Large-scale randomized-coordinate descent methods with non-separable linear constraints

Sashank J. Reddi [*] **Ahmed Hefny**[*] **Carlton Downey** **Avinava Dubey**
Machine Learning Department
Carnegie Mellon University

Suvrit Sra [†]
Massachusetts Institute of Technology

Abstract

We develop randomized block coordinate descent (CD) methods for linearly constrained convex optimization. Unlike other large-scale CD methods, we do not assume the constraints to be separable, but allow them be coupled linearly. To our knowledge, ours is the first CD method that allows linear coupling constraints, without making the global iteration complexity have an *exponential dependence* on the number of constraints. We present algorithms and theoretical analysis for four key (convex) scenarios: (i) smooth; (ii) smooth + separable nonsmooth; (iii) asynchronous parallel; and (iv) stochastic. We discuss some architectural details of our methods and present preliminary results to illustrate the behavior of our algorithms.

1 INTRODUCTION

Coordinate descent (CD) methods are conceptually among the simplest schemes for unconstrained optimization—they have been studied for a long time (see e.g., [1, 4, 28]), and are now enjoying greatly renewed interest. Their resurgence is rooted in successful applications in machine learning [15, 16], statistics [8, 17], and many other areas—see [31, 32, 35] and references therein for more examples.

A catalyst to the theoretical as well as practical success of CD methods has been randomization. (The idea of randomized algorithms for optimization methods is of course much older, see e.g., [29].) Indeed, generic non-randomized CD has resisted complexity analysis, though there is promising recent work [14, 34, 40]; remarkably for randomized CD for smooth convex optimization, Nesterov [25, 26] presented an analysis of global iteration complexity. This work triggered several improvements, such as [32, 33], who simplified and extended the analysis to include separable nonsmooth terms. Randomization has also been crucial to a host of other CD algorithms and analyses [5, 16, 20, 23, 30, 31, 33, 35–37].

Almost all of the aforementioned CD methods assume essentially unconstrained problems, which at best allow separable constraints. In contrast, we develop, analyze, and implement randomized CD methods for the following composite objective convex problem with *non-separable* linear constraints

$$\min_x F(x) := f(x) + h(x) \quad \text{s.t.} \quad Ax = 0. \qquad (1)$$

Here, $f : \mathbb{R}^n \to \mathbb{R}$ is assumed to be continuously differentiable and convex, while $h : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is lower semi-continuous, convex, coordinate-wise separable, but not necessarily smooth; the linear constraints (LC) are specified by a matrix $A \in \mathbb{R}^{m\times n}$, for which $m \ll n$, and a certain structure (see §4) is assumed. The reader may wonder whether one cannot simply rewrite Problem (1) in the form of $f + h$ (without additional constraints) using suitable indicator functions. However, the resulting regularized problem then *no longer* fits the known efficient CD frameworks [32], since the nonsmooth part is *not* block-separable.

Problem (1) subsumes the usual regularized optimization problems pervasive in machine learning for the simplest ($m = 0$) case. In the presence of linear constraints ($m > 0$), Problem (1) assumes a form used in the classic Alternating Direction Method of Multipliers (ADMM) [9, 10]. The principal difference between our approach and ADMM is that the latter treats the entire variable $x \in \mathbb{R}^n$ as a single block, whereas we use the structure of A to split x into b smaller blocks. Familiar special cases of Problem (1) include SVM (with bias) dual, fused Lasso and group Lasso [38], and linearly constrained least-squares regression [11, 19].

Recently, Necoara et al. [23] studied a special case of Problem (1) that sets $h \equiv 0$ and assumes a single sum constraint. They presented a randomized CD method that starts with a feasible solution and at each iteration updates a pair

[*] Indicates equal contribution.

[†] A part of this work was performed when the author was at Carnegie Mellon University

of coordinates to ensure descent on the objective while maintaining feasibility. This scheme is reminiscent of the well-known SMO procedure for SVM optimization [27]. For smooth convex problems with n variables, Necoara et al. [23] prove an $O(1/\epsilon)$ rate of convergence. More recently, in [22] considered a generalization to the general case $Ax = 0$ (assuming h is coordinatewise separable).

Unfortunately, the analysis in [22] yields an extremely pessimistic complexity result:

Theorem 1 ([22])**.** *Consider Problem* (1) *with h being coordinatewise separable, and $A \in \mathbb{R}^{m\times n}$ with b blocks. Then, the CD algorithm in [22] takes no more than $O(b^m/\epsilon)$ iterations to obtain a solution of ϵ-accuracy.*

This result is exponential in the number of constraints and too severe even for small-scale problems!

We present randomized CD methods, and prove that for important special cases (mainly $h \equiv 0$ *or* A is a sum constraint) we can obtain global iteration complexity that *does not have* an intractable dependence on either the number of coordinate blocks (b), or on the number of linear constraints (m). Previously, Tseng and Yun [39] also studied a linearly coupled block-CD method based on the Gauss-Southwell choice; however, their complexity analysis applies only to the special $m = 0$ and $m = 1$ cases.

To our knowledge, ours is the first work on CD for problems with more than one ($m > 1$) linear constraints that presents such results.

Contributions. In light of the above background, the primary contributions of this paper are as follows:

- Convergence rate analysis of a randomized block-CD method for the smooth case ($h \equiv 0$) with $m \geq 1$ general linear constraints.
- A tighter convergence analysis for the composite function optimization ($h \neq 0$) than [22] in the case of sum constraint.
- An asynchronous CD algorithm for Problem (1).
- A stochastic CD method with convergence analysis for solving problems with a separable loss $f(x) = (1/N)\sum_{i=1}^{N} f_i(x)$.

Table 1 summarizes our contributions and compares it with existing state-of-the-art coordinate descent methods. The detailed proofs of all our theoretical claims are available in the appendix.

Additional related work. As noted, CD methods have a long history in optimization and they have gained tremendous recent interest. We cannot hope to do full justice to all the related work, but refer the reader to [32, 33] and [20] for more thorough coverage. Classically, local linear convergence was analyzed in [21]. Global rates for randomized block coordinate descent (BCD) were pioneered by Nesterov [25], and have since then been extended by various authors [2, 32, 33, 40]. The related family of Gauss-Seidel like analyses for ADMM have also recently gained prominence [13]. A combination of randomized block-coordinate ideas with Frank-Wolfe methods was recently presented in [18], though algorithmically the Frank-Wolfe approach is very different as it relies on non projection based oracles.

Paper	LC	Prox	Parallel	Stochastic
[23]	YES	$\times$	$\times$	$\times$
[39]	YES	YES	$\times$	$\times$
[22]	YES	YES	$\times$	$\times$
[7]	$\times$	YES	YES	$\times$
[5]	$\times$	ℓ_1	YES	YES
[33]	$\times$	YES	YES	$\times$
Ours	**YES**	**YES**	**YES**	**YES**

Table 1: Summary comparison of our method with other CD methods; LC denotes 'linear constraints'; Prox signifies an extension using proximal operators (to handle $h \neq 0$).

2 PRELIMINARIES

In this section, we further explain our model and assumptions. We assume that the entire space $\mathbb{R}^n$ is decomposed into b *blocks*, i.e., $x = [x_1^\top, \cdots, x_b^\top]^\top$ where $x \in \mathbb{R}^n$, $x_i \in \mathbb{R}^{n_i}$ for all $i \in [b]$, and $n = \sum_i n_i$. For any $x \in \mathbb{R}^n$, we use x_i to denote the i^{th} block of x. We model communication constraints in our algorithms by viewing variables as nodes in a connected graph $G := (V, E)$. Specifically, node $i \in V \equiv [b]$ corresponds to variable x_i, while an edge $(i, j) \in E \subset V \times V$ is present if nodes i and j can exchange information. We use "pair" and "edge" interchangeably.

For a differentiable function f, we use $f_{i_1\cdots i_p}$ and $\nabla_{i_1\cdots i_p} f(x)$ (or $\nabla_{x_{i_1}\cdots x_{i_p}} f(x)$) to denote the restriction of the function and its partial gradient to coordinate blocks $(x_{i_1}, \cdots, x_{i_p})$. For any matrix B with n columns, we use B_i to denote the columns of B corresponding to x_i and B_{ij} to denote the columns of B corresponding to x_i and x_j. We use U to denote the $n \times n$ identity matrix and hence U_i is a matrix that places an n_i dimensional vector into the corresponding block of an n dimensional vector.

We make the following standard assumption on the partial gradients of f.

Assumption 1. The function has block-coordinate Lipschitz continuous gradient, i.e.,

$$\|\nabla_i f(x) - \nabla_i f(x + U_i h)\| \leq L_i \|h_i\| \text{ for all } x \in \mathbb{R}^n, .$$

Assumption 1 is similar to the typical Lipschitz continuous gradients assumed in first-order methods and it is necessary to ensure convergence of block-coordinate methods.

When functions f_i and f_j have Lipschitz continuous gradients with constants L_i and L_j respectively, one can show that the function f_{ij} has a Lipschitz continuous gradient with $L_{ij} = L_i + L_j$ [22; Lemma 1]. The following result is standard.

Lemma 2. *For any function $g : \mathbb{R}^n \to \mathbb{R}$ with L-Lipschitz continuous gradient ∇g, we have*

$$g(x) \le g(y) + \langle \nabla g(y), x - y\rangle + \tfrac{L}{2}\|x - y\|^2 \; x, y \in \mathbb{R}^n.$$

Following [32, 39], we also make the following assumption on the structure of h.

Assumption 2. The nonsmooth function h is block separable, i.e., $h(x) = \sum_i h_i(x_i)$.

This assumption is critical to composite optimization using CD methods. We also assume access to an oracle that returns function values and *partial gradients* at any points and iterates of the optimization algorithm.

3 ALGORITHM

We are now ready to present our randomized CD methods for Problem (1) in various settings. We first study composite minimization (§3.1) and later look at asynchronous (§3.2) and stochastic (§3.3) variants. The main idea underlying our algorithms is to pick a random pair $(i, j) \in E$ of variables (blocks) at each iteration, and to update them in a manner which maintains feasibility and ensures progress in optimization.

3.1 Composite Minimization

We begin with the nonsmooth setting, where $h \not\equiv 0$. We start with a feasible point x^0. Then, at each iteration we pick a random pair $(i, j) \in E$ of variables and minimize the first-order Taylor expansion of the loss f around the current iterate while maintaining feasibility. Formally, this involves performing the update

$$\begin{aligned} Z(f, x, (i, j), \alpha) := \underset{A_{ij}d_{ij}=0}{\arg\min}\, & f(x) + \langle \nabla_{ij} f(x), d_{ij}\rangle \\ & + (2\alpha)^{-1}\|d_{ij}\|^2 + h(x + U_{ij}d_{ij}), \end{aligned} \tag{2}$$

where $\alpha > 0$ is a stepsize parameter and d_{ij} is the update. The right hand side of Equation (2) upper bounds f at $x + U_{ij}d_{ij}$, as seen by using Assumption 1 and Lemma 2. If $h(x) \equiv 0$, minimizing Equation (2) yields

$$\begin{aligned} \lambda &\leftarrow \alpha(A_iA_i^\top + A_jA_j^\top)^+ (A_i\nabla_i f(x) + A_j\nabla_j f(x)) \\ d_i &\leftarrow -\alpha\nabla_i f(x) + A_i^\top \lambda \\ d_j &\leftarrow -\alpha\nabla_j f(x) + A_j^\top \lambda \end{aligned} \tag{3}$$

Algorithm 1 presents the resulting method.

Note that since we start with a feasible point x^0 and the update d^k satisfies $Ad^k = 0$, the iterate x^k is always feasible. However, it can be shown that a necessary condition for Equation (2) to result in a non-zero update is that A_i and A_j span the same column space. If the constraints are not block separable (i.e. for any partitioning of blocks $x_1, \dots, x_b$ into two groups, there is a constraint that involves blocks from both groups), a typical way to satisfy the aforementioned condition is to require A_i to be full row-rank for all $i \in [b]$. This constraints the minimum block size to be chosen in order to apply randomized CD.

Theorem 3 describes convergence of Algorithm 1 for the smooth case ($h \equiv 0$), while Theorem 6 considers the nonsmooth case under a suitable assumption on the structure of the interdependency graph G—both results are presented in Section 4.

```
1: $x^0 \in \mathbb{R}^n$ such that $Ax^0 = 0$
2: for $k \ge 0$ do
3:     Select a random edge $(i_k, j_k) \in E$ with probability $p_{i_kj_k}$
4:     $d^k \leftarrow U_{i_kj_k} Z(f, x^k, (i_k, j_k), \alpha_k/L_{i_kj_k})$
5:     $x^{k+1} \leftarrow x^k + d^k$
6:     $k \leftarrow k + 1$
7: end for
```

Algorithm 1: Composite Minimization with Linear Constraints

3.2 Asynchronous Parallel Algorithm for Smooth Minimization

Although the algorithm described in the previous section solves a simple subproblem at each iteration, it is inherently sequential. This can be a disadvantage when addressing large-scale problems. To overcome this concern, we develop an asynchronous parallel method that solves Problem (1) for the smooth case.

Our parallel algorithm is similar to Algorithm 1, except for a crucial difference: now we may have multiple processors, and each of these executes the loop 2–6 *independently* without the need for coordination. This way, we can solve subproblems (i.e., multiple pairs) simultaneously in parallel, and due to the asynchronous nature of our algorithm, we can execute updates as they complete, without requiring any locking.

The critical issue, however, with implementing an asynchronous algorithm in the presence of non-separable constraints is ensuring feasibility throughout the course of the algorithm. This requires the operation $x_i \leftarrow x_i + \delta$ to be executed in an *atomic* (i.e., sequentially consistent) fashion. Modern processors facilitate that without an additional locking structure through the "compare-and-swap" instruction [30]. Since the updates use atomic increments and each update satisfies $Ad^k = 0$, the net effect of T updates is $\sum_{k=1}^{T} Ad^k = 0$, which is feasible despite asynchronicity of the algorithm.

The next key issue is that of convergence. In an asynchronous setting, the updates are based on *stale* gradients that are computed using values of x read many iterations earlier. But provided that gradient staleness is bounded, we can establish a sublinear convergence rate of the asynchronous parallel algorithm (Theorem 4). More formally, we assume that in iteration k, *stale* gradients are computed based on $x^{D(k)}$ such that $k - D(k) \leq \tau$. The bound on staleness, denoted by τ, captures the degree of parallelism in the method: such parameters are typical in asynchronous systems and provides a bound on the delay of the updates [20].

Before concluding the discussion on our asynchronous algorithm, it is important to note the difficulty of extending our algorithm to nonsmooth problems. For example, consider the case where $h = \mathbb{I}_C$ (indicator function of some convex set). Although a pairwise update as suggested above maintains feasibility with respect to the linear constraint $Ax = 0$, it may violate the feasibility of being in the convex set C. This complication can be circumvented by using a convex combination of the current iterate with the update, as this would retain overall feasibility. However, it would complicate the convergence analysis. We plan to investigate this direction in future work.

3.3 Stochastic Minimization

An important subclass of Problem (1) assumes separable losses $f(x) = \frac{1}{N}\sum_{i=1}^{N} f_i(x)$. This class arises naturally in many machine learning applications where the loss separates over training examples. To take advantage of this added separability of f, we can derive a stochastic block-CD procedure.

Our key innovation here is the following: in addition to randomly picking an edge (i, j), we also pick a function randomly from $\{f_1, \cdots, f_N\}$ and perform our update using this function. This choice substantially reduces the cost of each iteration when N is large, since now the gradient calculations involve only the randomly selected function f_i (i.e., we now use a stochastic-gradient). Pseudocode is given in Algorithm 2.

```
1: Choose $x^0 \in \mathbb{R}^n$ such that $Ax^0 = 0$.
2: for $k \geq 0$ do
3:     Select a random edge $(i_k, j_k) \in E$ with probability $p_{i_k j_k}$
4:     Select random integer $l \in [N]$
5:     $x^{k+1} \leftarrow x^k + U_{i_k j_k} Z(f_l, x^k, (i_k, j_k), \alpha_k / L_{i_k j_k})$
6:     $k \leftarrow k + 1$
7: end for
```

Algorithm 2: Stochastic Minimization with Linear Constraints

Notice that the per iteration cost of Algorithm 2 is lower than Algorithm 1 by a factor of N. However, as we will see later, this speedup comes at a price of slower convergence rate (Theorem 5). Moreover, to ensure convergence, decaying step sizes $\{\alpha_k\}_{k\geq 0}$ are generally chosen.

4 CONVERGENCE ANALYSIS

In this section, we outline convergence results for the algorithms described above. The proofs are somewhat technical, and hence left in the appendix due to lack of space; here we present only the key ideas.

For simplicity, we present our analysis for the following reformulation of the main problem:

$$\min_{y,z} \quad f(y,z) + \sum\nolimits_{i=1}^{b} h(y_i, z_i) \qquad (4)$$
$$\text{subject to} \quad \sum\nolimits_{i=1}^{b} y_i = 0,$$

where $y_i \in \mathbb{R}^{n_y}$ and $z_i \in \mathbb{R}^{n_z}$. Let $y = [y_1^\top \cdots y_b^\top]^\top$ and $z = [z_1^\top \cdots z_b^\top]^\top$. We use x to denote the concatenated vector $[y^\top z^\top]^\top$ and hence we assume (unless otherwise mentioned) that the constraint matrix A is defined as follows

$$A \begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{b} y_i \\ 0 \end{pmatrix}. \qquad (5)$$

It is worth emphasizing that this analysis *does not* result in any loss of generality. This is due to the fact that Problem (1) with a general constraint matrix $\tilde{A}$ having full row-rank submatrices $\tilde{A}_i$'s can be rewritten in the form of Problem (4) by using the transformation specified in Section E of the appendix. It is important to note that this reduction is presented only for the ease of exposition. For our experiments, we directly solve the problem in Equation 2.

Let $\eta_k = \{(i_0, j_0), \ldots, (i_{k-1}, j_{k-1})\}$ denote the pairs selected up to iteration $k-1$. To simplify notation, assume (without loss of generality) that the Lipschitz constant for the partial gradient $\nabla_i f(x)$ and $\nabla_{ij} f(x)$ is L for all $i \in [n]$ and $(i,j) \in E$.

Similar to [23], we introduce a Laplacian matrix $\mathcal{L} \in \mathbb{R}^{b\times b}$ that represents the communication graph G. Since we also have unconstrained variables z_i, we introduce a diagonal matrix $\mathcal{D} \in \mathbb{R}^{b\times b}$.

$$\mathcal{L}_{ij} = \begin{cases} \sum_{r\neq i} \frac{p_{ir}}{2L} & i = j \\ -\frac{p_{ij}}{2L} & i \neq j \end{cases} \qquad \mathcal{D}_{ij} = \begin{cases} \frac{p_i}{L} & i = j \\ 0 & i \neq j \end{cases}$$

We use $\mathcal{K}$ to denote the concatenation of the Laplacian $\mathcal{L}$ and the diagonal matrix $\mathcal{D}$. More formally,

$$\mathcal{K} = \begin{bmatrix} \mathcal{L} \otimes I_{n_y} & 0 \\ 0 & \mathcal{D} \otimes I_{n_z} \end{bmatrix}.$$

This matrix induces a norm $\|x\|_{\mathcal{K}} = \sqrt{x^\top \mathcal{K} x}$ on the *feasible subspace*, with a corresponding dual norm

$$\|x\|_{\mathcal{K}}^{*} = \sqrt{x^\top \left(\begin{bmatrix} \mathcal{L}^{+} \otimes I_{n_y} & 0 \\ 0 & \mathcal{D}^{-1} \otimes I_{n_z} \end{bmatrix} \right) x}$$

Let X^* denote the set of optimal solutions and let x^0 denote the initial point. We define the following distance, which quantifies how far the initial point is from the optimal, taking into account the graph layout and edge selection probabilities

$$R(x^0) := \max_{x: f(x) \le f(x^0)} \max_{x^* \in X^*} \|x - x^*\|_{\mathcal{K}}^* \tag{6}$$

Note. Before delving into the details of the convergence results, we would like to draw the reader's attention to the impact of the communication network G on convergence. In general, the convergence results depend on $R(x^0)$, which in turn depends on the Laplacian $\mathcal{L}$ of the graph G. As a rule of thumb, the larger the connectivity of the graph, the smaller the value of $R(x^0)$, and hence, faster the convergence.

4.1 Convergence results for the smooth case

We first consider the case when $h = 0$. Here the subproblem at k^{th} iteration has a very simple update $d_{i_k j_k} = U_{i_k} d^k - U_{j_k} d^k$ where $d^k = \frac{\alpha_k}{2L}(\nabla_{j_k} f(x^k) - \nabla_{i_k} f(x^k))$. We now prove that Algorithm 1 attains an $O(1/k)$ convergence rate.

Theorem 3. *Let $\alpha_k = 1$ for $k \ge 0$, and let $\{x^k\}_{k \ge 0}$ be the sequence generated by Algorithm 1; let f^* denote the optimal value. Then, we have the following rate of convergence:*

$$\mathbb{E}[f(x^k)] - f^* \le \frac{2R^2(x^0)}{k}$$

where $R(x^0)$ is as defined in Equation 6.

Proof Sketch. We first prove that each iteration leads to descent in expectation. More formally, we get

$$\mathbb{E}_{i_k j_k}[f(x^{k+1}) | \eta_k] \le f(x^k) - \tfrac{1}{2} \nabla f(x^k)^\top \mathcal{K} \nabla f(x^k).$$

The above step can be proved using Lemma 2. Let $\Delta_k = \mathbb{E}[f(x^k)] - f^*$. It can be proved that

$$\frac{1}{\Delta_k} \le \frac{1}{\Delta_{k+1}} - \frac{1}{2R^2(x^0)}$$

This follows from the fact that

$$\begin{aligned} f(x^{k+1}) - f^* &\le \|x^k - x^*\|_{\mathcal{K}}^* \|\nabla f(x^k)\|_{\mathcal{K}} \\ &\le R(x^0) \|\nabla f(x^k)\|_{\mathcal{K}} \quad \forall k \ge 0 \end{aligned}$$

Telescoping the sum, we get the desired result. □

Note that Theorem 3 is a strict generalization of the analysis in [23] and [22] due to: (i) the presence of unconstrained variables z; and (ii) the presence of a non-decomposable objective function. it is also worth emphasizing that our convergence rates improve upon those of [22], since they do not involve an exponential dependence of the form b^m on the number of constraints.

We now turn our attention towards the convergence analysis of our asynchronous algorithm under a consistent reading model [20]. In this context we would like to emphasize that while our theoretical analysis assumes consistent reads, we do not enforce this assumption in our experiments.

Theorem 4. *Let $\rho > 1$ and $\alpha_k = \alpha$ be such that $\alpha < 2/(1 + \tau + \tau\rho^\tau)$ and $\alpha < (\rho - 1)/(\sqrt{2}(\tau + 2)(\rho^{\tau+1} + \rho))$. Let $\{x_k\}_{k \ge 0}$ be the sequence generated by asynchronous algorithm using step size α_k and let f^* denote the optimal value. Then, we have the following rate of convergence for the expected values of the objective function*

$$\mathbb{E}[f(x_k)] - f^* \le \frac{R^2(x^0)}{\mu k}$$

where $R(x^0)$ is as defined in Equation 6 and $\mu = \frac{\alpha_k^2}{2}\left(\frac{1}{\alpha_k} - \frac{1 + \tau + \tau\rho^\tau}{2}\right)$.

Proof Sketch. For ease of exposition, we describe the case where the unconstrained variables z are absent. The analysis of case with z variables can be carried out in a similar manner. Let $D(k)$ denote the iterate of the variables used in the k^{th} iteration (the existence of $D(k)$ follows from the consistent reading assumption). Let $d^k = \frac{\alpha_k}{2L}\left(\nabla_{y_{j_k}} f(x^{D(k)}) - \nabla_{y_{i_k}} f(x^{D(k)})\right)$ and $d^k_{i_k j_k} = x^{k+1} - x^k = U_{i_k} d^k - U_{j_k} d^k$. Using Lemma 2 and the assumption that staleness in the variables is bounded by τ, i.e., $k - D(k) \le \tau$ and definition of d^k_{ij}, we can derive the following bound:

$$\begin{aligned} \mathbb{E}[f(x^{k+1})] \le \mathbb{E}[f(x^k)] - L\left(\frac{1}{\alpha_k} - \frac{1+\tau}{2}\right) \mathbb{E}[\|d^k_{i_k j_k}\|^2] \\ + \frac{L}{2} \mathbb{E}\left[\sum_{t=1}^{\tau} \|d^{k-t}_{i_{k-t} j_{k-t}}\|^2\right]. \end{aligned}$$

In order to obtain an upper bound on the norms of $d^k_{i_k j_k}$, we prove that

$$\mathbb{E}\left[\|d^{k-1}_{i_{k-1} j_{k-1}}\|^2\right] \le \rho \mathbb{E}\left[\|d^k_{i_k j_k}\|^2\right]$$

This can proven using mathematical induction. Using the above bound on $\|d^k_{i_k j_k}\|^2$, we get

$$\begin{aligned} &\mathbb{E}[f(x^{k+1})] \le \\ &\quad \mathbb{E}[f(x^k)] - L\left(\frac{1}{\alpha_k} - \frac{1 + \tau + \tau\rho^\tau}{2}\right) \mathbb{E}[\|d^k_{i_k j_k}\|^2] \end{aligned}$$

This proves that the method is a descent method in expectation. Following similar analysis as Theorem 3, we get the required result. □

Note the dependence of convergence rate on the staleness bound τ. For larger values of τ, the stepsize α_k needs to be decreased to ensure convergence, which in turn slows down the convergence rate of the algorithm. Nevertheless, the convergence rate remains $O(1/k)$.

The last smooth case we analyze is our stochastic algorithm.

Theorem 5. *Let $\alpha_i = \sqrt{\Delta_0 L}/(M\sqrt{i+1})$ for $i \geq 0$ in Algorithm 2. Let $\{x^k\}_{k\geq 0}$ be the sequence generated by Algorithm 2 and let f^* denote the optimal value. We denote $\bar{x}^k = \arg\min_{0\leq i\leq k} f(x^k)$. Then, we have the following rate of convergence for the expected values of the objective:*

$$\mathbb{E}[f(\bar{x}^k)] - f^* \leq O\left(\frac{1}{\sqrt[4]{k}}\right)$$

where $\Delta_0 = f(x^0) - f^*$.

The convergence rate is $O(1/k^{1/4})$ as opposed to $O(1/k)$ of Theorem 3. On the other hand, the iteration complexity is lower by a factor of N; this kind of tradeoff is typical in stochastic algorithms, where the slower rate is the price we pay for a lower iteration complexity. We believe that the convergence rate can be improved to $O(1/\sqrt{k})$, the rate generally observed in stochastic algorithms, by a more careful analysis.

4.2 Nonsmooth case

We finally state the convergence rate for the nonsmooth case ($h \not\equiv 0$) in the case of a sum constraint. Similar to [22], we assume h is coordinatewise separable (i.e. we can write $h(x) = \sum_{i=1}^{b}\sum_j x_{ij}$), where x_{ij} is the j^{th} coordinate in the i^{th} block. For this analysis, we assume that the graph G is a clique [1] with uniform probability, i.e., $\lambda = p_{ij} = 2/b(b-1)$.

Theorem 6. *Assume $Ax = \sum_i A_i x_i$. Let $\{x^k\}_{k\geq 0}$ be the sequence generated by Algorithm 1 and let F^* denote the optimal value. Assume that the graph G is a clique with uniform probability. Then we have the following:*

$$\mathbb{E}[F(x^k) - F^*] \leq \frac{b^2 L R^2(x^0)}{2k + \frac{b^2 L R^2(x^0)}{\Delta_0}},$$

where $R(x^0)$ is as defined in Equation 6.

This convergence rate is a generalization of the convergence rate obtained in Necoara and Patrascu [22] for a single linear constraint (see Theorem 1 in [22]). It is also an improvement of the rate obtained in Necoara and Patrascu [22] for general linear constraints (see Theorem 4 in [22]) when applied to the special case of a sum constraint. Our improvement comes in the form of a tractable constant, as opposed to the exponential dependence $O(b^m)$ shown in [22].

[1]We believe our results also easily extend to the general case along the lines of [31–33], using the concept of *Expected Separable Overapproximation* (ESO). Moreover, the assumption is not totally impractical, e.g., in a multicore setting with a zero-sum constraint (i.e. $A_i = I$), the clique-assumption introduces little cost.

5 APPLICATIONS

To gain a better understanding of our approach, we state some applications of interest, while discussing details of Algorithm 1 and Algorithm 2 for them. While there are many applications of problem (1), due to lack of space we only mention a few prominent ones here.

Support Vector Machines: The SVM dual (with bias term) assumes the form (1); specifically,

$$\min_{\alpha} \tfrac{1}{2}\sum\nolimits_{i,j} \alpha_i\alpha_j y_i y_j z_i^\top z_j - \sum\nolimits_{i=1}^{n} \alpha_i$$
$$s.t. \sum\nolimits_i \alpha_i y_i = 0,\ \ 0 \leq \alpha_i \leq C \quad \forall\, i \in [n]. \tag{7}$$

Here, z_i denotes the feature vector of the i^{th} training example and $y_i \in \{1, -1\}$ denotes the corresponding label. By letting $f(\alpha) = \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j z_i^\top z_j - \sum_i \alpha_i$ and $h(\alpha) = \sum_i \mathbb{I}(0 \leq \alpha_i \leq C)$ and $A = [y_1, \ldots, y_n]$ this problem can be written in form of Problem (1). Using Algorithm 1 for SVM involves solving a sub-problem similar to one used in SMO in the scalar case (i.e., $\alpha_i \in \mathbb{R}$) and can be solved in linear time in the block case (see [3]).

Generalized Lasso: The objective is to solve the following optimization problem.

$$\min_\beta \quad \tfrac{1}{2}\|Y - X\beta\|_2^2 + \lambda\|D\beta\|_1$$

where $Y \in \mathbb{R}^N$ denotes the output, $X \in \mathbb{R}^{N\times n}$ is the input and $D \in \mathbb{R}^{q\times n}$ represents a specified penalty matrix. This problem can also be seen as a specific case of Problem (1) by introducing an auxiliary variable t and slack variables u, v. Then, $f(\beta, t) = \frac{1}{2}\|Y - X\beta\|_2^2 + \sum_i t_i$, $h(u, v) = \mathbb{I}(u \geq 0) + \mathbb{I}(v \geq 0)$ and, $t - D\beta - u = 0$ and $t + D\beta - v = 0$ are the linear constraints. To solve this problem, we can use either Algorithm 1 or Algorithm 2. In general, optimization of convex functions on a structured convex polytope can be solved in a similar manner.

Unconstrained Separable Optimization: Another interesting application is for unconstrained separable optimization. For any problem $\min_x \sum_i f_i(x)$—a form generally encountered across machine learning—can be rewritten using variable-splitting as $\min_{\{x_i = x, \forall i\in[N]\}} f_i(x_i)$. Solving the problem in distributed environment requires considerable synchronization (for the consensus constraint), which can slow down the algorithm significantly. However, the dual of the problem is

$$\min_{\lambda} \sum_i f_i^*(\lambda_i) \quad s.t \ \ \sum\nolimits_{i=1}^{N} \lambda_i = 0.$$

where f_i^* is the Fenchel conjugate of f_i. This reformulation perfectly fits our framework and can be solved in an asynchronous manner using the procedure described in Section 3.2.

Other interesting application include constrained least square problem, multi-agent planning problems, resource allocation—see [22, 23] and references therein for more examples.

6 EXPERIMENTS

In this section, we present our empirical results. In particular, we examine the behavior of random coordinate descent algorithms analyzed in this paper under different communication constraints and concurrency conditions. [2]

6.1 Effect of Communication Constraints

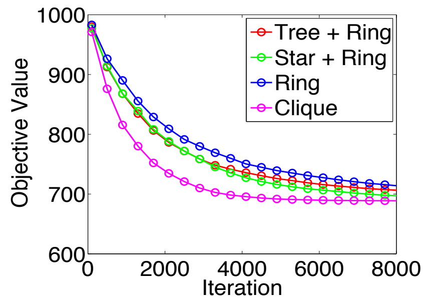
Our first set of experiments test the affect of the connectivity of the graph on the convergence rate. In particular, recall that the convergence analysis established in Theorem 3 depends on the Laplacian of the communication graph. In this experiment we demonstrate how communication constraints affect convergence in practice. We experiment with the following graph topologies of graph G: Ring, Clique, Star + Ring (i.e., the union of edges of a star and a ring) and Tree + Ring. On each layout we run the sequential Algorithm 1 on the following quadratic problem

$$\begin{aligned} \min \quad & C\sum_{i=1}^{N} \|x_i - (i \bmod 10)\mathbf{1}\|^2 \\ s.t. \quad & \sum_{i=1}^{N} A_i x_i = 0, \end{aligned} \tag{8}$$

Note the decomposable structure of the problem. For this experiment, we use $N = 1000$ and $x_i \in \mathbb{R}^{50}$. We have 10 constraints whose coefficients are randomly generated from $U[0,1]$ and we choose C such that the objective evaluates to 1000 when $x = 0$.

The results for Algorithm 1 on each topology for 10000 iterations are shown in Figure 1. The results clearly show that better connectivity implies better convergence rate. Note that while the clique topology has significantly better convergence than other topologies, acceptable long-term performance can be achieved by much sparser topologies such as Star + Ring and Tree + Ring.

Having a sparse communication graph is important to lower the cost of a distributed system. Furthermore, it is worth mentioning that the sparsity of the communication graph is also important in a multicore setting; since Algorithm 1 requires computing $(A_iA_i^\top + A_jA_j^\top)^+$ for each communicating pair of nodes (i, j). Our analysis shows that this computation takes a significant portion of the running time and hence it is essential to minimize the number of variable pairs that are allowed to be updated.



h

Figure 1: Objective value vs. number of iterations for different graph topologies. Note that larger the connectivity of the graph, faster is the convergence.

6.2 Concurrency and Synchronization

As seen earlier, compared to Tree + Ring, Star + Ring is a low diameter layout (diameter = 2). Hence, in a sequential setting, it indeed results in a faster convergence. However, Star + Ring requires a node to be connected to all other nodes. This high-degree node could be a contention point in a parallel setting. We test the performance of our asynchronous algorithm in this setting. To assess how the performance would be affected with such contention and how asynchronous updates would increase performance, we conduct another experiment on the synthetic problem (8) but on a larger scale ($N = 10000$, $x_i \in \mathbb{R}^{100}$, 100 constraints).

Our concurrent update follows a master/slave scheme. Each thread performs a loop where in each iteration it elects a master i and slave j and then applies the following sequence of actions:

1. Obtain the information required for the update from the master (i.e., information for calculating the gradients used for solving the subproblem).

2. Send the master information to the slave, update the slave variable and get back the information needed to update the master.

3. Update the master based (only) on the information obtained from steps 1 and 2.

We emphasize that the master is not allowed to read its own state at step 3 except to apply an increment, which is computed based on steps 1 and 2. This ensures that the master's increment is consistent with that of the slave, even if one or both of them was being concurrently overwritten by

[2]All experiments were conducted on a Google Compute Engine virtual machine of type "n1-highcpu-16", which comprises 16 virtual CPUs and 14.4 GB of memory. For more details, please refer to `https://cloud.google.com/compute/docs/machine-types#highcpu`.

another thread. More details on the implementation can be found in [12].

Given this update scheme, we experiment with three levels of synchronization: (a) **Double Locking:** Locks the master and the slave through the entire update. Because the objective function is decomposable, a more conservative locking (e.g. locking all nodes) is not needed. (b) **Single Locking:** Locks the master during steps 1 and 3 (the master is unlocked during step 2 and locks the slave during step 2). (c) **Lock-free:** No locks are used. Master and slave variables are updated through atomic increments similar to Hogwild! method.

Following [30], we use spinlocks instead of mutex locks to implement locking. Spinlocks are preferred over mutex locks when the resource is locked for a short period of time, which is the case in our algorithm. For each locking mechanism, we vary the number of threads from 1 to 15. We stop when $f_0 - f_t > 0.99(f_0 - f^*)$, where f^* is computed beforehand up to three significant digits. Similar to [30], we add artificial delay to steps 1 and 2 in the update scheme to model complicated gradient calculations and/or network latency in a distributed setting.

Figure 2 shows the speedup for Tree + Ring and Star + Ring layouts. The figure clearly shows that a fully synchronous method suffers from contention in the Star + Ring topology whereas asynchronous method does not suffer from this problem and hence, achieves higher speedups. Although the Tree + Ring layouts achieves higher speedup than Star + Ring, the latter topology results in much less running time ($\sim$ 67 seconds vs 91 seconds using 15 threads).

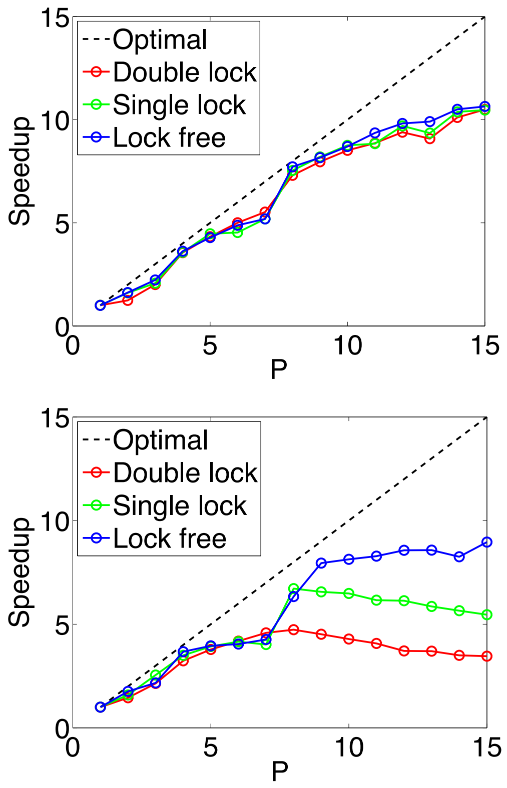


Figure 2: Speedup for Tree + Ring (top) and Star + Ring (bottom) topologies and different levels of synchronization. Note for Star + Ring topology, speedup of asynchronous algorithm is significantly higher than that of synchronous version.

6.3 Practical Case Study: Parallel Training of Linear SVM

In this section, we explore the effect of parallelism on randomized CD for training a linear SVM based on the dual formulation stated in (7). Necoara et. al. [22] have shown that, in terms of CPU time, a sequential randomized CD outperforms coordinate descent using Gauss-Southwell selection rule. It was also observed that randomized CD outperforms LIBSVM [6] for large datasets while maintaining reasonable performance for small datasets.

In this experiment we use a clique layout. For SVM training in a multicore setting, using a clique layout does not introduce additional cost compared to a more sparse layout. To maintain the box constraint, we use the double-locking scheme described in Section 6.2 for updating a pair of dual variables.

One advantage of coordinate descent algorithms is that they do not require the storage of the Gram matrix; instead they can compute its elements on the fly. That comes, however, at the expense of CPU time. Similar to [22], to speed up gradient computations without increasing memory requirements, we maintain the primal weight vector of the linear SVM and use it to compute gradients. Basically, if we increment α_i by δ_i and α_j by δ_j, then we increment the weight vector by $\delta_i y_i x_i + \delta_j y_j x_j$. This increment is accomplished using atomic additions. However, this implies that all threads will be concurrently updating the primal weight vector. Similar to [30], we require these updates to be sparse with small overlap between non-zero coordinates in order to ensure convergence. In other words, we require training examples to have sparse features with small overlap between non-zero features.

We report speedups on two datasets used in [22].[3] Table 2 provides a description of both the datasets. For each dataset, we train the SVM model until $f_0 - f_t > 0.9999(f_0 - f^*)$, where f^* is the objective reported in [22]. In Figure 3, we report speedup for both the datasets. The figure shows that parallelism indeed increases the performance of randomized CD training of linear SVM.

[3]Datasets can be downloaded from `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets`.

Dataset	# of instances	# of features	Avg # of non-zero features
`a7a`	16100	122	14
`w8a`	49749	300	12

Table 2: Datasets used for linear SVM Speedup experiment

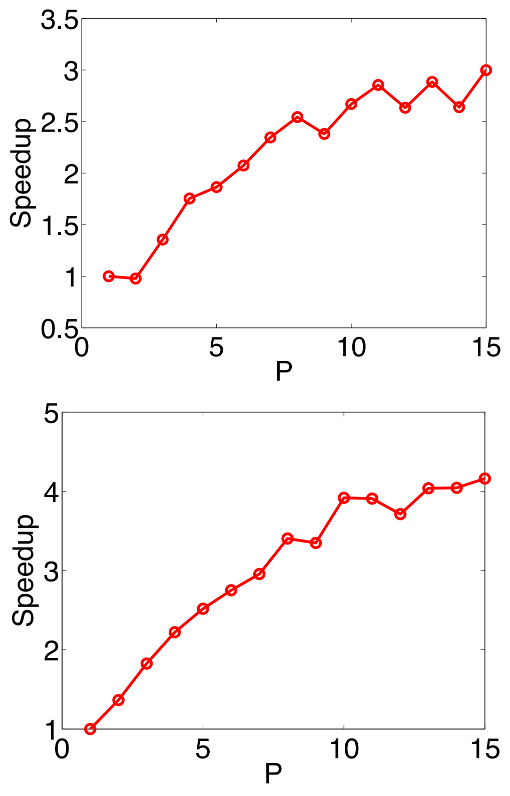


Figure 3: Speedup for linear SVM training on `a7a` (top) and `w8a` datasets.

7 DISCUSSION AND FUTURE WORK

We presented randomized coordinate descent methods for solving convex optimization problems with linear constraints that couple the variables. Moreover, we also presented composite objective, stochastic, and asynchronous versions of our basic method and provided their convergence analysis. We demonstrated the empirical performance of the algorithms. The experimental results of asynchronous algorithm look very promising.

There are interesting open problems for our problem in consideration: First, we would like to obtain high-probability results not just in expectation; another interesting direction is to extend the asynchronous algorithm to the non-smooth setting. Finally, while we obtain $O(1/k)$ for general convex functions, obtaining an accelerated $O(1/k^2)$ rate is a natural question.

Acknowledgments

SS is partly supported by NSF grant: IIS-1409802. We thanks the anonymous reviewers for the helpful comments.

References

[1] A. Auslender. *Optimisation Méthodes Numériques*. Masson, 1976.

[2] A. Beck and L. Tetruashvili. On the convergence of block coordinate descent type methods. *SIAM Journal on Optimization*, 23(4):2037–2060, 2013. doi: 10.1137/120887679.

[3] P. Berman, N. Kovoor, and P. M. Pardalos. A linear-time algorithm for the least-distance problem. Technical report, Pennsylvania State University, Department of Computer Science, 1992.

[4] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, second edition, 1999.

[5] J. Bradley, A. Kyrola, D. Bickson, and C. Guestrin. Parallel coordinate descent for L1-regularized loss minimization. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning*, pages 321–328. Omnipress, 2011.

[6] C.-C. Chang and C.-J. Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, May 2011. ISSN 2157-6904. doi: 10.1145/1961189.1961199. URL `http://doi.acm.org/10.1145/1961189.1961199`.

[7] O. Fercoq and P. Richtárik. Accelerated, parallel and proximal coordinate descent. *CoRR*, abs/1312.5799, 2013.

[8] J. Friedman, T. Hastie, H. Höfling, R. Tibshirani, et al. Pathwise coordinate optimization. *The Annals of Applied Statistics*, 1(2):302–332, 2007.

[9] D. Gabay and B. Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & Mathematics with Applications*, 2(1):17–40, 1976. doi: http://dx.doi.org/10.1016/0898-1221(76)90003-1. URL `http://www.sciencedirect.com/science/article/pii/0898122176900031`.

[10] R. Glowinski and A. Marrocco. Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de dirichlet non linéares. *Revue Française d'Automatique, Informatique, et Recherche Opérationelle*, 1975.

[11] G. H. Golub and C. F. Van Loan. *Matrix Computations*. John Hopkins University Press, Baltimore, MD, 3rd edition, 1996.

[12] A. Hefny, S. Reddi, and S. Sra. Coordinate descent algorithms with coupling constraints: Lessons learned. In *NIPS Workshop on Software Engineering For Machine Learning*, 2014.

[13] M. Hong and Z.-Q. Luo. On the linear convergence of the alternating direction method of multipliers. *arXiv preprint arXiv:1208.3922*, 2012.

[14] M. Hong, X. Wang, M. Razaviyayn, and Z.-Q. Luo. Iteration Complexity Analysis of Block Coordinate Descent Methods. *arXiv:1310.6957*, 2013.

[15] C. J. Hsieh and I. S. Dhillon. Fast coordinate descent methods with variable selection for non-negative matrix factorization. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining(KDD)*, pages 1064–1072, August 2011.

[16] C. J. Hsieh, K. W. Chang, C. J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In W. Cohen, A. McCallum, and S. Roweis, editors, *ICML*, pages 408–415. ACM, 2008.

[17] C.-J. Hsieh, M. A. Sustik, I. S. Dhillon, and P. D. Ravikumar. Sparse inverse covariance matrix estimation using quadratic approximation. In *NIPS*, pages 2330–2338, 2011.

[18] S. Lacoste-Julien, M. Jaggi, M. Schmidt, and P. Pletscher. Block-coordinate frank-wolfe optimization for structural svms. *arXiv preprint arXiv:1207.4747*, 2012.

[19] C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems*. Prentice–Hall, Englewood Cliffs, NJ, 1974. Reissued with a survey on recent developments by SIAM, Philadelphia, 1995.

[20] J. Liu, S. J. Wright, C. Ré, V. Bittorf, and S. Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. *arXiv:1311.1873*, 2013.

[21] Z.-Q. Luo and P. Tseng. On the convergence of the coordinate descent method for convex differentiable minimization. *Journal of Optimization Theory and Applications*, 72(1):7–35, 1992.

[22] I. Necoara and A. Patrascu. A random coordinate descent algorithm for optimization problems with composite objective function and linear coupled constraints. *Comp. Opt. and Appl.*, 57(2):307–337, 2014.

[23] I. Necoara, Y. Nesterov, and F. Glineur. A random coordinate descent method on large optimization problems with linear constraints. Technical report, Technical Report, University Politehnica Bucharest, 2011, 2011.

[24] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM J. on Optimization*, 19(4): 1574–1609, Jan. 2009. ISSN 1052-6234. doi: 10.1137/070704277.

[25] Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. Core discussion papers, Université catholique de Louvain, Center for Operations Research and Econometrics (CORE), 2010.

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

[27] J. C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, ADVANCES IN KERNEL METHODS - SUPPORT VECTOR LEARNING, 1998.

[28] B. T. Polyak. *Introduction to Optimization*. Optimization Software Inc., 1987. Nov 2010 revision.

[29] L. A. Rastrigin. *Statisticheskie Metody Poiska Ekstremuma (Statistical Extremum Seeking Methods)*. Nauka, Moscow, 1968.

[30] B. Recht, C. Re, S. J. Wright, and F. Niu. Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, and K. Q. Weinberger, editors, *NIPS*, pages 693–701, 2011.

[31] P. Richtárik and M. Takáč. Distributed coordinate descent method for learning with big data. *ArXiv e-prints*, Oct. 2013.

[32] P. Richtárik and M. Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *arXiv:1107.2848v1*, July 2011.

[33] P. Richtárik and M. Takáč. Parallel coordinate descent methods for big data optimization. *arXiv:1212.0873v1*, Dec 2012.

[34] A. Saha and A. Tewari. On the nonasymptotic convergence of cyclic coordinate descent methods. *SIAM Journal on Optimization*, 23(1):576–601, 2013.

[35] S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *JMLR*, 14, 2013.

[36] S. Shalev-Shwartz and T. Zhang. Accelerated mini-batch stochastic dual coordinate ascent. *arXiv:1305.2581v1*, 2013.

[37] R. Tappenden, P. Richtárik, and J. Gondzio. Inexact coordinate descent: complexity and preconditioning. *arXiv:1304.5530*, 2013.

[38] R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1):91–108, 2005.

[39] P. Tseng and S. Yun. A block-coordinate gradient descent method for linearly constrained nonsmooth separable optimization. *Journal of Optimization Theory and Applications*, 2009.

[40] P.-W. Wang and C.-J. Lin. Iteration complexity of feasible descent methods for convex optimization. *JMLR*, 15:1523–1548, 2014.