Graphical Newton for Huge-Block Coordinate Descent on Sparse Graphs

Issam Laradji
University of British Columbia
Julie Nutini
University of British Columbia
Mark Schmidt
University of British Columbia

Abstract

Block coordinate descent (BCD) methods are often very effective for optimization problems where dependencies between variables are sparse. These methods can make a substantial amount of progress by applying Newton's method to update a block of variables, but this leads to an iteration cost of $O(|b|^3)$ in terms of the block size $|b|$. In this paper, we show how to use message-passing to compute the Newton step in $O(|b|)$ when the block has a forest-structured dependency. Consequently, this allows us to update huge blocks for sparse problems, resulting in significant numerical improvements over existing approaches. We also present a greedy approach for selecting forest-structured blocks.

1 Motivation

Block coordinate descent (BCD) methods are widely-used for large-scale numerical optimization, and are especially useful when we have sparse dependencies between variables. Several authors have recently explored BCD methods that apply second-order Newton steps to update the block of coordinates at each iteration [Qu et al., 2015, Fountoulakis and Tappenden, 2015]. This makes more progress on each iteration than gradient updates, and is the optimal update for quadratic functions. However, even if the sub-Hessian associated with the block is very sparse, the cost of this update is *cubic* in the block size in the worst case so this approach can only be applied with medium-sized blocks.

In this work we propose to use *forest-structured* blocks, which allow us to use message passing algorithms in order to reduce the cost of the Newton update to be *linear* in the block size. This can be viewed as a BCD version of the "graphical Newton" approach of Srinivasan and Todorov [2015].[1] This previous approach still has a superlinear runtime in the worst case, but we propose to specifically *choose* the blocks so that they guarantee a linear runtime. Our numerical results show that this approach converges significantly faster than using general block structures that have the same runtime. We also propose a new greedy block selection rule that tries to approximate the optimal forest-structured block to update.

2 Notation

Consider minimizing a quadratic objective function,

$$\arg\min_{x \in \mathbb{R}^n} \frac{1}{2} x^T A x - c^T x, \tag{1}$$

where $A \in \mathbb{R}^{n \times n}$ is a positive-definite sparse matrix and $c \in \mathbb{R}^n$. The optimal update for a block of coordinates $b \subseteq \{1, 2, \dots, n\}$ is given by the solution of the linear system

$$A_{bb} x_b = \tilde{c}_b, \tag{2}$$

where $\tilde{c} = c_b - A_{b\bar{b}} x_{\bar{b}}$ with $\bar{b}$ defined as the complement of b. We note that in practice efficient BCD methods already need to track Ax so computing $\tilde{c}$ is efficient. In the case of non-quadratic objectives, the Newton

[1] The core idea is that certain sparsity patterns lead to limited fill-in when we perform Gaussian elimination. This idea dates back to the 1960s for the case of forests [Parter, 1961] and the 1970s for general graphs [Rose, 1970].

OPTML 2017: 10th NIPS Workshop on *Optimization for Machine Learning* (NIPS 2017).

update for a block of variables requires the solution of the Newton system,

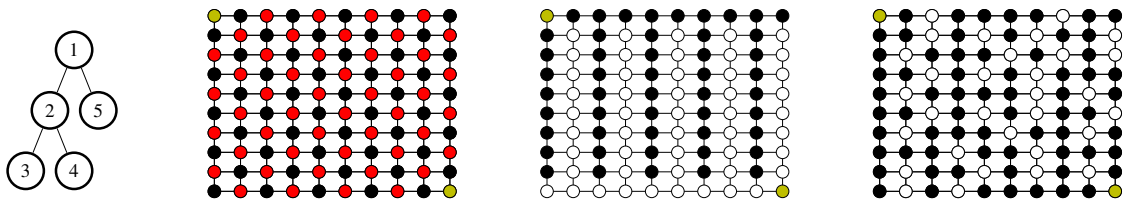
$$\nabla^2_{bb} f(x) d = -\nabla_b f(x), \tag{3}$$

for our block b and the current iterate x. The worst-case cost of solving both of the linear systems (2) and (3) is $O(|b|^3)$ even if the matrix on the left side has only $O(|b|)$ non-zero values. This limits our ability to update large blocks of variables. An alternative to computing the exact update is to use an approximation to the Newton update that has a runtime dominated by the sparsity level of the sub-Hessian. For example, we could use conjugate gradient methods or use randomized Hessian approximations [Dembo et al., 1982, Pilanci and Wainwright, 2015]. However, these approximations require setting an approximation accuracy and may be inaccurate if the sub-Hessian is not well-conditioned. In this work we consider an alternative approach: choosing blocks with a sparsity pattern that guarantees we can solve the resulting linear system in $O(|b|)$ using a "message-passing" algorithm. If the sparsity pattern is favourable, then this allows us to update huge blocks at each iteration.

3 Efficient message passing for sparse quadratics

Consider a pairwise undirected graph G, where each vertex in the set V corresponds to a coordinate of our problem and the set of edges $(i, j) \in E$ are given by the non-zero off-diagonal elements of A in (1). Thus, if A is diagonal then G has no edges, if A is dense then there are edges between all nodes (G is fully-connected), if A is tridiagonal then edges connect adjacent nodes (G is a chain-structured graph where $(1) - (2) - (3) - (4) - \dots$), and so on.

For BCD methods, unless we have a block size $n_b = n$, we only work with a subset of the nodes of G at each iteration corresponding to the reduced subproblem in (2). We define the *induced subgraph* G_b with respect to a block of coordinates b to be the graph obtained from the sub-matrix A_{bb}. Specifically, the nodes $V_b \in G_b$ are the coordinates in the set b, while the edges $E_b \in G_b$ are all edges $(i, j) \in E$ where both i and j are in b. We are interested in the special case where the structure of the induced sub-graph G_b forms a *forest*, meaning that it has no cycles.[2] In the special case of forest-structured induced subgraphs, we can compute the optimal update (2) in linear time using message passing [Shental et al., 2008] instead of the cubic worst-case time required by typical matrix factorization implementations. Indeed, in this case, the message passing algorithm is equivalent to Gaussian elimination [Bickson, 2009, Prop. 3.4.1] where the amount of "fill-in" is guaranteed to be linear [Parter, 1961].



(a) Tree graph.

(b) Red-black ordering.

(c) Tree-structured ordering.

(d) GSTree ordering.

Figure 1: (a) An example of a tree graph. (b)-(d) Examples of ordering strategies. Yellow nodes are the labeled nodes and the black nodes correspond to the selected block.

Without loss of generality, assume a forest-structured subgraph G_b defined by the block $b = \{1, 2, \dots, |b|\}$. To illustrate the message-passing algorithm in the terminology of Gaussian elimination, we first need to divide the nodes in b into sets $L\{1\}, L\{2\}, \dots, L\{T\}$, where $L\{1\}$ is an arbitrary node in graph G_b selected to be the root node, $L\{2\}$ is the set of all neighbours of the root node, $L\{3\}$ is the set of all neighbours of the nodes in $L\{2\}$ excluding parent nodes (nodes in $L\{1\!:\!2\}$), and so on until all nodes are assigned to a set (we repeat this for another root node if our graph is not connected). An instance of a graph is shown in Figure 1a, where the node partition level sets are: $L\{1\} = \{1\}, L\{2\} = \{2, 5\}$, and $L\{3\} = \{3, 4\}$. Once these sets are initialized, we start with the nodes furthest from the root node, $L\{T\}$ and carry out the row operations of Gaussian elimination. Then we use forward substitution to solve the system $A_{bb}x = \tilde{c}_b$. Performing Gaussian elimination in this order results in no "fill-in", and lets us solve the linear system in $O(|b|)$.

[2] An undirected cycle is a sequence of adjacent nodes in V starting and ending at the same node, where there are no repetitions of nodes or edges other than the final node.

4 Selecting forest-structured blocks

Whether or not message-passing is useful will depend on the sparsity pattern of A. In this section we explore different ways to select a block of coordinates b such that the resulting subgraph forms a forest. As our motivating example, consider semi-supervised learning based on the quadratic labeling criterion of Bengio et al. [2006], $\min_{y_i|i\notin S} \frac{1}{2}\sum_{i=1}^{b}\sum_{j=1}^{n} w_{ij}(y_i - y_j)^2$. Here, S is the set of labeled nodes (such as the yellow nodes in Figure 1), and w_{ij} represents how strongly we want labels y_i and y_j to be the same (if it is zero, then there is no edge between nodes y_i and y_j). The goal is to propagate the labels across the graph.

A classic partitioning strategy for problems with a lattice-structured graph is to use a "red-black ordering" (see Figure 1b), where at each iteration of the BCD method we either update all of the red nodes or we update all of the black nodes. In this setting, A_{bb} is diagonal and hence the induced subgraph with respect to the red nodes is a disconnected graph (and similarly for the black nodes). Since disconnected graphs have no cycles, given the values of the yellow nodes we can compute the optimal value of all the black nodes in linear time using message-passing (in fact, the algorithm is simpler for disconnected graphs since A_{bb} is diagonal). This corresponds to an optimal BCD update with a block size of $n/2$. We can generalize the red-black approach to arbitrary graphs by defining our blocks such that no two neighbours are in the same block (if our graph is very dense then the block sizes may need to be much smaller).[3] Red-black updating is very suitable for parallelization. However, its convergence rate may be very slow as the update does not capture any dependencies between nodes of the same colour. This makes it a slow choice for propagating information across the graph, especially when the labeled nodes are far from each other.

Message passing allows us to go beyond the red-black ordering scheme. Consider a tree-structured partition of the nodes as in Figure 1c, which spans a large number of dependent nodes. Unlike red-black blocks, these blocks include dependencies and allow the label information to spread more quickly through the graph. Further, since the blocks are forest-structured we can solve the Newton system associated with the blocks in linear time. Although we use a lattice-structured graph in Figure 1c, we can extend this idea to general graphs, where we could use any forest-structured partition of the nodes. However, the graph does require some level of sparsity if we want larger block sizes, as otherwise the forests may be very small (in the extreme case of a complete graph, forests can have at most 2 nodes). The actual size of the largest possible forest is related to the graph colouring problem [Esperet et al., 2015].

We can also use random and greedy approaches to construct large forest-structured blocks as in Figure 1d. Thus, the block size may vary at each iteration but restricting to forests still leads to a linear-time update. If we sample random forests, then the convergence rate under this strategy is covered by the arbitrary sampling theory [Zheng et al., 2014]. Also note that the maximum of the gradient norms over all forests defines a valid norm, so the analysis of Nutini et al. [2015] for the Gauss-Southwell (GS) greedy selection rules can be applied if we choose the forest with the largest gradient norm.

Unfortunately, computing the GS rule over forest-structured variable blocks is NP-hard, as we can reduce the 3-satisfiability problem to the problem of finding a maximum-weight forest [Garey and Johnson, 1979]. However, we can approximate the GS rule over the set of trees using the following heuristic:

1. Initialize b with the node i corresponding to the largest gradient, $|\nabla_i f(x)|$.
2. Search for the node i with the largest gradient that is not part of b and that maintains that b is a forest.
3. If such a node is found, add it to b and go back to step 2. Otherwise, stop.

Although this procedure does not yield the exact solution in general, it is appealing since (i) the procedure is efficient as it is easy to test whether adding a node maintains the forest property,[4] (ii) it outputs a forest so that the subsequent update is linear-time, (iii) we are guaranteed that the coordinate corresponding to the variable with the largest gradient is included in b (and it will often include many more of the largest gradient nodes), and (iv) we cannot add any additional node to the final forest and still maintain the forest property. Figure 1d shows an example of a forest-structured block selected in this way (it contains approximately $2n/3$ of the nodes).

[3] With fixed blocks, finding the minimum number of "colours" we need for a given graph is equivalent to the NP-hard graph colouring problem. However, there are various heuristics that quickly give a non-minimal valid colouring of the nodes.

[4] The time required to test the forest property is linear in the degree of the node if we employ two hash functions.

5 Experimental Results

We experimented with two label propagation problems:

- Dataset D: We used a 50 by 50 lattice structure with 100 labeled points, leading to a 2400 variable problem where each node has at most 4 neighbours in the graph.
- Dataset E: We used a 5-nearest neighbour graph based on 2000 samples from the 'two moons' dataset Zhou et al. [2004] and 100 labeled points. This results in a very sparse but unstructured graph.

We compared updating general blocks chosen using the greedy Gauss-Southwell rule (General), using the red-black ordering for the lattice-structured data (Red-Black), using the tree partition for the lattice-structured data (Tree Partitions), constructing a random tree (Random-Tree), or greedily constructing a tree (Greedy Tree). To make the iteration costs comparable, the General method uses a block size of $n^{1/3}$. Given the block, all methods thus compute the optimal update in $O(n)$ (we're ignoring the cost of choosing the block, but in general this should also be taken into account).

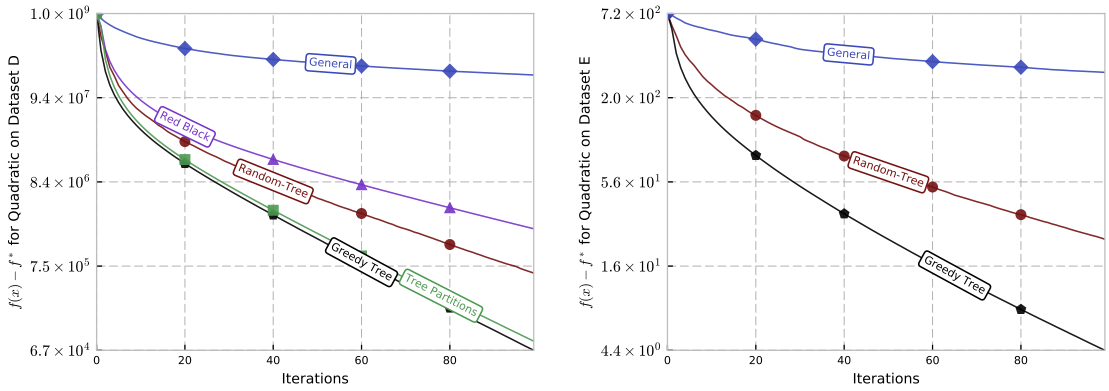


Figure 2: (Left) Dataset D; (Right) Dataset E.

We see in Figure 2 that tree-based methods perform substantially better than existing approaches that use unstructured blocks or a red-black ordering for the update. We see that the greedy tree-based approach offers a further advantage over using random trees. But even random trees perform much better than the greedy general approach which uses smaller blocks, indicating that updating large blocks may have more of an effect than cleverly choosing the blocks.

References

Y. Bengio, O. Delalleau, and N. Le Roux. Label propagation and quadratic criterion. *Semi-Supervised Learning*, pages 193–216, 2006.

D. Bickson. *Gaussian Belief Propagation: Theory and Application*. PhD thesis, Hebrew University of Jerusalem, 2009.

R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact Newton methods. *SIAM Journal on Numerical Analysis*, 19(2):400–408, 1982.

L. Esperet, L. Lemoine, and F. Maffray. Equitable partition of graphs into induced forests. *arXiv:1410.0861v3*, 2015.

K. Fountoulakis and R. Tappenden. A flexible coordinate descent method. *arXiv preprint arXiv:1507.03713*, 2015.

M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-completeness*. Freeman and Co, San Francisco, CA, 1979.

J. Nutini, M. Schmidt, I. H. Laradji, M. Friedlander, and H. Koepke. Coordinate descent converges faster with the Gauss-Southwell rule than random selection. *arXiv:1506.00552v1*, 2015.

S. Parter. The use of linear graphs in Gauss elimination. *SIAM Rev.*, 3(2):119–130, 1961.

M. Pilanci and M. J. Wainwright. Newton sketch: A linear-time optimization algorithm with linear-quadratic convergence. *arXiv preprint arXiv:1505.02250*, 2015.

Z. Qu, P. Richtárik, M. Takáč, and O. Feroq. SDNA: Stochastic dual coordinate ascent for empirical risk minimization. *arXiv:1502.02268v1*, 2015.

D. J. Rose. Triangulated graphs and the elimination process. *J. Math. Anal. Appl.*, 32(3):597–609, 1970.

O. Shental, P. H. Siegel, J. K. Wolf, D. Bickson, and D. Dolev. Gaussian belief propagation solver for systems of linear equations. *The 2008 IEEE International Symposium on Information Theory, Toronto*, 2008.

A. Srinivasan and E. Todorov. Graphical Newton. *arXiv preprint arXiv:1508.00952*, 2015.

Q. Zheng, P. Richtárik, and T. Zhang. Randomized dual coordinate ascent with arbitrary sampling. *arXiv:1411.5873*, 2014.

D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In *Advances in Neural Information Processing Systems*, pages 321–328, 2004.