Newton-Laplace Updates for Block Coordinate Descent

Si Yi Meng Mark Schmidt
University of British Columbia

Motivation

- **Block coordinate descent (BCD)** methods are used to optimize many machine learning objectives.
  - Easy to implement.
  - Low memory requirements.
  - Cheap iteration costs.
  - Adaptability to distributed settings.
- Applications: Group Lasso, SVMs, etc.
- Consider the problem: $\arg\min_{x\in\mathbb{R}^n} f(x)$ where f is convex and twice continuously differentiable.

BCD update rule

$$x_{b_k}^{k+1} = x_{b_k}^{k} + d^k$$

- First order updates use a gradient descent direction:

$$d^k = -\alpha_k \nabla_{b_k} f(x^k) \qquad \text{where } \alpha^k \text{ is the step size.}$$

- Speed up convergence using

Blockwise-Newton updates

$$d^k = -\alpha_k \left[\nabla^2_{b_k b_k} f(x^k)\right]^{-1} \nabla_{b_k} f(x^k)$$

  - Possible to obtain superlinear convergence for problems with certain structures
  - Use of larger blocks can lead to faster convergence, but iteration cost is $O(|b_k|^3)$
  - Step sizes are often chosen with line-search.

Contributions

Previously:

- When the chosen block's sparsity pattern has a tree structure, "message-passing" algorithms can be used to solve the system in linear time.
- One can also exploit the width of the Hessian's computation graph to speed up the blockwise-Newton update.
- There could still be a limit to how large the blocks can grow until these structural constraints are violated.

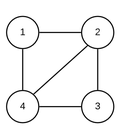
Our work:

- **Newton-Laplace updates**: if the sub-Hessian corresponding to the blocks have a Laplacian/SDD structure, we can use fast Laplacian solvers to compute the blockwise-Newton update in near-linear time.
- Empirically demonstrate its fast convergence rate for the graph-based semi-supervised learning objectives.

Laplacian solver

Laplacian matrix:

- Consider an undirected, possibly weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$:
  - $n = |\mathcal{V}|$ is number of vertices.
  - Adjacency matrix W and degree matrix D where $D_{ii} = \sum_j W_{ij}$.
  - The Laplacian matrix is defined as $L = D - W$.
- L has many nice properties:
  - Symmetric and diagonally dominant (SDD): $L_{ii} \geq \sum_{j\neq i} |L_{ij}|$
  - Positive semi-definite: $L^T = L \succeq 0$.
  - Multiplicity of the eigenvalue 0 indicates the number of connected components.



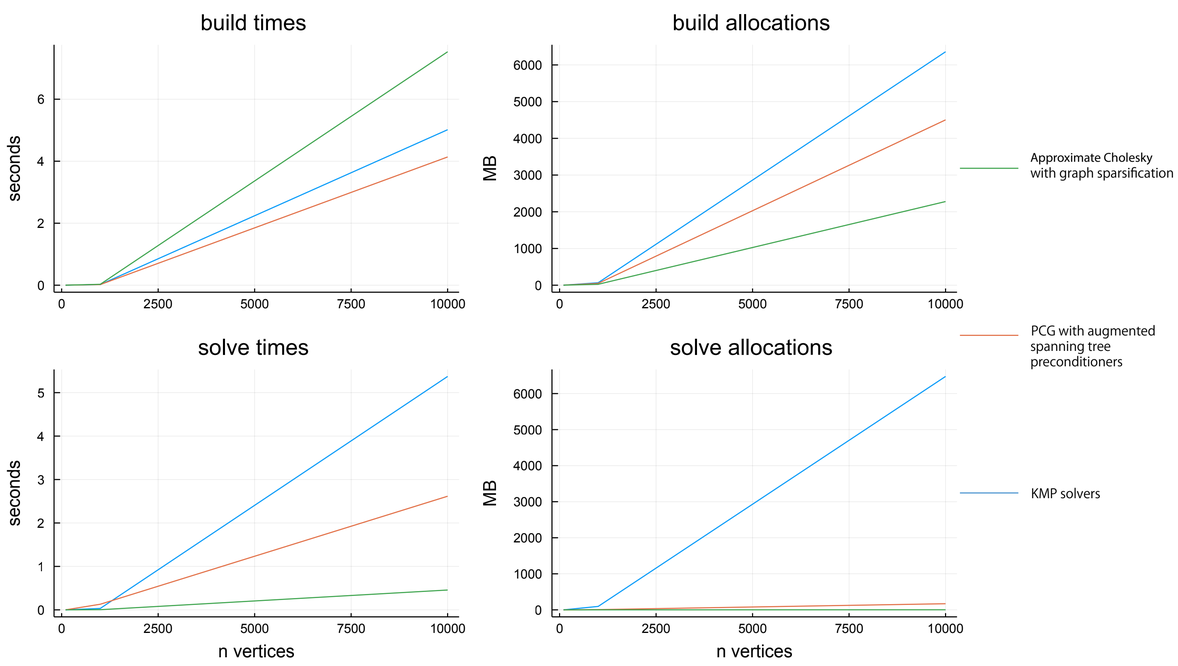
$$\begin{bmatrix} 2 & -1 & 0 & -1 \\ -1 & 3 & -1 & -1 \\ 0 & -1 & 2 & -1 \\ -1 & -1 & -1 & 3 \end{bmatrix}$$

Laplacian systems $Lx = b$:

- These systems arise in many applications: graph-based semi-supervised learning, solutions of PDEs via finite element, max flows, resistor networks, etc.
- A natural approach to solving symmetric linear systems is via Cholesky factorization.
  - But fill-in can create a bottleneck: the triangular factors $\mathcal{L}$ in $L = \mathcal{L}\mathcal{D}\mathcal{L}^T$ can be dense even when L is sparse.
  - For Laplacian systems, variable elimination corresponds to vertex eliminiation, and fill-in corresponds to adding a clique to the vertex's neighbours.

Fast solvers:

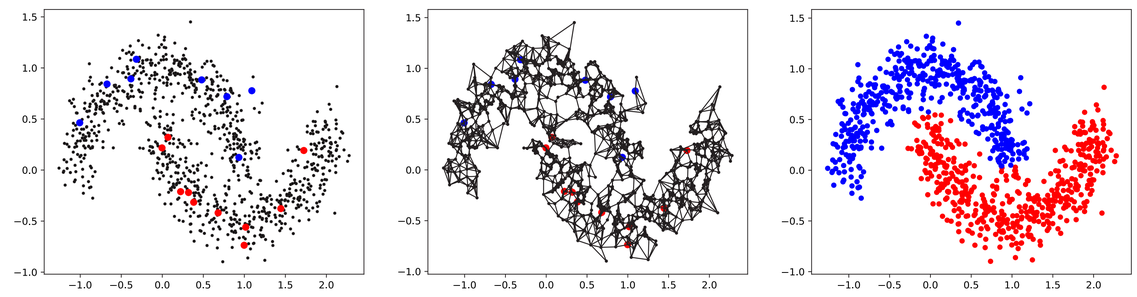
- Generate a sparse, approximate Cholesky decomposition for Laplacian matrices by:
  - Randomizing the order of elimination, which allows for bound on the sample variance.
  - Simpler procedure to estimate the effective resistances for computing the edge sampling probabilities.
- Cost is near-linear in the number of non-zeros in L.
- Efficient implementations in Julia for sparse Laplacian/SDD systems: Laplacians.jl
- Performance benchmark (with 10% sparsity in L):



Application

Graph-based semi-supervised learning problem:

- Given: a partial labeling of n examples $x = (x_l, x_u)$ and pairwise similarities w_{ij}, where l is the set of indices corresponds to labeled examples, and $u = [n] \setminus l$ is for unlabeled.
- Goal: infer the missing labels.



Label propagation objective

$$f(x_u) = \frac{1}{2} \sum_{i\in[n],\, j\in u} w_{ij}\, h(x_i - x_j) + \frac{1}{2} \sum_{i,\, j\in u} w_{ij}\, h(x_i - x_j).$$

- When $\mathbf{h(z) = z^2}$,

$$\nabla f(x_u) = (D_{uu} - W_{uu})\, x_u - W_{ul}\, y_l$$
$$\nabla^2 f(x_u) = L_{uu}$$

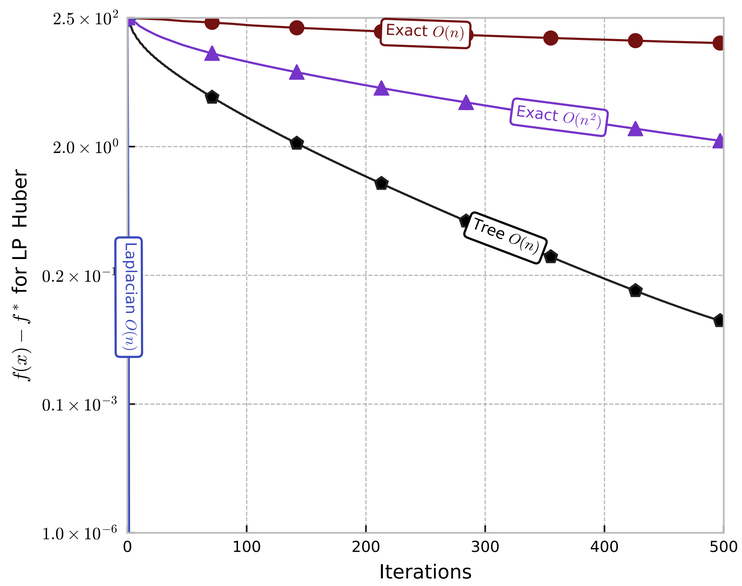
  where L_{uu} is the Laplacian on the graph formed by the unlabeled examples, and the analytical solution can be obtained from $x_u^* = L_{uu}^{-1}(W_{ul}\, x_l)$.
- Thus the (sub-)Hessian of f is a Laplacian matrix, and every blockwise-Newton update therefore involves solving a Laplacian system.
- We can replace $h(\cdot)$ with the huber loss:

$$h_\epsilon(z) = \begin{cases} \frac{1}{2}z^2 & \text{for } |z| \leq \epsilon \\ \epsilon(|z| - \frac{1}{2}\epsilon) & \text{otherwise} \end{cases}$$

  and the resulting update will also be a Laplacian system.
- Best of both worlds!
  - Cheap iterations using Laplacian solvers.
  - Fast convergence from second order updates.

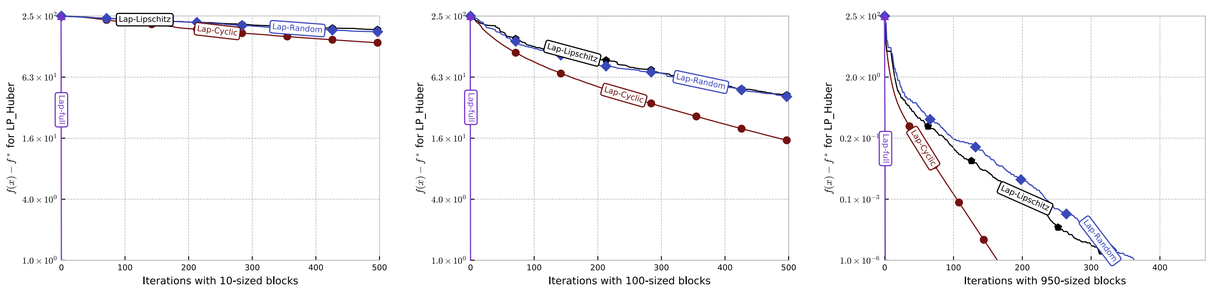
Experiments

- ℓ_2-regularized label propagation on the **"two-moons"** dataset.
  - 2000 examples with 100 labeled and $n = 1900$ unlabeled.
- Pairwise label distances measured in the **Huber loss**.
- **Newton-Laplace updates** using the approximate Cholesky Laplacian solver.



- **Using exact solvers with fixed-size blocks chosen to satisfy a particular iteration cost:**
  - For example, choosing $|b_k| = n^{1/3}$ gives an iteration cost of $O(n)$ with generic, exact solvers.
  - Although we are able to obtain a slightly faster convergence rate with larger blocks, the associated iteration costs have increased from $O(n)$ to $O(n^2)$.
- Using tree partitioning and compute the update direction using "message-passing" algorithms:
  - We can now use larger blocks with $O(n)$ iteration cost, but block size is still limited.
- Using Laplacian solvers:
  - We can use full blocks and reach the minimum within 10 iterations.

Convergence rate comparison of using different fixed-block sizes using different block selection strategies:



- All of these bear only a $O(|b|)$ iteration cost.

Take home message:

When memory is not an issue, one should take advantage of these solvers when the Hessian has or can be closely approximated with a Laplacian/SDD structure.