Randomized Block Coordinate Methods for Large-Scale Optimization Problems

Subhrajeet Ghosh
Department of Information Science, The University of Arizona
subhrajeetghosh@arizona.edu

Isaac Leiterman
Department of Mining and Geological Engineering, The University of Arizona
isaacleiterman@arizona.edu

Abstract

Optimizing a system with a large number of variables presents significant computational challenges. Solving for optimality is generally memory-expensive and time-intensive when it comes to computing the full gradient and updating all the variables accordingly. In many instances, it is not feasible to utilize a full gradient method for optimization. Yet, as many important, real-world problems are characterized as large-scale, we require scalable optimization techniques that are feasible in this realm. Random Block Coordinate (RBC) methods provide a means of accomplishing large-scale optimization by computing partial gradients and updating only a subset of the variables. This reduces memory requirements and increases computational speed while still ensuring descent in the expectation. This paper will present RBC methods, showcase their uses in practice and within the literature, and compare against a full gradient approach.

keywords: Stochastic Optimization, Large-Scale Optimization, Random Block Coordinate Methods, Gradient Descent

1 Introduction

As technology progresses, datasets and models are becoming increasingly larger. In fields such as machine learning, network flow, and signal processing, it is commonplace to see problems with millions of variables. Due to memory and time constraints, full gradient methods are not feasible. Random Block Coordinate (RBC) methods are an alternative that is more computationally efficient. By computing and updating the variables on a subset rather than the full dataset, memory usage and cost per iteration decrease.

1.1 Applications

This section will outline some of the general applications of RBC methods.

Sparse models, like Lasso and Group Lasso, often have millions of features. RBC methods are apt optimizers in sparse machine learning because they scale well to large feature sets by only making updates on subsets of the features. Additionally, typical RBC methods will skip over zero-valued blocks and focus on nonzero blocks, which is beneficial considering the sparsity these models

are trying to capture. Because of RBC methods' scalability and limited updates, they are a good way of optimizing algorithms such as Lasso.

Factoring large matrices is a common operation in applications such as recommender systems, seismic inversion, and natural language processing. Similarly to sparse machine learning, these matrices are generally incomplete or sparse. For example, seismic inversion attempts to use a limited amount of seismic data from sensors to gather the full picture of the structures and composition of the subsurface. RBC methods can optimize this process by only selecting randomly from the available seismic sensor data to interpolate the entire underground.

The scope of this study doesn't allow for an exhaustive summary of all the applications of RBC methods, but we have highlighted some very important instances where they can be successful.

1.2 Literature review

This section will outline the literature review that this study performed.

Nesterov demonstrated that Random Coordinate (RC) methods could be used instead of full gradient on huge-scale problems [2]. RC is a subcategory of RBC methods where single variables are chosen, not blocks. Nesterov assumed convexity of the objective function and component-wise Lipschitz continuity. Convergence rates of convex functions were proven to be $O(1/k)$ and linear for strongly convex functions. Nesterov also introduced an accelerated RCD method, based on his full gradient acceleration method, that achieved a convergence rate of $O(1/k^2)$.

Singh et al. introduced an RBC projection algorithm for handling constrained optimization problems [3]. Their algorithm selects blocks, computes their partial gradient, and projects the update onto the feasible region. The authors assumed the constraint space was convex and that the gradient was Lipschitz continuous. They found their algorithm to converge at a rate of $O(1/k)$ for convex spaces and linear for strongly convex spaces.

Wang and Banerjee applied the RBC methodology in tandem with Stochastic Gradient (SG) to online learning [4]. Their algorithm works for composite functions, like Lasso, with a smooth, convex loss function and a non-smooth regularization term. They assumed block separability in the regularization term and that the composite function had block-wise Lipschitz continuous gradient. Their algorithm uniformly at random selects an f_i (SG) and a block (RBC). Updates were made with a proximal gradient step. They got a convergence rate of $O(1/\sqrt{k})$ for convex loss functions while strongly convex loss functions converged at a rate of $O(\log(k)/k)$.

Diakonikolas et al. introduced a simple "alternating" RBC method [1]. They partitioned their variable set into two distinct blocks. The authors assumed block gradients were Lipschitz continuous. The algorithm is not truly alternating since blocks are chosen at random, but the expectation in block choice is alternating. It was found to have a convergence rate of $O(1/k)$ for smooth convex functions. Additionally, the authors developed an acceleration variant that achieved a convergence rate of $O(1/k^2)$.

RBC methods were applied to a large-scale optimal transport problem by Zhao et al. [5]. Optimal transport was modeled as a linear programming (LP) framework which is different from the previous works. The algorithm works by randomly selecting a subset (block) of the transport matrix and then solves the smaller LP problem. Local redefinition of constraints was required after each update to maintain feasibility. Although not theoretically proven, the study's empirical results suggest a sublinear convergence rate.

As exhibited by the literature review, RBC methods are flexible and can fit into the optimization frameworks of many different disciplines.

2 Algorithm Descriptions

In this section, the general RBC method, along with the work in Section 1.2 will be presented in an algorithmic format.

Algorithm 1 General Random Block Coordinate Method

Initialization: $x_1 \in \mathbb{R}^n$, number of blocks m, step size $\alpha > 0$

for $k = 1$ to K **do**

$i_k \in \{1, \ldots, m\}$ uniformly at random

$x^{k+1}_{(i_k)} = x^k_{(i_k)} - \alpha_k \nabla_{(i_k)} f(x_k)$

$x^{k+1}_{\setminus(i_k)} = x^k_{\setminus(i_k)}$

end for

In Algorithm 1 we present the general RBC method. We assume a convex function f and block-wise Lipschitz continuity.

Algorithm 2 Random Coordinate Descent Method

Initialization: $x_1 \in \mathbb{R}^n$, Lipschitz constants L_i, probabilities p_i based on L_i and σ

for $k = 1$ to K **do**

$i_k \in \{1, \ldots, m\}$ with probability p_{i_k}

$x^{k+1}_{i_k} = x^k_{i_k} - \alpha \nabla_{i_k} f(x_k)$ where $\alpha = \frac{1}{L_{i_k}}$

$x^{k+1}_{\setminus i_k} = x^k_{\setminus i_k}$

end for

Algorithm 2 presents the work of [2]. Here, the probability of choosing a coordinate is based on the coordinate's Lipschitz constant and a tuning parameter σ. All assumptions are the same as presented in Section 1.2.

Algorithm 3 Random Block Coordinate Projection

Initialization: $x_1 \in \mathbb{R}^n$, constant step size $\alpha > 0$, probabilities p_i for $1, \ldots, n$

for $k = 1$ to K **do**

For $i = 1, \ldots, n$:

Sample $q_i^k \sim$ Bernoulli(p_i)

If $q_i^k = 1$: $\quad x_i^{k+1} = \Pi_{X_i}\left(x_i^k - \alpha \nabla_i f(x^k)\right)$

Else: $\quad x_i^{k+1} = x_i^k$

end for

Algorithm 3 is the one discussed in [3]. Here, we select probabilities for each block. A Bernoulli probability variable is determined based on the probability previously selected to determine if an update will happen. The update is then projected onto the feasible region X_i.

Algorithm 4 Online Random Block Coordinate

Initialization: $x_1 = 0$, step size $\alpha > 0$
for $k = 1$ to K **do**
  $i_k \sim \text{Uniform}\{1, \ldots, I\}$
  $j_k \sim \text{Uniform}\{1, \ldots, J\}$
  $x^{k+1}_{(j_k)} = \text{Prox}_{g_{(j_k)}}\left(x^k_{(j_k)} - \alpha_k \nabla_{(j_k)} f_{i_k}(x^k)\right)$
  $x^{k+1}_{\setminus(j_k)} = x^t_{\setminus(j_k)}$
end for

Above, Algorithm 4 references [4]. It showcases sampling an f_i and block j at each iteration. Proximal updates are made on the selection because this work assumes a composite function.

Algorithm 5 Alternating Randomized Block Coordinate

Initialization: $x_1 \in \mathbb{R}^n$, probabilities p_i for $i = 1, \ldots, n-1$ with $\sum p_i = 1$
for $k = 1$ to K **do**
  Sample $i_k \in \{1, \ldots, n-1\}$ with probability p_{i_k}
  $x^k_{(i_k)} = x^{k-1}_{(i_k)} - \alpha \nabla_{i_k} f(x_{k-1})$ where $\alpha = \frac{1}{L_{i_k}}$
  Minimize over block n: $x^k_{(n)} = \arg\min_z f(x^k_{(1)}, \ldots, x^k_{(n-1)}, z)$
end for

Algorithm 5 follows [1]. In each iteration, a block $i \in \{1, \ldots, n-1\}$ is selected with probability p_i, and a gradient step is performed on that block. The algorithm then minimizes the objective function over a special block $x_{(n)}$, holding all other blocks fixed. This alternating structure is designed to yield more effective updates by exploiting problem structure in block $x_{(n)}$.

Algorithm 6 Optimal Transport Random Block Coordinate

Initialization: Feasible $x_1 \in \mathbb{R}^n$, batch size l with $\text{rank}(A) + 1 \leq l \leq n$
for $k = 1$ to K
  $I_k \subset \{1, \ldots, n\}$ such that $|I_k| = l$
  $d_k \in \arg\min_d \left\{ c^\top d \;\middle|\; Ad = 0,\ x_k + d \geq 0,\ d_i = 0\ \forall i \notin I_k \right\}$
  $x_{k+1} = x_k + d_k$
end for

Algorithm 6 was presented in [5]. At each iteration, the algorithm selects a subset of coordinates, (I_k), with a fixed size l. The size of l guarantees feasible descent directions. The algorithm solves an LP subproblem over this working set with updated constraints to ensure feasibility. Note that updates are only made on the selected block as expressed by the definition of d_k.

3 Numerical Experiments

In this section, we present a numerical comparison experiment between Full-Gradient Descent (FGD) and a Random Block Coordinate (RBC) method. Our goal is to show that RBC's convergence subject to time, along with its time per iteration, is lower than FGD.

3.1 Problem Formulation

We aim to solve or get a close solution for the function $Ax = b$ where x is our variable vector, A is a sparse matrix, and b is the observation vector. To do this, we minimize the distance between Ax and b. One of the most commonly used functions for this type of problem is the least squares function which is given in Equation 1.

$$\frac{1}{2}||Ax - b||_2^2 \tag{1}$$

The least squares equation is used because it is statistically the best for linear regression. Additionally, it is smooth and convex, which makes it ideal for many optimization techniques.

The specific problem that we aim to solve is that of a sparse matrix with dimensions 20,000 rows by 20,000 columns and an observation vector with 20,000 entries. The A matrix was generated by using SciPy *sparse_rand* function, which generates a sparse matrix with randomly selected values. Our study decided to use a density of 0.01, which corresponds to every 100th entry having a non-zero value. The allowed random integer values for A were the range $[1, 100]$. The same range was used for the randomly selected integer values of the b vector.

3.2 Methods

The following will be an overview of both FGD and RBC methods and how they were adapted to apply to the problem we are aiming to solve.

3.2.1 Full Gradient Descent

This section will outline the use of the FGD method for optimizing our problem. Note that we compute the Lipschitz constant as the largest eigenvalue of A^TA. Since A is a large, sparse matrix, it is easier computationally to calculate the largest eigenvalue using sparse methods instead of computing $||A^TA||_2$. Algorithm 7 details the entirety of the process we are using.

Algorithm 7 Full Gradient Descent

Initialization: $x_0 = 0$, step size $\alpha = \frac{1}{L}$, $L = \lambda_{max}(A^TA)$
for $k = 0$ to 2,000 **do**
    $\nabla f_k = A^T(Ax_k - b)$
    $x_{k+1} = x_k - \alpha \nabla f_k$

3.2.2 Random Block Coordinate

The following is an overview of how this study applies RBC to our problem. Similarly to FGD, the Lipschitz constant is calculated as $\lambda_{max}(A_B^TA_B)$ where B denotes the block we randomly selected. It was deemed that a block size of 200 was appropriate for the problem. Note that the proposed algorithm computes a residual r. Since we are only updating a block, we update $Ax-b$ incrementally based on the residual. The residual allows us to avoid updating $Ax-b$ every time. The full proposed RBC algorithm is outlined in Algorithm 8.

Algorithm 8 Random Block Coordinate Method

Initialization: $x_0 = 0$, m blocks with size 200, step size $\alpha = \frac{1}{L_B}$, $r_0 = Ax_0 - b$
for $k = 1$ to 4,600 **do**
$i_k \in \{1, \ldots, m\}$ uniformly at random
$\nabla f_{(i)_k} = A_{(i)}^T r_k$
$L_{(i)} = \lambda_{max}(A_{(i)}^T A_{(i)})$
$\alpha = \frac{1}{L_i}$
$x_{(i)}^{k+1} = x_{(i)}^k - \alpha \nabla f_{(i)}^k$
$r_{k+1} = r_k - \alpha \nabla f_{(i)}^k$
end for

3.3 Results

Our results indicate that for large, sparse matrices, RBC methods are a better option over FGD. The RBC method was able to achieve lower objective function values sooner compared to FGD. This due to the lower computational cost that it has to perform at each iteration. A graphic showing this is depicted in Figure 1.

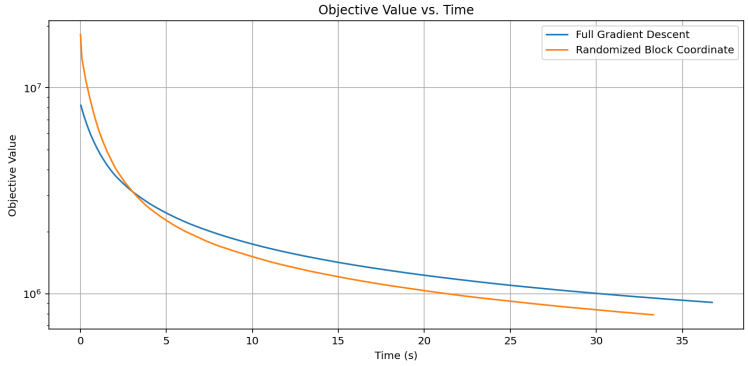

Figure 1: Objective function value vs. time to reach value for FGD and RBC

After roughly three seconds, the RBC method was producing lower values than FGD. This showcases its ability to produce lower values at a lower cost.

Additionally, RBC was shown to have a lower time per iteration than FGD. This is expected because RBC has to perform dramatically fewer computations and updates. Figure 2 shows the relationship between iteration run time and method.

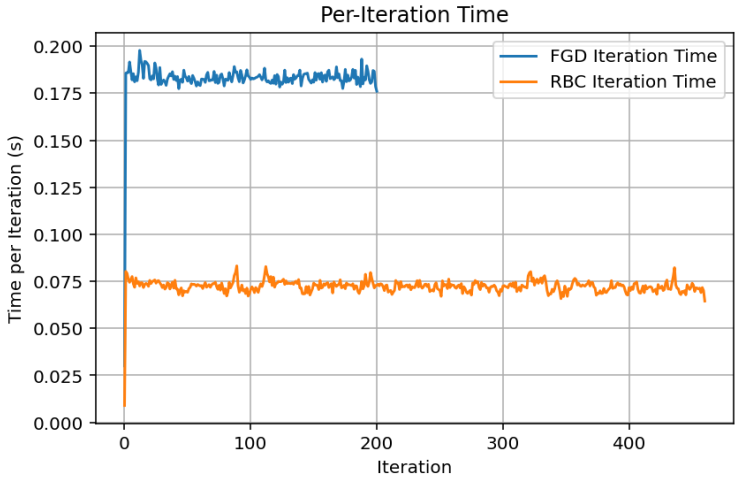

Figure 2: Iteration runtime for all iterations each method had to perform

Note that the methods end at different iteration times. These were the iteration times in which the total runtime of each method was roughly the same. Also note that sampling of times and objective values for both graphics occurred at every 10th iteration. The total number of iterations performed by FGD was 2000 in ∼37 seconds, while the total number of iterations performed by RBC was 4600 in ∼33 seconds.

4 Conclusion

The goal of this study was to present Random Block Coordinate methods and their use cases. They are applicable to high-dimensional and sparse problems. By updating a subset, block, of the solution at each iteration, it avoids excessive computations. As shown in Section 3.3, RBC was superior in a sparse environment with an extremely large number of variables (400,000,000). It excelled in both obtaining a lower objective function faster and performing lower-cost iterations. To conclude, RBC is a versatile and powerful optimization technique that could be applied to several real-world problems to improve efficiency.

5 Future Directions

There are several options of research into RBC methods. One direction involves exploring the parallelization of RBC. Variables can be partitioned into disjoint blocks. In a parallel computing setup, each thread can independently compute the partial gradient for its assigned block and perform the corresponding update.

The application of block-coordinate methods to deep learning is another option. Since neural network parameters are often grouped (layers or modules), these methods may allow for selective or asynchronous updates, which could improve convergence in large-scale settings.

Finally, a route could be extending the RBC framework to distributed environments, such as multi-GPU systems or cloud-based clusters. Efficient implementations in these settings would allow the method to scale to high-dimensional models and large datasets, improving its applicability.

References

[1] J. Diakonikolas and L. Orecchia, *Alternating randomized block coordinate descent*, in Proceedings of the 35th International Conference on Machine Learning, vol. 80, PMLR, 2018, pp. 1198–1207.

[2] Y. Nesterov, *Efficiency of coordinate descent methods on huge-scale optimization problems*, SIAM Journal on Optimization, 22 (2012), pp. 341–362.

[3] C. Singh, A. Nedić, and R. Srikant, *Random block-coordinate gradient projection algorithms*, in 53rd IEEE Conference on Decision and Control, IEEE, 2014, pp. 185–190.

[4] H. Wang and A. Banerjee, *Randomized block coordinate descent for online and stochastic optimization*, arXiv preprint arXiv:1407.0107, (2014).

[5] Y. Xie, Z. Wang, and Z. Zhang, *Randomized methods for computing optimal transport without regularization and their convergence analysis*, arXiv preprint arXiv:2212.07046, (2023).