Parallel and Distributed Block-Coordinate Frank-Wolfe Algorithms

Yu-Xiang Wang [†] YUXIANGW@CS.CMU.EDU
Veeranjaneyulu Sadhanala [†] VSADHANA@CS.CMU.EDU
Wei Dai [†] WDAI@CS.CMU.EDU
Willie Neiswanger [†] WILLIE@CS.CMU.EDU
Suvrit Sra [‡] SUVRIT@MIT.EDU
Eric P. Xing [†] EPXING@CS.CMU.EDU

[†] Carnegie Mellon University, 5000 Forbes Ave, PA 15213, USA

[‡] Massachusetts Institute of Technology, 77 Massachusetts Ave, Cambridge, MA 02139, USA

Abstract

We study parallel and distributed Frank-Wolfe algorithms; the former on shared memory machines with mini-batching, and the latter in a delayed update framework. In both cases, we perform computations asynchronously whenever possible. We assume block-separable constraints as in Block-Coordinate Frank-Wolfe (BCFW) method (Lacoste-Julien et al., 2013), but our analysis subsumes BCFW and reveals problem-dependent quantities that govern the speedups of our methods over BCFW. A notable feature of our algorithms is that they do not depend on worst-case bounded delays, but only (mildly) on *expected* delays, making them robust to stragglers and faulty worker threads. We present experiments on structural SVM and Group Fused Lasso, and observe significant speedups over competing state-of-the-art (and synchronous) methods.

1. Introduction

The classical Frank-Wolfe (FW) algorithm (Frank & Wolfe, 1956) has witnessed a huge surge of interest recently (Ahipasaoglu et al., 2008; Clarkson, 2010; Jaggi, 2011; 2013). The FW algorithm iteratively minimizes a smooth function f (typically convex) over a compact convex set $\mathcal{M} \subset \mathbb{R}^m$. Unlike methods based on projection, FW uses just a *linear oracle* that solves $\min_{x\in\mathcal{M}} \langle x, g\rangle$, which can be much simpler and faster than projection.

This feature underlies the great popularity of FW, which has by now witnessed several extensions such as regularized FW (Bredies et al., 2009; Harchaoui et al., 2015; Zhang et al., 2013), linearly convergent special cases (Garber & Hazan, 2013; Lacoste-Julien & Jaggi, 2015), stochastic versions (Hazan & Kale, 2012; Lafond et al., 2015; Ouyang & Gray, 2010), and a randomized block-coordinate FW (Lacoste-Julien et al., 2013).

Despite this progress, parallel and distributed FW variants are barely known. We fill this gap and develop new asynchronous FW algorithms, for the particular setting where the constraint set $\mathcal{M}$ is *block-separable*; thus, we solve

$$\min_x \; f(x) \text{ s.t. } x = [x_{(1)}, ..., x_{(n)}] \in \prod_{i=1}^{n} \mathcal{M}_i, \qquad (1)$$

where $\mathcal{M}_i \subset \mathbb{R}^{m_i}$ $(1 \le i \le n)$ is a compact convex set and $x_{(i)}$ are coordinate partitions of x. This setting for FW was considered in Lacoste-Julien et al. (2013), who introduced the Block-Coordinate Frank-Wolfe (BCFW) method.

Such problems arise in many applications, notably, structural SVMs (Lacoste-Julien et al., 2013), routing (LeBlanc et al., 1975), group fused lasso (Alaíz et al., 2013; Bleakley & Vert, 2011), trace-norm based tensor completion (Liu et al., 2013), reduced rank nonparametric regression (Foygel et al., 2012), and structured submodular minimization (Jegelka et al., 2013), among others.

A standard approach to solve (1) is via block-coordinate (gradient) descent (BCD), which forms a local quadratic model for a block of variables, and then solves a *projection* subproblem (Beck & Tetruashvili, 2013; Nesterov, 2012; Richtárik & Takáč, 2015). However, for many problems, including the ones noted above, projection can be expensive (e.g., projecting onto the trace norm ball, onto base polytopes Fujishige & Isotani, 2011), and in some cases even computationally intractable (Collins et al., 2008).

Frank-Wolfe methods excel in such scenarios as they rely

Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48.

only on linear oracles that solve $\min_{s\in\mathcal{M}}\langle s, \nabla f(\cdot)\rangle$. For $\mathcal{M} = \prod_i \mathcal{M}_i$, this breaks into the n independent problems

$$\min_{s_{(i)}\in\mathcal{M}_i} \quad \langle s_{(i)}, \nabla_{(i)} f(x)\rangle, \quad 1 \le i \le n, \tag{2}$$

where $\nabla_{(i)}$ denotes the gradient w.r.t. coordinates $x_{(i)}$. It is obvious that these n subproblems can be solved in parallel (an idea dating back to at least as early as LeBlanc et al., 1975). However, having to update all the coordinates at each iteration is expensive, hampering the use of FW on big-data problems.

This drawback is partially ameliorated by BCFW (Lacoste-Julien et al., 2013), which randomly selects a block $\mathcal{M}_i$ at each iteration and performs FW updates. But these updates are *strictly* sequential, and do not take advantage of modern multicore architectures or of distributed clusters.

Contributions. Our main contributions are the following:

- Asynchronous Parallel block-coordinate Frank-Wolfe algorithms (AP-BCFW) for both shared-memory and distributed settings. Moreover, AP-BCFW depends only (mildly) on the *expected* delay, therefore is robust to stragglers and faulty worker threads.
- An analysis of the primal and primal-dual convergence of AP-BCFW and its variants for any minibatch size and potentially unbounded maximum delay. When the maximum delay is actually bounded, we show stronger results using results from load-balancing on max-load bounds.
- Insightful deterministic conditions under which minibatching *provably* improves the convergence rate for a class of problems (sometimes by orders of magnitude).
- Experiments that demonstrate on real data how our algorithm solves a structural SVM problem several times faster than the state-of-the-art.

In short, our results contribute towards making FW more attractive for big-data applications. To add perspective, we compare our methods to closely related works below; we refer the reader to Freund & Grigas (2014); Jaggi (2013); Lacoste-Julien et al. (2013); Zhang et al. (2012) for additional notes and references.

BCFW and Structural SVM. Our algorithm AP-BCFW extends and generalizes BCFW to parallel computation. Our analysis follows the structure in (Lacoste-Julien et al., 2013), but uses different stepsizes that must be carefully chosen. Our results contain BCFW as a special case. Lacoste-Julien et al. (2013) primarily focus on more explicit (and stronger) guarantee for BCFW on structural SVM, while we mainly focus on a more general class of problems; the particular subroutine needed by structural SVM requires special treatment though (see Appendix C).

Parallelization of sequential algorithms. The idea of parallelizing sequential optimization algorithms is not new. It dates back to (Tsitsiklis et al., 1986) for stochastic gradient methods; more recently Lee et al. (2014); Liu et al. (2014); Richtárik & Takáč (2015) study parallelization of BCD. The conditions under which these parallel BCD methods succeed, e.g., expected separable overapproximation (ESO), and coordinate Lipschitz conditions, bear a close resemblance to our conditions in Section 3.2, but are not the same due to differences in how solutions are updated and what subproblems arise. In particular, our conditions are *affine invariant*. We provide detailed comparisons to parallel coordinate descent in Appendix D.5.

Asynchronous algorithms. Asynchronous algorithms that allow delayed parameter updates have been proposed earlier for stochastic gradient descent (Niu et al., 2011) and parallel BCD (Liu et al., 2014). We propose the first asynchronous algorithm for Frank-Wolfe. Our asynchronous scheme not only permits delayed minibatch updates, but also allows the updates for coordinate blocks *within each* minibatch to have different delays. Therefore, each update may not be a solution of (2) for any single x. Moreover, we obtain strictly better dependence on the delay parameter than predecessors (e.g., an *exponential improvement* over Liu et al. (2014)) possibly due to a sharper analysis.

Other related work. While preparing our manuscript, we discovered the preprint (Bellet et al., 2014) which also studies distributed Frank-Wolfe. We note that (Bellet et al., 2014) focuses on Lasso type problems and communication costs, and hence, is not directly comparable to our results.

Notation. We briefly summarize our notation now. The vector $x \in \mathbb{R}^m$ denotes the parameter vector, possibly split into n coordinate blocks. For block $i = 1, ..., n$, $E_i \in \mathbb{R}^{m\times m_i}$ is the projection matrix which projects $x \in \mathbb{R}^m$ down to $x_{(i)} \in \mathbb{R}^{m_i}$; thus $x_{(i)} = E_i x$. The adjoint operator E_i^* maps $\mathbb{R}^{m_i} \to \mathbb{R}^m$, thus $x_{[i]} = E_i^* x_{(i)}$ is x with zeros in all dimensions except $x_{(i)}$ (note the subscript $x_{[i]}$). We denote the size of a minibatch by τ, and the number of parallel workers (threads) by T. Unless otherwise stated, k denotes the iteration/epoch counter and γ denotes a stepsize. Finally, C_f^τ (and other such constants) denotes some curvature measure associated with function f and minibatch size τ. Such constants are important in our analysis, and will be described in greater detail in the main text.

2. Algorithm

In this section, we present an asynchronous parallel block-coordinate Frank-Wolfe algorithm (AP-BCFW) to solve (1). Our algorithm is designed to run fully asynchronously on either a shared-memory multicore architecture or on a distributed system.

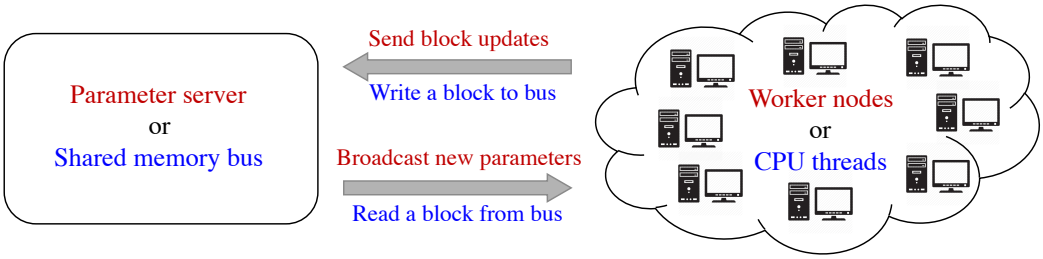


Figure 1. Illustration of the AP-BCFW in the distributed (in red) and share-memory settings (in blue). The "cloud" of all worker nodes (or CPU threads) is abstracted into an oracle that keeps feeding the server (or writing to the memory bus) with updates from solving possibly approximate (and/or delayed) solutions to (2) on iid uniform random blocks.

Algorithm 1 AP-BCFW: Asynchronous Parallel Block-Coordinate Frank-Wolfe (distributed)

Input: An initial feasible $x^{(0)}$, mini-batch size τ, a "Cloud" oracle $\mathcal{O}$ satisfying Assumptions A1, A2.
0. Broadcast $x^{(0)}$ to all workers in $\mathcal{O}$.
for k = 1,2,... (k is the iteration number) **do**
1. Keep receiving $(i, s_{(i)})$ from $\mathcal{O}$ until we have τ disjoint blocks (overwrite if collision[1]). Denote the index set by S.
2. Update $x^{(k)} = x^{(k-1)} + \gamma_k \sum_{i \in S}(s_{[i]} - x_{[i]}^{(k-1)})$ with $\gamma_k = \frac{2n\tau}{\tau^2 k + 2n}$ or via line-search.
3. Broadcast $x^{(k)}$ (or just $x^{(k)} - x^{(k-1)}$) to $\mathcal{O}$.
4. Break if converged.
end for
Output: $x^{(k)}$.

For the shared-memory model, the computational work is divided amongst worker threads, each of which has access to a pool of coordinates that it may work on, as well as to the shared parameters. This setup matches the system assumptions in (Liu et al., 2014; Niu et al., 2011; Richtárik & Takáč, 2015), and most modern multicore machines permit such an arrangement.

On a distributed system, the parameter server (Dai et al., 2013; Li et al., 2013) broadcasts the most recent parameter vector periodically to each worker and workers keep sending updates to the parameter vector after solving the subroutines corresponding to a randomly chosen parameter. In either setting, we do not wait for slower workers or synchronize the parameters at any point of the algorithm, therefore many updates sent from the workers could be calculated based on a delayed parameter.

For convenience, we treat the pool of all workers as a single "cloud" oracle $\mathcal{O}$ that keeps sending updates of form $\{i, s_{(i)}\}$ to the server, where i selects a block and $s_{(i)}$ is an approximate solution to (2) at the current parameter. Moreover, we assume that

A1. The sequence of i from $\mathcal{O}$ is sampled i.i.d. uniformly from $\{1, 2, ..., n\}$.

[1]We bound the probability of collisions in Appendix D.2.

Assumption A1 is critical as it ensures Step 2 in the algorithm is an unbiased approximation of the batch FW. This assumption allows the workers to be arbitrarily heterogeneous as long as they each sample blocks i.i.d. uniformly and the time for each worker to produce $s_{(i)}$ does not depend on the block index i. Admittedly, this could be troublesome for some applications with heterogeneous blocks, we describe ways to enforce A1 in Appendix D.1.

An advantage of this oracle abstraction is its potential applicability well beyond the per-worker i.i.d. scheme. In practice, each worker might only have access to a small subset of $[n]$ and might be doing sequential sampling with periodic reshuffling. At the aggregate level, however, the oracle assumptions might still be reasonable approximations, especially if the number of workers T is large.

Both distributed and shared-memory settings can be captured under this oracle as is illustrated in Figure 1. Pseudocode of our scheme is given in Algorithm 1.

3. Analysis

The three key questions pertaining to Algorithm 1 are:

- *Does it converge?*
- *How fast? How much faster than* BCFW *($\tau = 1$)?*
- *How do delayed updates affect the convergence?*

We answer the first two questions in Sections 3.1 and 3.2. Specifically, we show that AP-BCFW converges at a $O(1/k)$ rate. Our analysis reveals that the speedup of AP-BCFW over BCFW via parallelization is problem dependent. Intuitively, we show that speedups due to mini-batching ($\tau > 1$) depend on the average "coupling" of the objective function f across different coordinate blocks. For example, if f has a block symmetric diagonally dominant Hessian, then AP-BCFW converges $\tau/2$ times faster. We address the third question in Section 3.3, where we establish convergence results that depend only mildly on the "expected" delay κ. The bound is proportional to κ when we

allow the delay to grow unboundedly, and proportional to $\sqrt{\kappa}$ when the delay is bounded by a small $\kappa_{\max}$.

3.1. Main convergence results

We begin by defining a few quantities needed for our analysis. The first key quantity—also key to the analysis of several other FW methods—is the notion of **curvature**. Since AP-BCFW updates a subset of coordinate blocks at a time, we define *set curvature* for an index set $S \subseteq [n]$ as

$$C_f^{(S)} := \sup_{\substack{x\in\mathcal{M}, s_{(S)}\in\mathcal{M}^{(S)},\\ \gamma\in[0,1],\\ y=x+\gamma(s_{[S]}-x_{[S]})}} \frac{2}{\gamma^2}\big(f(y)-f(x)- \langle y_{(S)}-x_{(S)}, \nabla_{(S)}f(x)\rangle\big). \tag{3}$$

For index sets of size τ, we define the *expected set curvature* over a uniform choice of subsets as

$$C_f^\tau := \mathbb{E}_{S:|S|=\tau}[C_f^{(S)}] = \binom{n}{\tau}^{-1}\sum\nolimits_{S\subset[n],|S|=\tau} C_f^{(S)}. \tag{4}$$

These curvature definitions are closely related to the global curvature constant C_f of Jaggi (2013) and the coordinate curvature $C_f^{(i)}$ and product curvature $C_f^{\otimes}$ of Lacoste-Julien et al. (2013). Lemma 1 makes this relation more precise.

Lemma 1 (Curvature relations)**.** *Suppose $S \subseteq [n]$ with cardinality $|S| = \tau$ and $i \in S$. Then,*

i) $C_f^{(i)} \le C_f^{(S)} \le C_f$;

ii) $\frac{1}{n}C_f^{\otimes} = C_f^1 \le C_f^\tau \le C_f^n = C_f$.

How the expected set curvature C_f^τ scales with τ is critical to bounding the speedup we can expect over BCFW; we provide a detailed analysis of this speedup in Section 3.2.

The next key object is an **approximate linear minimizer**. As in (Jaggi, 2013; Lacoste-Julien et al., 2013), we also allow the core computational subroutine that solves (2) to yield an approximate minimizer $s_{(i)}$. Formally, we assume:

A2. There is a constant $\delta \ge 0$, such that for every $k \ge 1$, the chosen minibatch $S \subset [n]$ of size τ and the corresponding blocks $s_{(S)} := (s_{(i)})_{i\in S}$ from $\mathcal{O}$ obey

$$\mathbb{E}\left[\langle s_{(S)}, \nabla_{(S)}f^{(k)}\rangle - \min_{s'\in\mathcal{M}^{(S)}}\langle s', \nabla_{(S)}f^{(k)}\rangle\right] \le \frac{\delta\gamma_k C_f^\tau}{2}. \tag{5}$$

where the expectation is taken over the random sequence of minibatch indices and corresponding updates from $\mathcal{O}$ in the entire history up to step k.

Assumption A2 is strictly weaker than what is required in (Jaggi, 2013; Lacoste-Julien et al., 2013), as we only need the approximation to hold *in expectation*. With these definitions in hand, we are ready to state our convergence result.

Theorem 2 (Primal Convergence)**.** *Say we use a "Cloud" oracle $\mathcal{O}$ that generates a sequence of updates satisfying A1 and A2. Then, for each $k \ge 0$, the iterations in Algorithm 1 and its line search variant obey*

$$\mathbb{E}[f(x^{(k)})] - f(x^*) \le \frac{2nC}{\tau^2 k + 2n},$$

where $C = nC_f^\tau(1+\delta) + f(x^{(0)}) - f(x^*)$.

At a first glance, the $n^2 C_f^\tau$ term in the numerator might seem bizarre, but as we will see in the next section, C_f^τ can be as small as $O(\frac{\tau}{n^2})$. This is the scale of the constant one should keep in mind to compare the rate to other methods, e.g., coordinate descent. Also note that so far this convergence result does not explicitly work for delayed updates, which we will analyze in Section 3.3 separately via the approximation parameter δ from (5).

For FW methods, one can also easily obtain a convergence guarantee in an appropriate primal-dual sense. To this end, we introduce our version of the **surrogate duality gap** (Jaggi, 2013); we define this as

$$\begin{aligned} g(x) &:= \max_{s\in\mathcal{M}}\langle x-s, \nabla f(x)\rangle \\ &= \sum_{i=1}^n \max_{s_{(i)}\in\mathcal{M}^{(i)}}\langle x_{(i)}-s_{(i)}, \nabla_{(i)}f(x)\rangle = \sum_{i=1}^n g^{(i)}(x). \end{aligned} \tag{6}$$

To see why (6) is actually a duality gap, note that since f is convex, the linearization $f(x) + \langle s-x, \nabla f(x)\rangle$ is always smaller than the function evaluated at any s, so that

$$g(x) \ge \langle x - x^*, \nabla f(x)\rangle \ge f(x) - f(x^*).$$

This duality gap is obtained for "free" in batch FW, but not in BCFW or AP-BCFW. Here, we only have an unbiased estimate $\frac{n}{|S|}\sum_{i\in S} g^{(i)}(x)$. For large τ, this estimate is close to $g(x)$ with high probability (McDiarmid's Inequality), and can still be useful as a stopping criterion.

Theorem 3 (Primal-Dual Convergence)**.** *Assume $\mathcal{O}$ satisfies A1 and A2. Define the expected surrogate duality gap $g_k := \mathbb{E}g(x^{(k)})$ and weighted average $\bar{g}_k := \frac{2}{K(K+1)}\sum_{k=1}^K k g_k$ for the sequence of parameters $x^{(k)}$ in Algorithm 1. Then for very $K \ge 1$, there exists $k^* \in [K]$ such that*

$$g_{k^*} \le \bar{g}_K \le \frac{6nC}{\tau^2(K+1)},$$

with the same C in Theorem 2.

Relation with FW and BCFW: The above convergence guarantees can be thought of as an interpolation between BCFW and batch FW. If we take $\tau = 1$, they give exactly the convergence guarantee for BCFW (Lacoste-Julien et al., 2013, Theorem 2), and if we take $\tau = n$, we can drop $f(x^{(0)}) - f(x^*)$ from C (with a small modification in the analysis) and recover the classic batch guarantee as in Jaggi (2013).

Dependence on initialization: Unlike classic FW, the convergence rate of our method depends on the initialization. When $h_0 := f(x^{(0)}) - f(x^*) \geq nC_f^\tau$ and $\tau^2 < n$, the convergence is slower by a factor of $\frac{n}{\tau^2}$. The same concern was also raised in (Lacoste-Julien et al., 2013) with $\tau = 1$. We can actually remove the $f(x^{(0)}) - f(x^*)$ from C as long as we know that $h_0 \leq nC_f^\tau$. By Lemma 1, the expected set curvature C_f^τ increases with τ, so the fast convergence region becomes larger when we increase τ. In addition, if we pick $\tau^2 > n$, the rate of convergence is not affected by initialization anymore.

Speedup: The reader may have noticed the $n^2 C_f^\tau$ term in the numerator. This is undesirable as n can be large (for instance, in structural SVM n is the total number of data points). The saving grace in BCFW is that when $\tau = 1$, C_f^τ is as small as $O(n^{-2})$ (see Lacoste-Julien et al., 2013, Lemmas A1 and A2), and it is easy to check that the dependence on n is the same even for $\tau > 1$. What really matters is how much speedup one can achieve over BCFW, and this relies critically on how C_f^τ depends on τ. Analyzing this dependence is our main focus in the next section.

3.2. Effect of parallelism / mini-batching

To understand when mini-batching is meaningful and to quantify its speedup, below we take a more careful look at the expected set curvature C_f^τ. In particular, we analyze and present a set of insightful conditions that govern its dependence on τ. The key idea is to quantify how strongly different coordinate blocks interact with each other.

To begin, assume that there exists a positive semidefinite matrix H such that for any $x, y \in \mathcal{M}$

$$f(y) \leq f(x) + \langle y - x, \nabla f(x) \rangle + \frac{1}{2}(y-x)^T H (y-x). \quad (7)$$

The matrix H may be viewed as a generalization of the gradient's Lipschitz constant (a scalar) to a matrix. For quadratic functions $f(x) = \frac{1}{2}x^T Q x + c^T x$, we can take $H = Q$. For twice differentiable functions, we can choose $H \in \{K \mid K \succeq \nabla^2 f(x), \ \forall x \in \mathcal{M}\}$.

Since $x = [x_1, \ldots, x_n]$ (we write x_i instead of $x_{(i)}$ for brevity), we separate H into $n \times n$ blocks; so H_{ij} represents the block corresponding to x_i and x_j such that we can take the product $x_i^T H_{ij} x_j$. Now, we define a *boundedness* parameter B_i for every i, and an *incoherence condition* with parameter μ_{ij} for every block coordinate pair $\mathcal{M}_i, \mathcal{M}_j$ such that

$$B_i = \sup_{x_i \in \mathcal{M}_i} x_i^T H_{ii} x_i, \quad \mu_{ij} = \sup_{x_i \in \mathcal{M}_i, x_j \in \mathcal{M}_j} x_i^T H_{ij} x_j,$$

$$B = \mathbb{E}_{i \sim \mathrm{Unif}([n])} B_i, \qquad \mu = \mathbb{E}_{(i,j) \sim \mathrm{Unif}(\{(i,j) \in [n]^2, i \neq j\})} \mu_{ij}.$$

Using these quantities, we obtain the following bound on the expected set-curvature.

Theorem 4. $C_f^\tau \leq 4(\tau B + \tau(\tau - 1)\mu)$ *for any* $\tau \in [n]$.

It is clear that when the incoherence term μ is large, the expected set curvature C_f^τ is proportional to τ^2, and when μ is close to 0, then C_f^τ is proportional to τ. In other words, when the interaction between coordinates block is small, one gains from parallelizing BCFW. This is analogous to the situation in parallel coordinate descent (Liu et al., 2014; Richtárik & Takáč, 2015) and we will compare the rate of convergence explicitly with them in Appendix D.5.

Corollary 5. *Consider a matrix M with B_i on the diagonal and μ_{ij} on the off-diagonal. If M is* symmetric diagonally dominant *(SDD), i.e., the sum of absolute off-diagonal entries in each row is no greater than the diagonal entry, then C_f^τ is proportional to τ.*

The above result depends on the parameters B and μ. In Appendix D.4, we provide two concrete examples (multi-class classification with structural SVM and graph fused lasso) where we can express B and μ as problem-dependent quantities and provide explicit upper bounds of C_f^τ. In both examples, we show that choosing larger τ yields faster convergence (at least up to some point).

3.3. Convergence with delayed updates

Often due to the delays in communication, some updates pushed back by workers are calculated based on delayed parameters that were broadcast earlier. Dropping these updates or enforcing synchronization will create a huge system overhead especially when the size of the minibatch is small. Ideally, we want to just accept the delayed updates as if they were correct, and broadcast new parameters to workers without locking the updates. The question is, does this idea work?

In this section, we model delays from updates to be i.i.d. from an unknown distribution that can depend on k, but not on blocks. Under these assumptions, we show that the effect of delayed updates can be treated as an approximate oracle that satisfies A2 in (5) with some specific constant δ that depends on the expected delay κ and the maximum delay parameter $\kappa_{\max}$ (when it exists). This allows us to invoke results in Section 3.1 to establish convergence for delayed updates. The results also depend on the following diameter and gradient Lipschitz constant for a norm $\|\cdot\|$

$$D_{\|\cdot\|}^{(S)} = \sup_{x,y \in \mathcal{M}^{(S)}} \|x - y\|,$$

$$L_{\|\cdot\|}^{(S)} = \sup_{\substack{x,y \in \mathcal{M}, y = x+s \\ \|s\| \leq \gamma, \\ s \in \mathrm{span}(\mathcal{M}^{(S)})}} \frac{1}{\gamma^2}(f(y) - f(x) - \langle y - x, \nabla f(x) \rangle),$$

$$D_{\|\cdot\|}^{\tau} = \max_{S \subset [n] \big| |S| = m} D_{\|\cdot\|}^{(S)}, \text{ and } L_{\|\cdot\|}^{\tau} = \max_{S \subset [n] | |S| = m} L_{\|\cdot\|}^{(S)}.$$

Theorem 6 (Delayed Updates as Approximate Oracle)**.** *For each norm $\|\cdot\|$ of choice, let $D^{\tau}_{\|\cdot\|}$ and $L^{\tau}_{\|\cdot\|}$ be defined above. Let the a random variable of delay be $\varkappa$ and let $\kappa := \mathbb{E}\varkappa$ be the expected delay from any worker. Moreover, assume that the algorithm drops any updates with delay greater than $k/2$ at iteration k. Then for the version of the algorithm without line-search, the delayed oracle will produce $s \in \mathcal{M}^{(S)}$ such that (5) holds with*

$$\delta = 4\kappa\tau L^{1}_{\|\cdot\|} D^{1}_{\|\cdot\|} D^{\tau}_{\|\cdot\|}/(C^{\tau}_{f}). \tag{8}$$

Furthermore, if we assume that there is a $\kappa_{\max}$ such that $\mathbb{P}(\varkappa \leq \kappa_{\max}) = 1$ for all k, then (5) holds with $\delta = c_{n,\tau\kappa_{\max}} \frac{4\tau L^{1}_{\|\cdot\|} D^{1}_{\|\cdot\|} \mathbb{E} D^{\varkappa\tau}_{\|\cdot\|}}{C^{\tau}_{f}}$ where

$$c_{n,\tau\kappa_{\max}} = \begin{cases} \frac{3\log n}{\log(n/(\tau\kappa_{\max}))} & \text{if } \kappa_{\max}\tau < n/\log n, \\ O(\log n) & \text{if } \kappa_{\max}\tau = O(n\log n), \\ \frac{(1+o(1))\tau\kappa_{\max}}{n} & \text{if } \kappa_{\max}\tau \gg n\log n. \end{cases} \tag{9}$$

The results above imply that AP-BCFW (without line-search) converges in both primal optimality and in duality gap according to Theorems 2 and 3 with the same $O(1/k)$ rate. Comparing to versions that solve (2) exactly, the delayed version has an additional additive factor in the numerator of form

$$4n\kappa\tau L^{1}_{\|\cdot\|} D^{1}_{\|\cdot\|} D^{\tau}_{\|\cdot\|} \quad \text{or} \quad O\left(\tau L^{1}_{\|\cdot\|} D^{1}_{\|\cdot\|} \mathbb{E} D^{\varkappa\tau}_{\|\cdot\|} \log n\right)$$

with the additional assumption that $\kappa_{\max} = O(n\log n/\tau)$.

Note that (8) depends on the expected delay rather than the maximum delay, and as $k \to \infty$ we allow the maximum delay to grow unboundedly. This allows the system to automatically deal with heavy-tailed delay distributions and sporadic stragglers. When we do have a small bounded delay, we produce stronger bounds (9) with a multiplier that is either a constant (when $\tau\kappa_{\max} = O(n^{1-\epsilon})$ for any $\epsilon > 0$), proportional to $\log n$ (when $\tau\kappa \leq n$) or proportional to $\frac{\tau\kappa_{\max}}{n}$ (when $\tau\kappa$ is large). The whole expression often has sublinear dependence on the expected delay κ. For instance, we prove in the appendix the following:

Lemma 7. *When $\|\cdot\|$ is Euclidean norm*

$$\mathbb{E} D^{\varkappa\tau}_{\|\cdot\|} \leq D^{\lceil \mathbb{E}\varkappa\tau \rceil}_{\|\cdot\|} \leq \sqrt{\lceil \mathbb{E}\varkappa \rceil} D^{\tau}_{\|\cdot\|}.$$

The bound is proportional to $\sqrt{\kappa}$ when $\kappa = \Omega(1)$. This is strictly better than Niu et al. (2011) which has quadratic dependence on $\kappa_{\max}$ and Liu et al. (2014) which has exponential dependence on $\kappa_{\max}$. Our mild $\kappa_{\max}$ dependence for the cases $\tau\kappa_{\max} > n$ suggests that the (9) remains proportional to $\sqrt{\kappa}$ even when we allow the maximum delay parameter to be as large as n/τ or larger without significantly affecting the convergence. Note that this allows some workers to be delayed for several data passes.

Observe that when $\tau = 1$, where the results reduces to a lock-free variant for BCFW, δ becomes proportional to $L^{1}_{\|\cdot\|}[D^{1}_{\|\cdot\|}]^2/C^{1}_{f}$. This is always greater than 1 (see e.g., Jaggi, 2013, Appendix D) but due to the flexibility of choosing the norm, this quantity corresponding to the most favorable norm is typically a small constant. For example, when f is a quadratic function, we show that $C^{1}_{f} = L^{1}_{\|\cdot\|}[D^{1}_{\|\cdot\|}]^2$ (see Appendix D.3). When $\tau > 1$, $\tau L^{1}_{\|\cdot\|} D^{1}_{\|\cdot\|} D^{\tau}_{\|\cdot\|}/C^{\tau}_{f}$ is often $O(\sqrt{\tau})$ for an appropriately chosen norm. Therefore, (8) and (9) are roughly in the order of $O(\kappa\sqrt{\tau})$ and $O(\sqrt{\kappa\tau})$ respectively[2].

Lastly, we remark that κ and τ are not independent. When we increase τ, we update the parameters less frequently and κ gets smaller. In a real distributed system, with constant throughput in terms of number of oracle solves per second from all workers. If the average delay is a fixed number in clock time specified by communication time. Then $\tau\kappa$ is roughly a constant regardless how τ is chosen.

4. Experiments

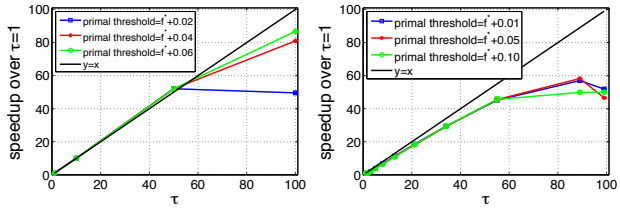
In this section, we experimentally demonstrate performance gains from the three key features of our algorithm: minibatches of data, parallel workers, and asynchrony.

4.1. Minibatches of Data

We conduct simulations to study the effect of mini-batch size τ, where larger τ implies greater degrees of parallelism as each worker can solve one or more subproblems in a mini-batch. In our simulation, we re-use the structural SVM setup from Lacoste-Julien et al. (2013) for a sequence labeling task on a subset of the OCR dataset (Taskar et al., 2004) ($n = 6251, d = 4082$). The dual problem has block-separable probability simplex constraint therefore allowing us to run AP-BCFW, and each subproblem can be solved efficiently using the Viterbi algorithm (more details are included in Appendix C). The speedup on this dataset is shown in Figure 2(a). For this dataset, we use $\lambda = 1$ with weighted averaging and line-search throughout (no delay is allowed). We measure the speedup for a particular $\tau > 1$ in terms of the number of iterations (Algorithm 1) required to converge relative to $\tau = 1$, which corresponds to BCFW. Figure 2(a) shows that AP-BCFW achieves linear speedup for mini-batch size up to $\tau \approx 50$. Further speedup is sensitive to the convergence criteria.

In our simulation for Group Fused Lasso, we generate a piecewise constant dataset of size ($n = 100, d = 10$, in Eq. 2) with Gaussian noise. We use $\lambda = 0.01$ and a primal suboptimality threshold as our convergence criterion. At each iteration, we solve τ subproblems (i.e. the mini-batch size). Figure 2(b) shows the speed-up over $\tau = 1$ (BCFW).

[2]For details, see our discussion in Appendix D.3



(a) Structural SVM (n=6251) (b) Group Fused Lasso (n=100)

Figure 2. Performance improvement with τ for (a) Structual SVM on the OCR dataset (Lacoste-Julien et al., 2013; Taskar et al., 2004) and (b) Group Fused Lasso on a synthetic dataset. f^* denotes primal optimum (the "primal" problem is actually referring to the dual problem in both cases). The performance metric here is the number of iterations to achieve ϵ-suboptimality.

Similar to the structural SVM, the speedup is almost perfect for small τ ($\tau \leq 55$) but tapers off for large τ to varying degrees depending on the convergence thresholds.

4.2. Shared Memory Parallel Workers

We implement AP-BCFW for the structural SVM in a multicore shared-memory system using the full OCR dataset ($n = 6877$). All shared-memory experiments were implemented in C++ and conducted on a 16-core machine with Intel(R) Xeon(R) CPU E5-2450 2.10GHz processors and 128G RAM. We first fix the number of workers at $T = 8$ and vary the mini-batch size τ. Figure 3(a) shows the absolute convergence (i.e. the convergence per second). We note that AP-BCFW outperforms single-threaded BCFW under all investigated τ, showing the efficacy of parallelization. Within AP-BCFW, convergence improves with increasing mini-batch sizes up to $\tau = 3T$, but worsens when $\tau = 5T$ as the error from the large mini-batch size dominates additional computation. The optimal τ for a given number of workers (T) depends on both the dataset (how "coupled" are the coordinates) and also system implementations (how costly is the synchronization).

Since speedup for a given T depends on τ, we search for the optimal τ across multiples of T to find the best speedup for each T. Figure 3(b) shows faster convergence of AP-BCFW over BCFW ($T = 1$) when $T > 1$ workers are available. It is important to note that the x-axis is *wall-clock time* rather than the number of epochs.

Figure 3(c) shows the speedup with varying T. AP-BCFW achieves near-linear speed up for smaller T. The speedup curve tapers off for larger T for two reasons: (1) Large T incurs higher system overheads, and thus needs larger τ to utilize CPU efficiently; (2) Larger τ incurs errors as shown in Fig. 2(a). If the subproblems were more time-consuming to solve, the affect of system overhead would be reduced. We simulate harder subproblems by simply solving them $m \sim$Uniform$(5, 15)$ times instead of just once. The speedup is nearly perfect as shown in Figure 3(d). Again, we observe that a more generous convergence threshold produces higher speedup, suggesting that resource scheduling could be useful (e.g., allocate more CPUs initially and fewer as algorithm converges).

We repeated the experiment on a larger synthetic dataset with $n = 103155, d = 4082$ created from the above mentioned OCR data as follows: for each of the 6877 words, generate 15 words with noisy images for characters, where the noise is introduced by flipping the bits of the images with probability 0.05 independently. The speedup with parallelization, shown in Figure 4, essentially follows the same pattern as it did in Figures 3c, 3b for original data.

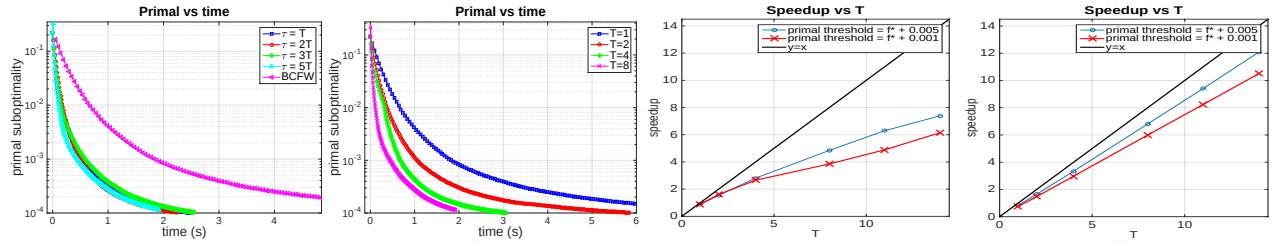
4.3. Performance gain with asynchronous updates

We compare AP-BCFW[3] with a synchronous version of the algorithm (SP-BCFW) where the server assigns τ/T subproblems to each worker, then waits for and accumulates the solutions before proceeding to the next iteration. We simulate workers of varying slow-downs in our shared-memory setup by assigning a *return probability* $p_i \in (0, 1]$ to each worker w_i. After solving each subproblem, worker w_i reports the solution to the server with probability p_i. Thus, a worker with $p_i = 0.8$ will drop 20% of the updates on average corresponding to 20% slow-down.

We use $T = 14$ workers for the experiments in this section. We first simulate the scenario with just one straggler with return probability $p \in (0, 1]$ while the other workers run at full speed ($p = 1$). Figure 5(a) shows that the average time per effective datapass (over 20 passes and 5 runs) of AP-BCFW stays almost unchanged with slowdown factor $1/p$ of the straggler, whereas it increases linearly for SP-BCFW. This is because AP-BCFW relies on the average available worker processing power, while SP-BCFW is only as fast as the slowest worker.

Next, we simulate a heterogeneous environment where the workers have varying speeds. While varying a parameter $\theta \in [0, 1]$, we set $p_i = \theta + i/T$ for $i = 1, \ldots, T$. Figure 5(b) shows that AP-BCFW slows down only by a factor of 1.4 compared to the no-straggler case. Assuming that the server and worker each take about half the (wall-clock) time on average per epoch, we would expect the run time to increase by 50% if the average worker speed halves, which is the case if $\theta = 0$ (i.e., $\frac{1}{\theta} \to \infty$). Thus, a factor of 1.4 is reasonable. The performance of SP-BCFW is almost identical to that in the previous experiment as its speed is determined by the slowest worker. Our experiments show that AP-BCFW is robust to stragglers and system heterogeneity.

[3]The version that has no delayed updates, but allows workers to asynchronously solve subproblems within each mini-batch.



(a) (b) (c) (d)

Figure 3. From left: **(a)** Primal suboptimality vs wall-clock time using 8 workers ($T = 8$) and various mini-batch sizes τ. **(b)** Primal suboptimality vs wall-clock time for varying T with best τ chosen for each T separately. **(c)** Speedup via parallelization with the best τ chosen among multiples of T ($T, 2T, ...$) for each T. **(d)** The same with longer subproblems.

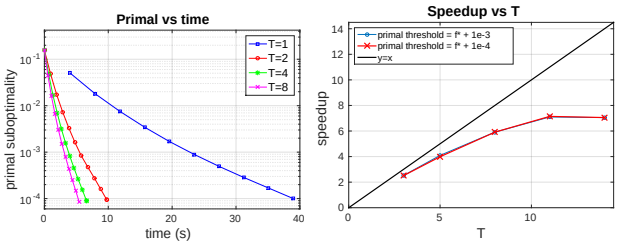


Figure 4. Speedup with parallelization on a synthetic OCR dataset. Left plot shows the decay of primal suboptimality and the right one shows the speedup.

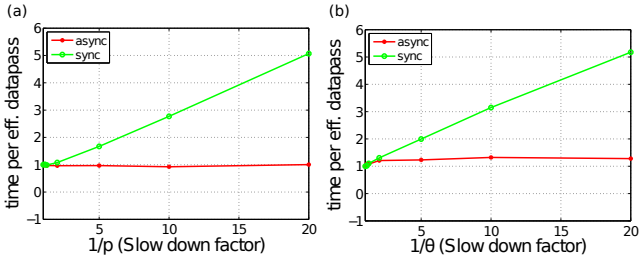


Figure 5. Average time per data pass in asynchronous and synchronous modes for two cases: one worker is slow with return probability p (left); workers have return probabilities (p_is) uniformly in $[\theta, 1]$ (right). Times normalized separately for AP-BCFW, SP-BCFW w.r.t. to where workers run at full speed.

4.4. Convergence under unbounded heavy-tailed delay

In this section, we illustrate the mild effect of delay on convergence by randomly drawing an independent delay variable for each worker. For simplicity, we use $\tau = 1$ (BCFW) on the group fused lasso problem from Section 4.1. We sample $\varkappa$ using either a Poisson distribution or a heavy-tailed Pareto distribution (round to the nearest integer). The Pareto distribution is chosen with shape parameter $\alpha = 2$ and scale parameter $x_m = \kappa/2$ such that $\mathbb{E}\varkappa = \kappa$ and $\mathrm{Var}\varkappa = \infty$. During the experiment, at iteration k, any updates that were based on a delay greater than $k/2$ are dropped (as our theory stipulates). The results are shown in Figure 6. Observe that for both cases, the impact of delay is rather mild. With expected delays up to 20, the algorithm only takes fewer than twice as many iterations to converge.

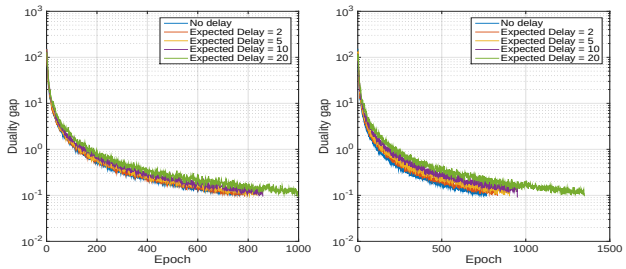


Figure 6. Illustrations of the convergence BCFW with delayed updates. On the left, we have the delay sampled from a Poisson distribution. The figure on the right is for delay sampled from a Pareto distribution. We run each problem until the duality gap reaches 0.1.

5. Conclusion

In this paper, we propose an asynchronous parallel generalization of the block-coordinate Frank-Wolfe method (Lacoste-Julien et al., 2013), analyze its convergence and provide intuitive conditions under which it has a provable speed-up over BCFW. We also show that the method is resilient to delayed updates in the distributed setting. The convergence bound depends only linearly on the expected delay and possibly sublinearly if the delay is bounded, yielding an exponential improvement over the dependence on the same parameter in parallel coordinate descent (Liu et al., 2014). The asynchronous updates allow our method to be robust to stragglers and node failure as the speed of AP-BCFW depends on *average* worker speed instead of the slowest. We demonstrate the effectiveness of the algorithm in structural SVM and Group Fused Lasso with both controlled simulation and real-data experiments on a multi-core workstation. For the structural SVM, it leads to a speed-up over the state-of-the-art BCFW by an order of magnitude using 16 parallel processors. As a projection-free FW method, we expect our algorithm to be very competitive in large-scale constrained optimization problems, especially when projections are expensive. Future work includes analysis for the strongly convex setting, the non-convex setting and ultimately releasing a general purpose software package for practitioners to deploy in Big Data applications.

Acknowledgements

We thank the AC and anonymous reviewers for helpful suggestions that led to significant improvement of the paper. YW was supported by NSF Award BCS-0941518 to CMU Statistics and a grant by Singapore NSF under its International Research Centre @ Singapore Funding Initiative administered by the IDM Programme Office. VS was partially supported by NSF grant DMS-1309174. WD was supported by DARPA XDATA FA87501220324. SS acknowledges partial support from NSF-IIS-1409802.

References

Ahipasaoglu, Damla S, Sun, Peng, and Todd, Michael J. Linear convergence of a modified frank–wolfe algorithm for computing minimum-volume enclosing ellipsoids. *Optimisation Methods and Software*, 23(1):5–19, 2008.

Alaíz, Carlos M, Barbero, Álvaro, and Dorronsoro, José R. Group fused lasso. In *Artificial Neural Networks and Machine Learning–ICANN 2013*, pp. 66–73. Springer, 2013.

Beck, Amir and Tetruashvili, Luba. On the convergence of block coordinate descent type methods. *SIAM Journal on Optimization*, 23(4):2037–2060, 2013.

Bellet, Aurélien, Liang, Yingyu, Garakani, Alireza Bagheri, Balcan, Maria-Florina, and Sha, Fei. Distributed Frank-Wolfe algorithm: A unified framework for communication-efficient sparse learning. *CoRR*, abs/1404.2644, 2014.

Bleakley, Kevin and Vert, Jean-Philippe. The group fused Lasso for multiple change-point detection. arXiv preprint arXiv:1106.4199, 2011.

Bredies, Kristian, Lorenz, Dirk A, and Maass, Peter. A generalized conditional gradient method and its connection to an iterative shrinkage method. *Computational Optimization and Applications*, 42(2):173–193, 2009.

Clarkson, Kenneth L. Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm. *ACM Transactions on Algorithms (TALG)*, 6(4):63, 2010.

Collins, Michael, Globerson, Amir, Koo, Terry, Carreras, Xavier, and Bartlett, Peter L. Exponentiated gradient algorithms for conditional random fields and max-margin markov networks. *JMLR*, 9:1775–1822, 2008.

Dai, Wei, Wei, Jinliang, Zheng, Xun, Kim, Jin Kyu, Lee, Seunghak, Yin, Junming, Ho, Qirong, and Xing, Eric P. Petuum: a framework for iterative-convergent distributed ML. *arXiv:1312.7651*, 2013.

Fercoq, Olivier and Richtárik, Peter. Accelerated, parallel, and proximal coordinate descent. *SIAM Journal on Optimization*, 25(4):1997–2023, 2015.

Foygel, Rina, Horrell, Michael, Drton, Mathias, and Lafferty, John D. Nonparametric reduced rank regression. In *NIPS'12*, pp. 1628–1636, 2012.

Frank, Marguerite and Wolfe, Philip. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.

Freund, Robert M. and Grigas, Paul. New analysis and results for the frank–wolfe method. *Mathematical Programming*, 155(1):199–230, 2014. ISSN 1436-4646.

Fujishige, Satoru and Isotani, Shigueo. A submodular function minimization algorithm based on the minimum-norm base. *Pacific Journal of Optimization*, 7(1):3–17, 2011.

Garber, Dan and Hazan, Elad. A linearly convergent conditional gradient algorithm with applications to online and stochastic optimization. *arXiv:1301.4666*, 2013.

Harchaoui, Zaid, Juditsky, Anatoli, and Nemirovski, Arkadi. Conditional gradient algorithms for norm-regularized smooth convex optimization. *Mathematical Programming*, 152(1-2):75–112, 2015. ISSN 0025-5610.

Hazan, Elad and Kale, Satyen. Projection-free online learning. In *ICML'12*, 2012.

Jaggi, Martin. *Sparse convex optimization methods for machine learning*. PhD thesis, Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 20013, 2011, 2011.

Jaggi, Martin. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In *ICML'13*, pp. 427–435, 2013.

Jegelka, Stefanie, Bach, Francis, and Sra, Suvrit. Reflection methods for user-friendly submodular optimization. In *NIPS'13*, pp. 1313–1321, 2013.

Lacoste-Julien, Simon and Jaggi, Martin. On the global linear convergence of Frank-Wolfe optimization variants. In *NIPS'15*, pp. 496–504, 2015.

Lacoste-Julien, Simon, Jaggi, Martin, Schmidt, Mark, and Pletscher, Patrick. Block-coordinate Frank-Wolfe optimization for structural svms. In *ICML'13*, pp. 53–61, 2013.

Lafond, Jean, Wai, Hoi-To, and Moulines, Eric. Convergence analysis of a stochastic projection-free algorithm. *arXiv:1510.01171*, 2015.

LeBlanc, Larry J, Morlok, Edward K, and Pierskalla, William P. An efficient approach to solving the road network equilibrium traffic assignment problem. *Transportation Research*, 9(5):309–318, 1975.

Lee, Seunghak, Kim, Jin Kyu, Zheng, Xun, Ho, Qirong, Gibson, Garth A, and Xing, Eric P. On model parallelization and scheduling strategies for distributed machine learning. In *NIPS'14*, pp. 2834–2842, 2014.

Li, Mu, Zhou, Li, Yang, Zichao, Li, Aaron, Xia, Fei, Andersen, David G, and Smola, Alexander. Parameter server for distributed machine learning. In *NIPS Workshop: Big Learning*, 2013.

Liu, Ji, Musialski, Przemyslaw, Wonka, Peter, and Ye, Jieping. Tensor completion for estimating missing values in visual data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(1):208–220, 2013.

Liu, Ji, Wright, Stephen J, Ré, Christopher, and Bittorf, Victor. An asynchronous parallel stochastic coordinate descent algorithm. *JMLR*, 2014.

Mitzenmacher, Michael. The power of two choices in randomized load balancing. *Parallel and Distributed Systems, IEEE Transactions on*, 12(10):1094–1104, 2001.

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

Niu, Feng, Recht, Benjamin, Ré, Christopher, and Wright, Stephen J. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS'11*, pp. 693–701, 2011.

Ouyang, Hua and Gray, Alexander G. Fast stochastic Frank-Wolfe algorithms for nonlinear SVMs. In *SDM*, 2010.

Qu, Zheng and Richtárik, Peter. Coordinate descent with arbitrary sampling ii: Expected separable overapproximation. *arXiv preprint arXiv:1412.8063*, 2014.

Raab, Martin and Steger, Angelika. Balls into bins - a simple and tight analysis. In *Randomization and Approximation Techniques in Computer Science*, pp. 159–170. Springer, 1998.

Richtárik, Peter and Takáč, Martin. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, pp. 1–52, 2015.

Taskar, Ben, Guestrin, Carlos, and Koller, Daphne. Max-margin Markov networks. In *NIPS'04*, pp. 25–32. MIT Press, 2004.

Tsitsiklis, John N, Bertsekas, Dimitri P, Athans, Michael, et al. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Transactions on Automatic Control*, 31(9), 1986.

Yu, Chun-Nam John and Joachims, Thorsten. Learning structural svms with latent variables. In *ICML'09*, pp. 1169–1176. ACM, 2009.

Zhang, Xinhua, Yu, Yaoliang, and Schuurmans, Dale. Accelerated training for matrix-norm regularization: A boosting approach. In *NIPS'12*, pp. 2915–2923, 2012.

Zhang, Xinhua, Yu, Yao-Liang, and Schuurmans, Dale. Polar operators for structured sparse estimation. In *NIPS'13*, pp. 82–90, 2013.