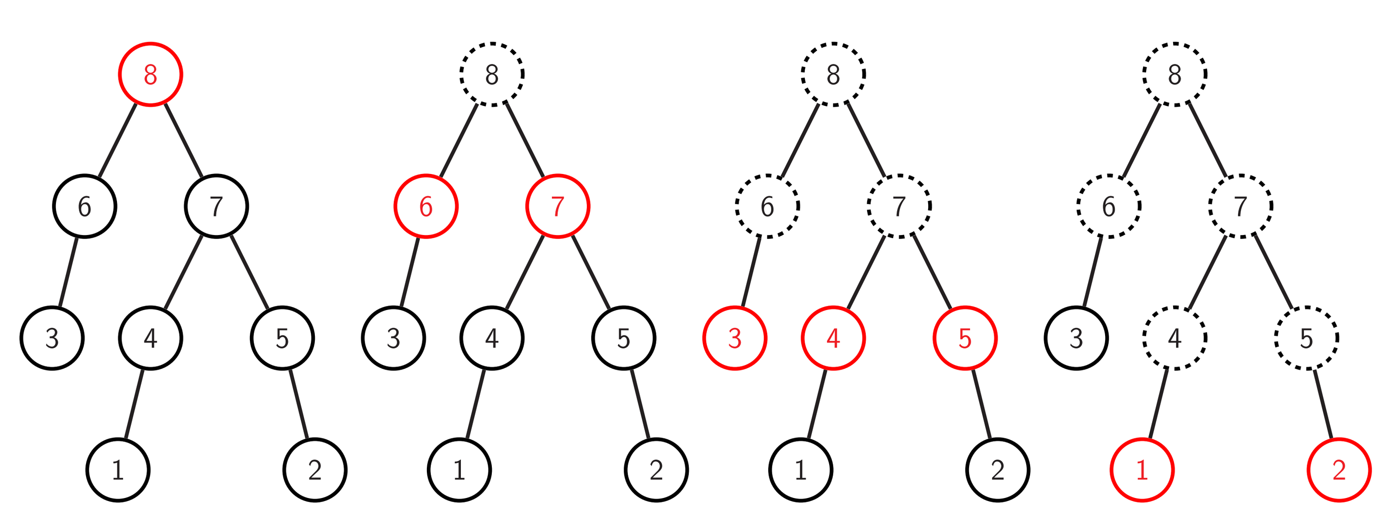
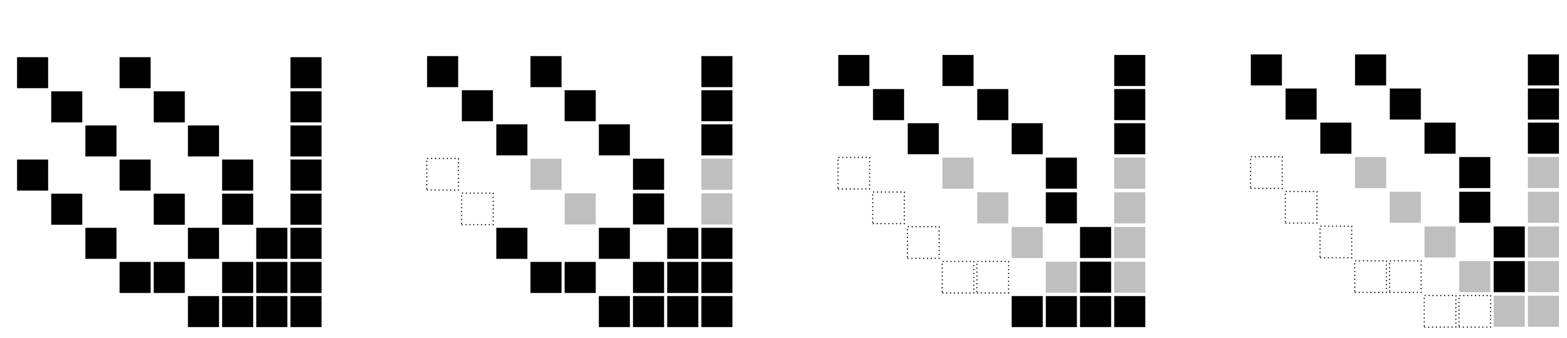


Figure: Process of partitioning nodes into level sets. For the above graph we have the following sets: $L\{1\} = \{8\}, L\{2\} = \{6, 7\}, L\{3\} = \{3, 4, 5\}$ and $L\{4\} = \{1, 2\}$.

- **Row operations**
  - Select the nodes furthest from the root node, $L\{T\}$.
  - Carry out the row operations of Gaussian elimination on $L\{T\}$, $L\{T-1\}$,...



- Forest structure and this elimination order guarantees there is no fill-in.
  - Total cost is $O(|b|)$.

Greedy Selection of Forest-Structured Blocks

- The General Gauss-Southwell rule chooses the "best" block b to update using

$$\operatorname*{argmin}_{|b|\le\tau}\{\|\nabla f(x^k)\|^2\}.$$

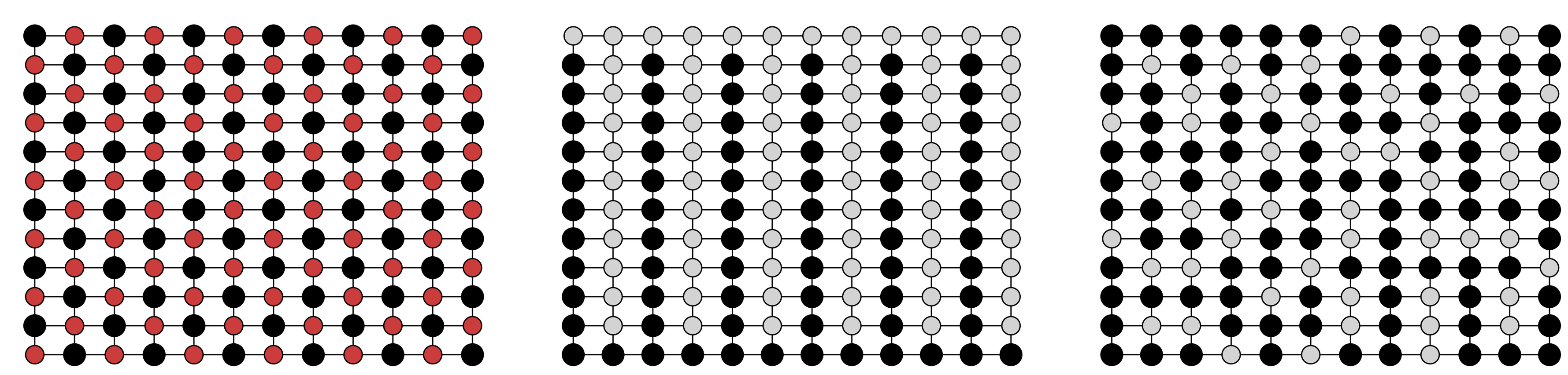
- We can solve this problem by sorting the $|\nabla f(x^k)|$ values and taking τ largest.
  - But this ignores the structure, so the Newton update costs $O(\tau^3)$
  - We need to choose $\tau = \sqrt[3]{n}$ for Newton update to have linear cost in n.
- The Tree Gauss-Southwell rule chooses the "best" block among all forests $\mathcal{F}$,

$$\operatorname*{argmin}_{b\in\mathcal{F}}\{\|\nabla f(x^k)\|^2\}.$$

- For sparse graphs some forests may have $O(n)$ nodes ("huge blocks").
  - But the cost of the update is always in $O(n)$.
- This is NP-hard to compute so we use a greedy approximation:
  1. Initialize b with the node i corresponding to the largest gradient, $|\nabla_i f(x^k)|$.
  2. Add to b the node i with largest gradient that maintains the forest property.
  3. Repeat until no nodes can be added to b.
- This can be implemented in $O(n \log n + |E|)$ by sorting and hashing.
- Alternatives are random forests or cycling through a partition into trees.

Comparison of Forest-Structured Blocks with Lattice Dependency

- Comparison of red-black ordering, tree partition, and greedy forest rules:



- Red-black and tree-partition methods update $n/2$ nodes.
  - Red-black takes $O(\sqrt{n})$ iterations to propagate information between all nodes.
  - Tree-partition and greedy only need 2 iterations.
- Greedy method tends to update around $2n/3$ nodes.

Empirical Evaluation

- We compared BCD methods with different selection on label propagation problems,

$$\min_{y_i|i\notin S} \frac{1}{2}\sum_{i=1}^{b}\sum_{j=1}^{n} w_{ij}(y_i - y_j)^2$$

- We considered lattice-structured problem and a semi-supervised learning: