Asynchronous Parallel Block-Coordinate Frank-Wolfe

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@gmail.com
Eric P. Xing†
epxing@cs.cmu.edu
† Machine Learning Department, Carnegie Mellon University, Pittsburgh, PA 15213

Abstract

We develop mini-batched parallel Frank-Wolfe (conditional gradient) methods for smooth convex optimization subject to block-separable constraints. Our work includes the basic (batch) Frank-Wolfe algorithm as well as the recently proposed Block-Coordinate Frank-Wolfe (BCFW) method [22] as special cases. Our algorithm permits asynchronous updates within the minibatch, and is robust to stragglers and faulty worker threads. Our analysis reveals how the potential speedups over BCFW depend on the minibatch size and how one can provably obtain large problem dependent speedups. We present several experiments to indicate empirical behavior of our methods, obtaining significant speedups over competing state-of-the-art (and synchronous) methods on structural SVMs.

1 Introduction

The classical Frank-Wolfe algorithm [31] has recently witnessed a surge of interest in machine learning [7, 15, 16, 2]. Its wide applicability and practical appeal have spurred several extensions, including to regularized optimization [34, 6, 13], to linearly convergent algorithms [21, 12], to stochastic/online versions [28, 14], and to a powerful randomized block-coordinate version [22].

Motivated by [22], we consider FW methods for the following convex optimization problem

\[
\min_x f(x) \quad \text{s.t. } x = [x_1, \ldots, x_n] \in \mathcal{M}_1 \times \cdots \times \mathcal{M}_n =: \mathcal{M}
\]

(1.1)

where each \( \mathcal{M}_i \subset \mathbb{R}^{m_i} \) \( (1 \leq i \leq n) \) is a compact convex set. Such Cartesian product constraints arise in several problems, notably the structured SVM dual [22], routing problems [23], dual of the group fused-lasso [1, 5], nearest point projection onto convex sets [17], among others.

A typical approach to solving (1.1) is to use a block-coordinate descent method, which minimizes \( f(x) \) with respect to a block of variables at each iteration; a simpler approach forms a local quadratic model of \( f(x) \) at each iteration and essentially solves a projection subproblem over some block [26, 29, 3]. However, sometimes solving a projection subproblem can be very expensive, e.g., submodular minimization [11], or even computationally intractable [8]. Frank-Wolfe (FW) methods often shine in such scenarios because in contrast to (quadratic) projection based techniques at each iteration they only require optimizing a linear objective, namely by solving a problem of the form

\[
\min_s \langle s, \nabla f(\cdot) \rangle \quad \text{s.t. } s \in \mathcal{M}.
\]

(1.2)
This simpler structure can bestow great computational advantages; moreover, it enables another practical advantage of FW methods: sparsity or low-rank of intermediate iterates [7, 16].

For $\mathcal{M} = \prod_i \mathcal{M}_i$, problem (1.2) decomposes into $n$ independent subproblems

$$\min_{s(i) \in \mathcal{M}_i} \langle s(i), \nabla f(x(i)) \rangle, \quad 1 \leq i \leq n,$$

where $x(i)$ denotes the $i$th block of coordinates in $x$. Clearly, these $n$ subproblems can be solved in parallel (an idea that dates back to at least 1975 [23]), potentially allowing large-scale problems. However, updating all the coordinates at each iteration (whether in serial or parallel) is often expensive, which handicaps the applicability of FW to high-dimensional big-data problems, e.g., large-scale nonparametric problems 2. This drawback was partially ameliorated by the recent randomized Block-Coordinate Frank-Wolfe (BCFW) method of [22]. They propose randomly selecting a block $\mathcal{M}_i$ of coordinates at each iteration and performing FW updates with it. However, it is a strictly sequential procedure that cannot easily take advantage of modern multicore CPU architecture or of high-performance clusters to address problems at an even larger scale.

In light of the above, our paper makes the following key contributions:

- A parallel (and distributed) block-coordinate Frank-Wolfe algorithm that allows asynchronous computation within each minibatch. The algorithm is robust to stragglers and faulty worker threads.
- An analysis of the primal and primal-dual convergence of our asynchronous parallel BCFW algorithm and its variants for any minibatch size.
- It presents 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 a real system how our algorithm solves a structural SVM problem several times faster than the state-of-the-art.

Thus, our results contribute to making FW algorithms more attractive for big-data applications. To lend further perspective, we mention directly related work below—due to space paucity we cannot offer a comprehensive summary and refer the reader to [16, 33, 22] and references therein.

**Block coordinate Frank-Wolfe and Structural SVM.** Our algorithm extends and generalizes BCFW to minibatch or parallel computation. Our convergence analysis makes use of their proof structure but with carefully chosen different stepsizes. Essentially, our results contain BCFW as a special case. A large portion of [22] is to provide a more explicit (and stronger) guarantee for BCFW on structural SVM. While we mainly focus on a more general class of problems, the particular subroutine that the structural SVM uses requires special treatment; we discuss it in a separate section in the supplement.

**Parallelization of sequential algorithms.** The idea of parallelizing sequential optimization algorithms is not new. It dates back to [30] for stochastic gradient methods, and more recently [29, 25] studied parallelization of coordinate descent algorithms. The conditions that these parallel coordinate descent methods succeed, e.g., expected separable overapproximation (ESO), and coordinate Lipschitz condition are closely related to our conditions in Section 2.2, but are not exactly the same due to the difference in the way solutions are updated and the subproblems that arise.

**Asynchronous algorithms.** Conceptually, our asynchronous scheme is different from both [27] and [25] in that we do not introduce any errors and inconsistency into the algorithm. The price to pay is that our asynchronous updates are restricted to within each minibatch.

2 when the number of parameters scale with number of data points
Others related work. Lastly, while preparing our manuscript, we discovered the recent arXiv paper by Bellet et al. [4] which also studies distributed Frank-Wolfe. We note that [4] focuses on Lasso type problems and communication costs, hence is not directly comparable to our results.

Notation. Before proceeding with the rest of the paper, we briefly summarize our notation here. The vector \( x \in \mathbb{R}^m \) denotes the parameter vector, possibly split into \( n \) coordinate blocks. For block \( i = 1, \ldots, n \), \( E_i \in \mathbb{R}^{m \times m_i} \) is the coordinate 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 \( \mathbb{R}^{m_i} \rightarrow \mathbb{R}^m \), thus \( x[i] = E_i^* x(i) \) (note the subscript \( x[i] \)). We denote the size of a minibatch by \( \tau \), and the number of parallel workers (threads) by \( T \). Unless otherwise stated, \( k \) denotes the iteration/epoch counter and \( \gamma \) denotes a stepsize. Finally, \( C_f^\tau \) (and other such constants) denotes some curvature measure that is important in our convergence proof; this will be described in greater detail in the main text.

2 Algorithm

In this section, we develop and analyze an asynchronous parallel block-coordinate Frank-Wolfe algorithm, hereafter \( \text{Ap-Bcfw} \), to solve (1.1).

Our algorithm is designed for a shared-memory multicore architecture. 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 gradient. This setup matches the system assumptions in [27, 29, 25], and most modern multicore machines permit such an arrangement.

At a high-level, \( \text{Ap-Bcfw} \) may be viewed as randomized mini-batch version of BCFW, so that at each iteration it processes a randomly chosen set of \( 1 \leq \tau \leq n \) coordinate blocks; the case \( \tau = 1 \) corresponds to BCFW while \( \tau = n \) corresponds to standard (batch) FW. However, to handle the mini-batching, \( \text{Ap-Bcfw} \) divides the workload of \( \tau \) coordinates across \( T \) worker threads that operate asynchronously. This helps avoid having to wait for the slowest workers, and allows fast workers (threads) to process more than one block. Ideally, if we choose \( \tau \) to be larger (say, twice or more) than \( T \), the computational workload will be well-balanced and the runtime of each iteration depends more on the average worker than on the worst one.

The above scheme is made explicit by the pseudocode in Algorithm 1.

The two most important questions pertaining to Algorithm 1 are:

- Does it converge?
- If so, then how fast? And how much faster is it compared to BCFW (\( \tau = 1 \))?

A third question that we also ask is whether \( \text{Ap-Bcfw} \) can be implemented in a distributed system instead of a shared-memory architecture.

We answer the first two questions in Sections 2.1 and 2.2. Specifically, we show \( \text{Ap-Bcfw} \) converges at the familiar \( O(1/k) \) rate. Our analysis reveals that the speed-up of \( \text{Ap-Bcfw} \) over BCFW is problem dependent. Intuitively, we show that the extent that mini-batching (\( \tau > 1 \)) can speed up convergence depends on the the average “coupling” of the objective function \( f \) across different coordinate blocks. For example, we show that if \( f \) has a block symmetric diagonally dominant Hessian, then \( \text{Ap-Bcfw} \) converges \( \tau/2 \) times faster. We address the third question in Section 2.4, where we describe how \( \text{Ap-Bcfw} \) can be made to work in a distributed setting and analyze the case when the workers are not allowed to communicate with each other at all.
Algorithm 1: AP-BFW: Asynchronous Parallel Block-Coordinate Frank-Wolfe

**Input:** An initial feasible \( x(0) \), mini-batch size \( \tau \), number of workers \( T \).

**for** \( \tilde{k} = 1,2,... \) (\( \tilde{k} \) is the epoch.) **do**

1. Randomly pick \( S \subset [n] \) such that \(|S| = \tau\).
2. Assign index set \( \tilde{S} = S \).

**while** \( \tilde{S} \) is not empty **do**

   1. If any worker \( i \in \{1,...,T\} \) is idle **then**
      
      a. Worker \( i \) chooses \( j \in \tilde{S} \) and solves (1.3) “approximately” for \( s(j) \).
      
      b. (Optionally) determine the line-search stepsizes \( \gamma_i \).
      
      c. Remove \( j \) from set \( \tilde{S} \).
   **end if**

3. Set stepsize \( \gamma = \frac{2n\tau}{\tau^2k+2n} \).

4. Update \( x(k+1) = x(k) + \gamma \sum_{i \in S} (s_i - x_i(k)) \).

5. (Optionally) Compute \( x^{(k+1)}_{\text{line-search}} = x(k) + \sum_{i \in S} \gamma_i (s_i - x_i(k)) \).

   Update \( x^{(k+1)} \) to \( x^{(k+1)}_{\text{line-search}} \) if line-search improves the objective value.

   **if** converged **then**
   
   break
   **end if**

**end while**

**end for**

**Output:** limit point \( \bar{x} \).

### 2.1 Main convergence results

Before stating the results, we need to define a few quantities. The first key quantity—also key to the analysis of several other FW methods—is the notion of curvature. Since AP-BFW updates a subset of coordinate blocks at a time, we define set curvature for an index set \( S \subseteq [n] \) as

\[
C_f(S) := \sup_{x \in M_{S}, y(S) \in M(f), \gamma \in [0,1], y = x + \gamma (y(S) - x(S)), \nabla_S f(x)} \frac{2}{\gamma^2} (f(y) - f(x) - \langle y(S) - x(S), \nabla_S f(x) \rangle) \tag{2.1}
\]

For index sets of size \( \tau \), we define the expected set curvature over a uniform choice of subsets as

\[
C_f^\tau := \mathbb{E}_{S:|S|=\tau} [C_f(S)] = \left( \frac{n}{\tau} \right)^{-1} \sum_{S \subseteq [n], |S| = \tau} C_f(S). \tag{2.2}
\]

These curvature definitions are closely related to the local curvature constant \( C_f \) of Jaggi [16] and the coordinate curvature \( C_f^{(i)} \) and product curvature \( C_f^{\otimes} \) of Lacoste-Julien et al. [22]. Lemma 1 makes this relation more precise.

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

1. \( C_{f}^{(i)} \leq C_f^{(S)} \leq C_f \);

2. \( \frac{1}{n} C_f^{\otimes} = C_f \leq C_f^\tau \leq C_f^n = C_f \).

The way the average set curvature \( C_f^\tau \) scales with \( \tau \) is critical for bounding the amount of speedup we can expect over BCFW; we provide a detailed analysis of this speedup in Section 2.2.
The next key object an approximate linear minimizer. At iteration $k$, as in [16, 22], we also allow the core computational subroutine that solves (1.3) to yield an approximate minimizer $s(i)$ (up to an additive constant $\delta \geq 0$) that satisfies the inequality

$$
\langle s(i), \nabla f(x) \rangle \leq \min_{s'(i) \in \mathcal{M}(i)} \langle s'(i), \nabla f(x^{(k)}) \rangle + \frac{\delta \gamma_k C^2_f}{2\tau}.
$$

(2.3)

With definitions (2.2) and (2.3) in hand, we are ready to state our first main convergence result.

**Theorem 1** (Primal Convergence). Suppose we employ a linear minimizer that satisfies (2.3) when solving the subproblem (1.3). Then, for each $k \geq 0$, the iterations in Algorithm 1 and its line search variant (Steps 2b and 5) obey

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

where the constant $C = nC^2_f (1 + \delta) + f(x^{(0)}) - f(x^*)$.

At a first glance, the $n^2 C^2_f$ term in the numerator might seem bizarre since it depends quadratically in the data size, but as we will see in the next section, $C^2_f$ can be as small as $O(\frac{1}{\tau^2})$. This is the scale of the constant one should keep in mind to compare the rate to other methods, e.g. coordinate descent.

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 [16]; we define this as

$$
g(x) := \max_{s \in \mathcal{M}} \langle x - s, \nabla f(x) \rangle = \sum_{i=1}^{n} \max_{s(i) \in \mathcal{M}(i)} \langle x(i) - s(i), \nabla f(x) \rangle =: \sum_{i=1}^{n} g^{(i)}(x). \quad (2.4)
$$

To see why (2.4) 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

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

This duality gap is obtained for “free” in batch Frank-Wolfe, but not in BCFW or AP-Bcfw. Here, we only have an unbiased estimator $\hat{g}(x) = \frac{1}{|S|} \sum_{i \in S} g^{(i)}(x)$. As $\tau$ gets large, $\hat{g}(x)$ is close to $g(x)$ with high probability (McDiarmid’s Inequality), and can still be useful as a stopping criterion.

**Theorem 2** (Primal-Dual Convergence). Suppose we run Algorithm 1 and its line search variant up to $K$ iterations, let $g_k := E\hat{g}(x^{(k)})$ and $K \geq 1$, then there exists at least one $k^* \in [1, ..., K]$ such that the expected surrogate duality gap satisfies

$$
g_k \leq \hat{g}_k \leq \frac{6nC}{\tau^2(K + 1)},
$$

where $C$ is as in Theorem 1 and $\hat{g}_k$ is the weighted average $\frac{2}{K(K + 1)} \sum_{k=1}^{K} kg_k$.

We defer the proofs of the two theorem to the appendix and focus on discussing their consequences.

**Relation with FW and BCFW.** The above convergence guarantees can be thought of as an interpolation between BCFW and batch FW. If we take $\tau = 1$, this gives exactly
the convergence guarantee for BCFW [22, 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 it reduces to the classic batch guarantee as in [16].

**Weaker dependence on initialization.** Unlike classic FW, the convergence rate for our method depends on the initialization. When \( h_0 := f(x^{(0)}) - f(x^*) \geq nC_f^\tau \), the convergence is slower by a factor of \( \frac{n}{\tau} \). The same concern was also raised in [22] 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 \( C_f^\tau \) increases with \( \tau \), so the fast convergence region becomes larger when we increase \( \tau \). In addition, if we pick \( \tau^2 > n \), the rate of convergence is not affected by initialization anymore.

**Stopping condition.** A drawback of BCFW over FW is that we no longer have the duality gap \( g_k \) at every iteration for free. We can however estimate it by \( \hat{g}_k = \frac{1}{n} \sum_{i \in S} g_k^{(i)} \). As \( \tau \) increases, \( \hat{g}_k \) converges to \( g_k \). In addition, the plug-in estimator of \( \hat{g}_k \) is to simply replace \( g_k \) with \( \hat{g}_k \) in the same weighted average. It seems intuitively true that the sequence converges to \( g_k \) with rate \( O_p(\frac{1}{\sqrt{\tau K}}) \), and this essentially provides a practically useful stopping condition “for free” (since one can update the estimate of \( \hat{g}_k \) incrementally). We will leave the formal proof of this conjecture to future work.

**Speedup.** The careful reader may have noticed the \( n^2C_f^\tau \) 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 \( \tau = 1 \), \( C_f^\tau \) is as small as \( O(n^{-2}) \) (see [22, Lemmas A1 and A2]), and it is easy to check that the dependence in \( n \) is the same even for \( \tau > 1 \). What really matters is how much speedup one can achieve over BCFW, and this speedup critically relies on how \( C_f^\tau \) depends on \( \tau \). Analyzing this dependence will be our main focus in the next section.

### 2.2 Effect of parallelism / mini-batching

To understand when mini-batching is meaningful and quantify its speedup, we take a more careful look at the expected set curvature \( C_f^\tau \) in this section. In particular, we analyze and present a set of insightful conditions that govern its relationship with \( \tau \). The key idea is to roughly 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 \( x, y \in M \)

\[
  f(y) \leq f(x) + (y - x, \nabla f(x)) + (y - x)^T H (y - x).
\]

(2.5)

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 \( H \in \{ K \mid K \succeq \nabla^2 f(x), \ \forall x \in M \} \).

Since \( x = [x_1, ..., x_n] \) (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 \( x_i^T H_{ij} x_j \) makes sense. Now, we define a *boundedness* parameter \( B_i \) for every \( i \), and an *incoherence condition* with parameter \( \mu_{ij} \) for every block coordinate pair \( M_i, M_j \) such that

\[
  B_i := \sup_{x_i \in M_i} x_i^T H_{ii} x_i, \quad \mu_{ij} := \sup_{x_i \in M_i, x_j \in M_j} x_i^T H_{ij} x_j, \quad \mu := \mathbb{E}_{(i,j) \sim \text{Unif}([n]^2)} B_i, \quad \mu := \mathbb{E}_{(i,j) \sim \text{Unif}([n]^2)} \mu_{ij}.
\]

(2.6)

(2.7)

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

**Theorem 3.** If problem (1.1) obeys \( B \)-expected boundedness and \( \mu \)-expected incoherence. Then,

\[
  C_f^\tau \leq 4(\tau B + \tau(\tau - 1)\mu) \quad \text{for any} \quad \tau = 1, ..., n.
\]

(2.8)
It is clear that when the incoherence term $\mu$ is large, the expected set curvature $C^T_j$ is proportional to $\tau^2$, and when $\mu$ is close to 0, then $C^T_j$ is proportional to $\tau$. In other words, when the interaction between coordinates block is small, one would gain from parallelizing the block-coordinate Frank-Wolfe. This is analogous to the situation in parallel coordinate descent [29, 25] and we will compare the rate of convergence explicitly with them in the next section.

**Remark 1.** Let us form a matrix $M$ with $B_i$ on the diagonal and $\mu_{ij}$ 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 $C^T_j$ is proportional to $\tau$. SDD matrices are a common class of matrices whose computational properties have been well-studied recently (see e.g., [20, 18]).

The above result depends on the parameters $B$ and $\mu$. We now derive now specific instances of the above results for structural SVM and Group Fused Lasso—details may be found in the appendix. For structural SVM, a simple generalization of Lacoste-Julien et al. [22, Lemmas A.1, A.2] (details in the appendix) shows that in the worst case, using $\tau > 1$ offers no gain at all. Fortunately, if we are willing to consider a more specific problem and consider the average case instead, using larger $\tau$ does make the algorithm converge faster (and this is the case according to our experiments).

**Example 1** (Structural SVM for multi-label classification (with random data)). According to Yu and Joachims [32], the compatibility function $\phi(x, y)$ for multiclass classification will be $[0, \ldots, 0, x^T, 0, \ldots, 0]^T$ where the only nonzero block that we fill with the feature vector is the $(y)$th block. So $\psi_i(x, j) = \phi(x, y_i) - \phi(x, j)$ looks like $[0, \ldots, 0, x^T, 0, \ldots, 0, -x^T, 0, \ldots, 0]^T$. This already ensures that $B = \frac{C}{\tau^2}$. Suppose we have $K$ classes and each class has a unique feature vector drawn randomly from a unit sphere in $\mathbb{R}^d$; furthermore, for simplicity assume we always draw $\tau < K$ data points with $\tau$ distinct labels$^3$ $\mu \leq \sqrt{\frac{C \log d}{d - n^2 \lambda}}$, for some constant $C$. In addition, if $d \geq \tau^2 \sqrt{C \log d}$, then with high probability

$$C^T_j \leq \frac{2\tau + 2\tau^2}{n^2 \lambda} \leq \frac{C\tau}{n^2 \lambda},$$

which yields a convergence rate $O(\frac{R^2}{\tau^2})$, where $R := \max_{i \in [n], y \in Y_i} \|\psi_i(y)\|_2$ using notation from Lemmas A.1 and A.2 of Lacoste-Julien et al. [22].

This analysis suggests that a good rule-of-thumb is that we should choose $\tau$ to be at most the number of categories for the classification. If each class is a mixture of random draws from the unit sphere, then we can choose $\tau$ to be the underlying number of mixture components.

**Example 2** (Group Fused Lasso). The Group Fused Lasso aims to solve (typically for $q = 2$)

$$\min_X \frac{1}{2}\|X - Y\|_F^2 + \lambda\|XD\|_{1,q}, \quad q > 1, \quad (2.9)$$

where $X, Y \in \mathbb{R}^{d \times n}$, and column $y_t$ of $Y$ is an observed noisy $d$-dimensional feature vector at time $1 \leq t \leq n$. The matrix $D \in \mathbb{R}^{n \times (n-1)}$ is the differencing matrix that takes the difference of feature vectors at adjacent time points (columns). The formulation aims to filter the trend that has some the piecewise constant structures. The dual to (2.9) is

$$\max_U -\frac{1}{2}\|UD^T\|_{F^2} + \text{tr}UD^TY^T \quad \text{s.t.} \quad \|U_{t,t}\|_F \leq \lambda, \quad \forall t = 1, \ldots, n - 1,$$
where \( p \) is conjugate to \( q \), i.e., \( 1/p + 1/q = 1 \). This block-constrained problem fits our structure (1.1). For this problem, we find that \( B \leq 2\lambda^2d \) and \( \mu \leq \lambda^2d \), which yields the bound

\[
C_f^\tau \leq 4\tau\lambda^2d.
\]

Consequently, the rate of convergence becomes \( O(\frac{n^2\lambda^2d}{\tau^2}) \). In this case, batch FW will have a better rate of convergence than BCFW. 

### 2.3 Comparison to parallel block coordinate descent

With some understanding on \( C_f^\tau \), we can now explicitly compare the rate of convergence in Theorem 1 with parallel BCD [29, 25] under the assumption of \( \mu = O(B/\tau) \) — a fair and equally favorable case to all these methods. To facilitate comparison, we will convert the constants in all three methods to block coordinate gradient Lipschitz constant \( L_i \), which obeys

\[
f(x + s_{[i]}) \leq f(x) + \langle s_{[i]}, \nabla f(x) \rangle + L_i \| s_{[i]} \|^2,
\]

for any \( x \in \mathcal{M}, s_{[i]} \in \mathcal{M}_i \).

Apparently, \( B_i \leq 4L_i \text{diam}(\mathcal{M}_i)^2 = L_i \max_{x_i, x^*_i \in \mathcal{M}_i} \| x_i - x^*_i \|^2 \), so

\[
B \leq \frac{1}{n} \sum_i L_i \max_{x_i, x^*_i} \| x_i - x^*_i \| \leq \frac{1}{n} \sum_i L_i \max_x \| x - x^* \|^2 = \mathbb{E}_i(L_i)R^2
\]

where \( R := \max_x \| x - x^* \|. \) The rate of convergence for the three methods (with \( \tau \) oracle calls considered as one iteration) are given below.

|       | AP-BCFW (Ours) | P-BCD [29, Theorem 19] | AP-BCD [25, Theorem 3] |
|-------|----------------|------------------------|------------------------|
| Rate  | \( O_p \left( \frac{n\mathbb{E}_i(L_i)R^2}{\tau^2} \right) \) | \( O_p \left( \frac{n\mathbb{E}_i(L_i)R^2}{\tau^2} \right) \) | \( O_p \left( \frac{n\max_i L_i R^2}{\tau^2} \right) \) |

The comparison illustrates that these methods have the same \( O(1/k) \) rate and almost the same dependence on \( n \) and \( \tau \) despite the fact that we use a much simpler linear oracle. Nothing comes for free though, Nesterov acceleration does not apply for Frank-Wolfe based methods in general, while a careful implementation of parallel coordinate descents can achieve \( O(1/k^2) \) rate without any full-vector interpolation in every iteration [10]. Also, Frank-Wolfe methods usually need additional restrictive conditions or algorithmic steps to get linear convergence for strongly convex problems.

These facts somewhat limits the applicability of our method to cases when projection can be computed as efficiently as (1.3). However, as is surveyed in Jaggi [16], there are many interesting cases when (1.3) is much cheaper than projections, e.g., projection onto a nuclear norm ball takes \( O(n^3) \) while (1.3) takes only \( O(n^2) \).

### 2.4 Towards distributed computation

The previous section demonstrates that for highly incoherent problems, we can use a much larger \( \tau \) than the number of threads in a single computer to further speed convergence. To take advantage of this scenario, we design a scheme for distributed computation below. We operate in a setup that relies on a centralized parameter server [24, 9] and \( T \) workers (running at different speeds).

---

4Observe that \( C_f^\tau \) does not have an \( n^2 \) term in the denominator to cancel out the numerator. This is because the objective function is not appropriately scaled with \( n \) like it does in the structural SVM formulation.
Parameter server

- Receiver worker updates $\Delta x_i$.
- Discard conflicts if receiving more than one update for the same block.
- Update and broadcast $x^{k+1}$ when $\tau$ distinct block-coordinate updates are received.

Worker nodes

- Check network buffer for parameter updates.
- Pick $i$ randomly from $1, \ldots, n$, solve the subroutine (2.3) for $i$th block.
- Send $\Delta(x_i)$ with iteration num $k$ to the server.

Table 1: Distributed AP-BCFW: Server and worker responsibilities.

First we note that this has essentially the same parameter updates as in Algorithm 1, so the same convergence analysis in Section 2.1 applies. The key difference from the shared memory setting is that the workers are allowed to work independently for a period of time without querying “What’s next?” hence avoiding unnecessary communications. Not surprisingly, in the new scheme, different workers may end up working on the same coordinate block, but the following proposition shows that for a large range of $\tau$, this is actually not so bad. The proof is given in the appendix.

**Proposition 1.** In the distributed asynchronous update scheme above:

i) The expected number of subroutine calls to complete each iteration is $\tau + \sum_{i=1}^{\tau-1} \frac{i}{n-i}$.

ii) If $0.02n < \tau < 0.6n$, with probability at least $1 - \exp(-n/60)$, no more than $2\tau$ random draws (2$\tau$ subroutine calls in total from all workers) suffice to complete each iteration.

Another key resource for distributed computation is communication. We note that the only communication needed in our scheme is for the server to communicate random coordinate blocks of the gradient vector to each worker and for the workers to send specific blocks of the updates to the server. This is arguably the minimum amount of communication one would expect for this type of problems.

## 3 Experiments

In this section, we experimentally demonstrate the performance gains of the two key features of our algorithm: parallelism and asynchronous nature.

### 3.1 Performance gain due to minibatch size $\tau$

We first describe our experiments to show the performance gain due to $\tau$. We begin with a simulation: we generate a small piecewise constant dataset of size $(n=100, d=10)$ with Gaussian noise for Group fused lasso. We use $\lambda = 0.01$ and a primal suboptimality threshold as our convergence criterion. At each iteration, we solve $\tau$ subproblems. This simulates the parallel setting where we have $\tau$ workers and each worker solves one subproblem. We measure the speedup for a particular $\tau > 1$ by comparing it with number of iterations required for the case $\tau = 1$, which corresponds to BCFW. Figure 1 (left) shows that the speedup is almost ideal until a reasonably large value for $\tau$. We ran a similar experiment on a part of the OCR dataset [19] $(n = 6251, d = 4082)$, which is a sequence labeling task where the subproblem

---

5For structural SVM, the scheme is slightly modified to cater to the special subroutine, which requires to communicate the primal variable instead (see Appendix C).
can be solved using the Viterbi algorithm. The speedup on this dataset is shown in Figure 1 (right). For this dataset, we use $\lambda = 1$, turn on weighted averaging and line-search throughout. The results are similar. However, we observed that the speedup gets worse if $\tau$ is too large as there are too many stale updates based on reference iterates on the workers.

![Speedup on Group fused lasso](a) Group Fused Lasso (n=100)

![Speedup on OCR dataset](b) Structural SVM on OCR dataset (n=6251)

Figure 1: Performance improvement with $\tau$ for (left) Group Fused Lasso on a synthetic dataset and (right) Structural SVM on the OCR dataset.

We implemented Ap-Bcfw in a multicore shared-memory system and applied to the full OCR dataset ($n = 6877$) while varying $T$ and $\tau$. Figure 2 (left) shows that the algorithm gets better for the first few multiples of $T = 8$, but gets worse after that. For each $T$, there seems to be an optimal $\tau$ which gives the best performance. Figure 2 (middle) shows how performance improves with $T$. For this experiment, for each $T$, we set $\tau$ to a multiple of $T$ which gives the optimal speedup. Figure 2 (right) shows the speedup that is attained by the algorithm over BCFW when $T$ workers are available. It is important to note that the speedup is in terms of wall-clock time rather than the number of iterations.

![Primal suboptimality vs wall-clock time for $T = 8$ and various $\tau$s](left) Primal suboptimality vs wall-clock time for $T = 8$ and various $\tau$s; (middle) Primal suboptimality vs wall-clock time for varying $T$s with best $\tau$ chosen for each $T$ separately; (right) Speedup via parallelization

3.2 Performance gain with asynchronous nature

We compare Ap-Bcfw with a synchronous version of the algorithm (Sp-Bcfw) where the server assigns $\tau/T$ subproblems to each worker, then waits for and accumulates the solutions before proceeding to the next iteration. We simulate workers of varying speeds in our shared-memory setup by assigning a return probability $p_i \in (0, 1]$ to each worker $w_i$. After solving a subproblem, worker $w_i$ reports the solution to the server with probability $p_i$. 

10
We use $T = 14$ workers for the experiments in this section. We first simulate a scenario where there is just one slow machine (straggler) with a return probability $p \in (0, 1]$ while the other workers run at full speed ($p = 1$). Figure 3 (left) 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 to be expected as Ap-Bcfw relies on the average processing power available at the workers, while Sp-Bcfw is only as fast as the slowest worker. Next, we simulate a heterogeneous environment where the workers have varied speeds. While varying a parameter $\theta \in [0, 1]$, we set $p_i = \theta + i/T$ for $i = 1, \ldots, T$. Figure 3 (right) shows that Ap-Bcfw slows down by only a factor of 1.4 compared to the no-straggler case. Assuming that the server takes about half the (wall-clock) time and workers take the other half time on average per iteration, since the average time taken by the workers gets at most doubled when $\theta$ is close to 0 (for Ap-Bcfw), we would expect the average effective datapass time to go up by 1.5 times. A factor of 1.4 thus seems reasonable. The performance of Sp-Bcfw is almost identical to that in the previous experiment as its speed is determined by the slowest worker. Thus our simulations show that Ap-Bcfw is robust to stragglers and heterogeneity.

All shared-memory experiments were implemented in C++ and conducted on a 16-core machine with Intel(R) Xeon(R) CPU E5-2450 0 @ 2.10GHz processors and 128G RAM.

4 Conclusion

In this paper, we proposed an asynchronous parallel generalization of the block-coordinate Frank-Wolfe method [22] and provided intuitive conditions under which it has a provable speed-up over BCFW. The asynchronous updates allows our method to be robust to stragglers and node failure. We demonstrated the effectiveness of the algorithm in Group Fused Lasso and Structured SVM with both controlled simulation and real-data experiments on a multi-core workstation. For Structured SVM, it leads to a speed-up over the state-of-the-art for about 8 times using 16 parallel processors. As a projection-free Frank-Wolfe 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 case, allowing data distribution, bounding communication cost and ultimately releasing a carefully-implemented software package for practitioners’ easy deployment into Big Data applications.
Appendices

A Convergence analysis

We provide a self-contained convergence proof in this section. Most proof ideas follow closely from Lacoste-Julien et al. [22] and Jaggi [16]. Note that for the cleanness of the presentation, we focus on the primal and primal-dual convergence of the version of the algorithms with pre-defined step sizes and additive approximate subroutine, it is simple to extend the same analysis for line-search variant and multiplicative approximation.

A.1 Primal Convergence

Lemma 2. The iterative updates in Algorithm 1 (with arbitrary fixed stepsize $\lambda$ or by the coordinate-line search) obey
\[
\mathbb{E}[f(x^{(k+1)}) | x^{(k)}] \leq f(x^{(k)}) - \frac{\gamma T}{n} g(x^{(k)}) + \frac{\gamma^2}{2} (1 + \delta) C_f^r. \tag{A.1}
\]

Proof. Let $x := x^{(k)}$ for notational convenience. We prove the result for Algorithm 1 first. Apply the definition of $C(S)f$ and then apply the definition of the additive approximation in (2.3), to get
\[
f(x^{(k+1)}_{\text{line-search}}) \leq f(x_{\gamma}^{(k+1)}) = f(x + \gamma \sum_{i \in S} (s_{[i]} - x_{[i]}))
\leq f(x) + \gamma \sum_{i \in S} (s_{[i]} - x_{[i]}, \nabla_{[i]} f(x)) + \frac{\gamma^2}{2} C(S)_f
\leq f(x) + \gamma \sum_{i \in S} (-g^{(i)}(x) + \frac{\delta \gamma}{2} C_f^r) + \frac{\gamma^2}{2} C(S)_f
= f(x) - \gamma \sum_{i \in S} g^{(i)}(x) + \frac{\gamma^2}{2} (\delta C_f^r + C(S)_f).
\]
Taking expectations on both sides and substituting $\mathbb{E} \sum_{i \in S} g^{(i)}(x) = \frac{\tau}{n} g(x)$ and $\mathbb{E} S C(S)_f = C_f^r$, we get the equation as required by setting $x$ to $x^{(k)}$.

Now we are ready to state the proof for Theorem 1.

Proof of Theorem 1. We follow the proof in Theorem C.1 in Lacoste-Julien et al. [22] to prove the statement for Algorithm 1.

Denote the gap between current $f(x^k)$ and the optimal $f(x^*)$ to be $h(x^k)$. We have $g(x^k) \geq h(x^k)$ due to the fact that $g(x) = f(x) - f^*(\cdot)$. Apply this to $g(x)$ in the result in Lemma 2, and subtract $f^*$ from both sides, to get
\[
\mathbb{E}[h(x^{(k+1)}) | x^{(k)}] \leq (1 - \frac{\gamma T}{n}) h(x^{(k)}) + \frac{\gamma^2}{2} (1 + \delta) C_f^r.
\]
We may take expectation over $x^{(k)}$ on both side of the equation. For simplicity, take
\[
C = h_0 + n(1 + \delta) C_f^r,
\]

12
and denote $Eh(x^{(k)})$ as $h_k$. The above inequality then simplifies to

$$h_{k+1} \leq \left(1 - \frac{\gamma n \tau}{n}\right)h_k + \frac{\gamma^2 C}{2n}.$$ 

Now we will prove $h_k \leq \frac{2nC}{\tau^2 k + 2n}$ for $\gamma_k = \frac{2n\tau}{\tau^2 k + 2n}$ by induction. The base case $k = 0$ is trivially true since $C > h_0$. Assuming that the claim holds for $k$, we apply the induction hypothesis and the above inequality is reduced to

$$h_{k+1} \leq \left(1 - \frac{\gamma n \tau}{n}\right)h_k + \frac{\gamma^2 C}{2n} \leq \frac{2nC}{\tau^2 k + 2n} \left[1 - \frac{\gamma n \tau}{n} + \frac{\gamma^2 k + 2n \gamma^2}{2n}\right]$$

$$= \frac{2nC}{\tau^2 k + 2n} \left[\tau^2 k + 2n - \frac{2n\tau}{\tau^2 k + 2n} \cdot \frac{\tau}{n} + \frac{(2n\tau)^2}{4n^2(\tau^2 k + 2n)}\right]$$

$$= \frac{2nC}{\tau^2 k + 2n} \cdot \frac{\tau^2 k + 2n - \tau^2}{\tau^2 k + 2n} \leq \frac{2nC}{\tau^2 k + 2n} \cdot \frac{\tau^2 k + 2n - \tau^2 + \tau^2}{\tau^2 k + 2n + \tau^2}$$

This completes the induction and hence the proof for the primal convergence for Algorithm 1.

A.2 Convergence of the surrogate duality gap

Proof of Theorem 2. We mimic the proof in Lacoste-Julien et al. [22, Section C.3] for the analogous result closely, and we will use the same notation for $h_k$, $g_k$ and $C$ as in the proof for primal convergence. First from Lemma 2, we have

$$h_{k+1} \leq h_k - \frac{\gamma n \tau}{n} g_k + \frac{\gamma^2 C}{2n}.$$ 

Rearrange the terms, we get

$$g_k \leq \frac{n}{\gamma \tau} (h_k - h_{k+1}) + \frac{\gamma C}{2\tau}.$$ 

(A.2)

The idea is that if we take an arbitrary convex combination of $\{g_1, ..., g_K\}$, the result will be within the convex hull, namely between the minimum and the maximum, hence proven the existence claim in the theorem. By choosing weight $\rho_k := k/S_K$ where normalization constant $S_K = \frac{K(K+1)}{2}$ and taking the convex combination of both side of (A.2), we have

$$\mathbb{E}(\min_{k \in [K]} g_k) \leq \sum_{k=0}^{K} \rho_k g_k \leq \frac{n}{\tau} \sum_{k=1}^{K} \rho_k \left(\frac{h_k}{\gamma_k} - \frac{h_{k+1}}{\gamma_k}\right) + \sum_{k=0}^{K} \rho_k \frac{\gamma e C}{2\tau}$$

$$= \frac{n}{\tau} \left(\frac{h_0\rho_0}{\gamma_0} - h_{K+1}\frac{\rho_K}{\gamma_K}\right) + \frac{n}{\tau} \sum_{k=1}^{K-1} h_{k+1} \left(\frac{\rho_{k+1}}{\gamma_{k+1}} - \frac{\rho_k}{\gamma_k}\right) + \sum_{k=0}^{K} \rho_k \frac{\gamma e C}{2\tau}$$

$$\leq \frac{n}{\tau} \sum_{k=0}^{K-1} h_{k+1} \left(\frac{\rho_{k+1}}{\gamma_{k+1}} - \frac{\rho_k}{\gamma_k}\right) + \sum_{k=0}^{K} \rho_k \frac{\gamma e C}{2\tau}.$$ 

(A.3)
We can evenly partition sets $T$. Proof of Lemma 1.

$$\frac{\rho_{k+1} - \rho_k}{\gamma_{k+1}} = \frac{k + 1}{S_K} \frac{\tau^2(k + 1)2n}{2n} - \frac{k}{S_K} \frac{\tau^2k + 2n}{2n}$$

$$= \frac{1}{2nS_K} \left[ \tau^2(k + 1)^2 + 2n(k + 1) - \tau^2k^2 - 2nk \right]$$

$$= \frac{\tau^2(2k + 1) + 2n}{2nS_K \tau}.$$

Plug the above back into (A.3) and use the bound $h_k \leq 2nC/(\tau^2(k + 1) + 2n)$, we get

$$\mathbb{E}(\min_{k \in [K]} g_k) \leq \sum_{k=0}^{K} \rho_k g_k \leq \frac{nC}{\tau^2 S_K} \left[ \sum_{k=0}^{K-1} \frac{\tau^2(2k + 2) + 2n}{2n} \frac{2n}{\tau^2(k + 1) + 2n} + \sum_{k=0}^{K} \frac{k}{S_K} \frac{2n \tau}{\tau^2k + 2n} \frac{C}{2\tau} \right]$$

$$= \frac{nC}{\tau^2 S_K} \left[ \sum_{k=0}^{K-1} (1 + \frac{\tau^2}{2n}) + \sum_{k=1}^{K} \frac{k\tau^2}{(\tau^2k + 2n)} \right]$$

$$\leq \frac{nC}{\tau^2 S_K} \left[ 2K + 1 \right] = \frac{2nC}{\tau^2(K + 1)} \cdot 3.$$

This completes the proof for $K \geq 1$. \hfill \Box

**B Proofs of other technical results**

**Relationship of the curvatures.**

Proof of Lemma 1. $C_f^{(S)} \leq C_f$ follows from the fact that

$$\langle y_{(S)} - x_{(S)}, \nabla_{(S)} f(x) \rangle = \langle y_{[S]} - x_{[S]}, \nabla f(x) \rangle,$$

and $s_{[S]} \in M$. In other words, the arg sup of (2.1) is a feasible solution in the sup to compute the global $C_f$. Similar argument holds for the proof $C_f^{(i)} \leq C_f^{(S)}$ as $i \in S$.

In the second part,

$$C_f^T = \frac{1}{n} \sum_{T \subseteq [n], |T| = \tau} C_f^{(T)}.$$

We can evenly partition sets $T$ in the summation into $n$ parts $P_j$ for $j \in [n]$, such that sets in $P_j$ have the element $j$. Clearly each $P_j$ has a size of $\binom{n}{\tau}/n$. We can use $C_f^{(S)} \geq C_f^{(j)}$ from the first inequality of the lemma, to get the inequality below.

$$C_f^T = \frac{1}{n} \sum_{j \in [n]} \sum_{T \in P_j} C_f^{(T)} \geq \frac{1}{n} \sum_{j \in [n]} \sum_{T \in P_j} C_f^{(j)} = \frac{1}{n} \sum_{j \in [n]} \left( \binom{n}{\tau} \frac{1}{n} C_f^{(j)} \right) = \frac{1}{n} C_f^{(j)}$$

The relaxation of $C_f^T$ to $C_f$ is trivial since $C_f^{(T)} \leq C_f$ holds for any $T \subseteq [n]$ from the first part of the lemma. \hfill \Box
Computational guarantee for distributed workers

Proof of Proposition 1. The first claim is the well-known coupon collector problem.

The second claim requires an upper bound of the expectation. In expectation, we need \( \frac{n}{n-k} \) balls to increase the unique count from \( k \) to \( k+1 \). So in expectation we need

\[
1 + \frac{n}{n-1} + \frac{n}{n-2} + \ldots + \frac{n}{n-\tau + 1} = \tau + \sum_{i=1}^{\tau-1} \frac{i}{n-i}.
\]

To see the second claim, first defined \( f_t \) to be the number of non-empty bins after \( t \) random ball throws, which can be consider as a function of the \( t \) iid ball throws \( X_1, X_2, \ldots, X_t \). It is clear that if we change only one of the ball throws, which can be consider as a function of the \( t \) ball throws which can be changed by at most 1. Also, note that the probability that any one bin being filled is \( 1 - \left( 1 - \frac{1}{n} \right)^{t} \), so \( \mathbb{E} f_t = n \left[ 1 - \left( 1 - \frac{1}{n} \right)^{t} \right] \).

By the McDiarmid’s inequality, \( \mathbb{P}[f_t < \mathbb{E} f_t - \epsilon] \leq \exp \left( -\frac{2\epsilon^2}{\tau} \right) \). Take \( t = 2\tau \), and \( \epsilon = \mathbb{E} f_{2\tau} - \tau \), then

\[
\mathbb{P}[f_{2\tau} < \tau] \leq \exp \left( -\frac{1}{\tau} \left( n \left[ 1 - (1 - \frac{1}{n})^{2\tau} \right] - \tau \right)^2 \right) \leq \exp \left( -\frac{1}{\tau} \left( n \left[ 1 - e^{-2\tau} \right] - \tau \right)^2 \right) = \exp \left( -n \cdot \frac{1}{\tau} \left( 1 - e^{-2\tau} - \frac{\tau}{n} \right)^2 \right) \leq \exp \left[ -Cn \right],
\]

where \( C \) is some constant which is the smaller of the two evaluations of the function \( \frac{2}{\tau} \left( 1 - e^{-2\tau} - \frac{\tau}{n} \right)^2 \) at \( \tau = 0.02n \) and \( \tau = 0.6n \) (where the function is concave between the two). As a matter of fact, \( C \) can be taken as \( \frac{1}{100} \).

Let \( g_{\tau} \) be the number of balls that one throws that fills \( \tau \) bins, the result is proven by noting that

\[
\mathbb{P}(g_{\tau} \leq 2\tau) = \mathbb{P}(f_{2\tau} \geq \tau) \geq 1 - \exp \left[ -Cn \right].
\]

Bounding \( \kappa_f^S \) using expected boundedness and expected incoherence

Proof of Theorem 3. By Definition of \( H \), for any \( x, z \in \mathcal{M}, \gamma \in [0, 1] \)

\[
f(x + \gamma(z - x)) \leq f(x) + \gamma(z - x)^T \nabla f(x) + \frac{\gamma^2}{2} (z - x)^TH(z - x).
\]

Rearranging the terms we get

\[
\frac{2}{\gamma^2} \left[ f(x + \gamma(z - x)) - f(x) - \gamma(z - x)^T \nabla f(x) \right] \leq (z - x)^TH(z - x)
\]

The definition of set curvature (2.1) is written in an equivalent notation with \( x = x|S^c| + s|S| \) and \( y = x + \gamma(z - x) = x + \gamma(s|S| - x|S|) \). So we know \( z - x \) is constrained to be within the coordinate blocks \( S \).
Plugging this into the definition of (2.1) we get an analog of Equation (2.12) in Jaggi [15] for $C_j^{(S)}$.

$$
C_j^{(S)} = \sup_{x \in M, \gamma \in [0,1], \gamma' \in \mathcal{M}^{\tau}} \frac{2}{\gamma^2} \left[ f(x + \gamma (z - x)) - f(x) - \gamma (z - x)^T \nabla f(x) \right]
$$

$$
\leq \sup_{x, z \in M, \gamma \in [0,1]} (z - x)^T H(z - x) = \sup_{x, z \in M, \gamma \in [0,1]} s^T_{(S)} H s_{(S)}
$$

$$
\leq \sup_{w \in M^{(S)}} (2w^T) H_S (2w) = 4 \left\{ \sup_{w_i \in M^{(1)} : i \in S} \sum_{i \in S} w_i^T H_{ii} w_i + \sum_{i,j \in S, i \neq j} w_i^T H_{ii} w_j \right\}
$$

$$
\leq 4 \left\{ \sum_{i \in S} \sup_{z \in M} w_i^T H_{ii} (z) w_i + \sum_{i,j \in S, i \neq j} \sup_{z \in M} w_i^T H_{ii} (z) w_j \right\}
$$

$$
\leq 4 \left( \sum_{i \in S} B_i + \sum_{i,j \in S, i \neq j} \mu_{ij} \right).
$$

Take expectation for all possible $S$ of size $\tau$ and we obtain the lemma statement. \hfill \Box

**Proof of specific examples**

**Proof of Example 1.** First of all, $H = \lambda A^T A$. Since all columns of $A$ have the same magnitude $\sqrt{2/n}$. By the Holder’s inequality and the 1-norm constraint in every block, we know $B_i = \frac{2}{\pi^2}$ for any $i$ therefore $B = \frac{2}{\pi^2}$. Secondly, by well-known upper bound for the area of the spherical cap, which says for any fixed vector $z$ and random vector $a$ on a unit sphere in $\mathbb{R}^d$,

$$
P(\|\langle z, a \rangle \| > \epsilon \|z\|) \leq 2e^{-\frac{\epsilon^2}{2}}
$$

we get

$$
P(\mu_{ij} > 2 \sqrt{\frac{20 \log d}{d}}) \leq \frac{2}{d^10}.
$$

Take union bound over all pairs of labels we get the probability as claimed. \hfill \Box

**Proof of Example 2.** The matrix $D^T D$ is tridiagonal with 2 on the diagonal and 1 on the off-diagonal. If we vectorize $U$ by concatenating $u = [u_1; \ldots; u_{n-1}]$, the Hessian matrix for $u$ will be $H = \Pi U \otimes (D^T D) \Pi^T$ where $\Pi$ is some permutation matrix. Without calculating it explicitly, we can express

$$
u_t^T H_S u_S = u_t^T (D^T \otimes 1_d) (D^T \otimes 1_d)^T u_S
$$

$$
= \sum_{i \in S} u_i^T \begin{bmatrix} D_{t,i}^T \mspace{10mu} D_{t,i} \mspace{10mu} \vdots \mspace{10mu} D_{t,i} \end{bmatrix} \begin{bmatrix} D_{t,i} \mspace{10mu} D_{t,i} \mspace{10mu} \vdots \mspace{10mu} D_{t,i} \end{bmatrix} u_i + \sum_{i,j \in S, i \neq j} u_i^T \begin{bmatrix} D_{t,i}^T \mspace{10mu} D_{t,i} \mspace{10mu} \vdots \mspace{10mu} D_{t,i} \end{bmatrix} \begin{bmatrix} D_{t,j} \mspace{10mu} D_{t,j} \mspace{10mu} \vdots \mspace{10mu} D_{t,j} \end{bmatrix} u_j.
$$

We note that for any $|i - j| \geq 2$, the second term is 0. Apply the constraint that $\|u_i\|_2 \leq \lambda$ and the fact that the $\ell_2$ operator norm of $[D_{t,j} \mspace{10mu} D_{t,j} \mspace{10mu} \vdots \mspace{10mu} D_{t,j}]$ is $\sqrt{2d}$, we get $B_i = 2\lambda^2 d$. Similarly, $2(n-2)$ nonzero obeys $\mu_{ij} = \lambda^2 d$. This allows us to obtain an upper bound

$$
C_j^2 \leq 2\tau \lambda^2 d + \frac{2(n-2)\tau(\tau-1)}{(n-2)(n-1)} \lambda^2 d \leq 4\tau \lambda^2 d.
$$

which scales with $\tau$. \hfill \Box
C Application to Structural SVM

We briefly review structural SVMs and show how to solve the associated convex optimization problem using our AP-BcFW method.

In structured prediction setting, the task is to predict a structured output \( y \in \mathcal{Y} \), given \( x \in \mathcal{X} \). For example, \( x \) could be the pixels in the picture of a word, \( y \) could be the sequence of characters in the word. A feature map \( \phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d \) encodes compatibility between inputs and outputs. A linear classifier parameter \( w \) is learned from data so that argmax \( y \in \mathcal{Y} \{ \langle w, \phi(x, y) \rangle \} \) gives the output for an input \( x \). Suppose we have the training data \( \{x_i, y_i\}_{i=1}^n \) to learn \( w \). Define \( \psi_i(y) := \phi(x_i, y_i) - \phi(x_i, y) \) and let \( L_i(y) := L(y_i, y) \) denote the loss incurred by predicting \( y \) instead of the correct output \( y_i \). The classifier parameter \( w \) is learned by solving the optimization problem

\[
\begin{align*}
\min_{w \in \mathbb{R}^d} & \quad \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^n \xi_i \\
\text{s.t.} & \quad \langle w, \psi_i(y) \rangle \geq L(y_i, y) - \xi_i \quad \forall i, y \in \mathcal{Y}(x_i).
\end{align*}
\]  

(C.1)

We solve the dual of this problem using our method. We introduce some more notation to formulate the dual. Denote \( \mathcal{Y}_i := \mathcal{Y}(x_i) \), the set of possible labels for \( x_i \). Note that \( |\mathcal{Y}_i| \) is exponential in the length of label \( y_i \). Let \( m = \sum_{i=1}^n |\mathcal{Y}_i| \). Let \( A \in \mathbb{R}^{d \times m} \) denote a matrix whose \( m \) columns are given by \( \{ \frac{1}{m} \psi_i(y) | i \in [n], y \in \mathcal{Y}_i \} \). Let \( b \in \mathbb{R}^m \) be a vector given by the entries \( \{ \frac{1}{m} L_i(y) | i \in [n], y \in \mathcal{Y}_i \} \). The dual of (C.1) is given by

\[
\begin{align*}
\min_{\alpha \in \mathbb{R}^m} & \quad f(\alpha) := \frac{\lambda}{2} \| A \alpha \|^2 - b^T \alpha \\
\text{s.t.} & \quad \sum_{y \in \mathcal{Y}_i} \alpha_i(y) = 1 \quad \forall i \in [n], \alpha \geq 0
\end{align*}
\]  

(C.2)

The primal solution \( w \) can be retrieved from the dual solution \( \alpha \) from the relation \( w = A \alpha \) obtained from KKT conditions. Also note that the domain \( \mathcal{M} \) of (C.2) is exactly the product of simplices \( \mathcal{M} = \Delta_{|\mathcal{Y}_1|} \times \cdots \times \Delta_{|\mathcal{Y}_n|} \).

The subproblem in equation (1.3) takes a well-known form in the Frank-Wolfe setup for solving (C.2). The gradient is given by

\[
\nabla f(\alpha) = \lambda A^T A \alpha - b = \lambda A^T w - b
\]

whose \((i, y)\)-th component is given by \( \frac{1}{m} \langle \psi_i(y), \psi_i(y) \rangle \) so that the \((i, y)\)-th component of the gradient is \( -\frac{1}{m} H_i(y; w) \). In the subproblem (1.3), the domain \( \mathcal{M}^{(i)} \) is the simplex \( \Delta_{|\mathcal{Y}_i|} \) and the block gradient \( \nabla_{(i)} f(\alpha) \) is linear. So, the objective is minimized at a corner of the simplex \( \mathcal{M}^{(i)} \) and the optimum value is simply given by \( \min_{\alpha} \nabla_{(i)} f(\alpha) \) which can be rewritten as \( \max_{y} H_i(y; w) \). Further, the corner can be explicitly written as the indicator vector \( e_{y^*_i} \in \mathcal{M}^{(i)} \) where \( y^*_i = \arg\max_{y} H_i(y; w) \). It turns out that this maximization problem can be solved efficiently for several problems. For example, when the output is a sequence of labels, a dynamic programming algorithm like Viterbi can be used.

As mentioned before, \( m \) is too large to update the dual variable \( \alpha \) directly. So, we make an update to the primal variable \( w = A \alpha \) instead. The Block-Coordinate Frank-Wolfe update for the \( i \)-th block maybe written as \( \alpha_{(i)}^{k+1} = \alpha_{(i)}^k + \gamma (s_i - \alpha_{(i)}^k) \) where \( \gamma \) is the step-size. Recalling that the optimal \( s_i \) is \( e_{y_i^*} \), by multiplying the previous equation by \( A_i \), we arrive at \( w_i^{(k+1)} = w_i^k + \gamma (A_i y_i^* - w_i^k) \) where \( w_i^{(k)} := A_i \alpha_{(i)}^k \). From this definition of \( w_i^{(k)} \), the primal update is
obtained by noting that $w^{(k)} = \sum_i w_i^{(k)}$. Explicitly, the primal update is given by

$$w^{(k+1)} = w^k + \gamma (A_i, y_i^* - w_i^{(k)})$$

Note that $A_i, y_i^* = \frac{1}{\lambda n} \psi_i(y_i^*)$. This Block-Coordinate version can be easily extended to AP-BCFW. In our shared memory implementation, for OCR dataset, we do the line search computation and $w_i^{(k)}$ update step on the workers instead of the server because these computations turn out to be expensive enough to make the server the bottleneck even for modest number of workers.

\section{Other technical results and discussions}

\textbf{Example 3} (Structural SVM worst-case bound). For structural SVM with arbitrary data (including even pathological/trivial data), using notation from Lemmas A.1 and A.2 of Lacoste-Julien et al. [22], define $R := \max_{i \in [n], y \in Y_i} \|\psi_i(y)\|_2$. Then we can provide an upper bound

$$B, \mu \leq \frac{R^2}{\lambda n^2} \Rightarrow C_\tau \leq \frac{4\tau^2 R^2}{\lambda n^2}.$$  \hfill (D.1)

In this case, for any $\tau = 1, \ldots, n$, the rate of convergence will be the same $O(\frac{R^2}{\lambda n^2})$.

\textbf{An illustration for the group fused lasso} Figure 4 shows a typically application for group fused lasso (filtering piecewise constant multivariate signals whose change points are grouped together).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{signal_data.png}
\caption{Illustration of the signal data used in the Fused Lasso experiments. We show the original signal (left), the noisy signal given to the algorithm (middle), and the signal recovered after performing the fused lasso optimization (right).}
\end{figure}

\begin{thebibliography}{99}
\bibitem{1} C. M. Alaiz, Á. Barbero, and J. R. Dorronsoro. Group fused lasso. In \textit{Artificial Neural Networks and Machine Learning–ICANN 2013}, pages 66–73. Springer, 2013.
\bibitem{2} F. Bach. Conditional gradients everywhere. 2013.
\bibitem{3} A. Beck and L. Tetruashvili. On the convergence of block coordinate descent type methods. \textit{SIAM Journal on Optimization}, 23(4):2037–2060, 2013.
\end{thebibliography}
[4] A. Bellet, Y. Liang, A. B. Garakani, M.-F. Balcan, and F. Sha. Distributed Frank-Wolfe algorithm: A unified framework for communication-efficient sparse learning. arXiv preprint arXiv:1404.2644, 2014.

[5] K. Bleakley and J.-P. Vert. The group fused lasso for multiple change-point detection. arXiv, 2011.

[6] K. Bredies, D. A. Lorenz, and P. Maass. A generalized conditional gradient method and its connection to an iterative shrinkage method. Computational Optimization and Applications, 42(2):173–193, 2009.

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

[8] M. Collins, A. Globerson, T. Koo, X. Carreras, and P. L. Bartlett. Exponentiated gradient algorithms for conditional random fields and max-margin markov networks. The Journal of Machine Learning Research, 9:1775–1822, 2008.

[9] W. Dai, J. Wei, X. Zheng, J. K. Kim, S. Lee, J. Yin, Q. Ho, and E. P. Xing. Petuum: A framework for iterative-convergent distributed ml. arXiv preprint arXiv:1312.7651, 2013.

[10] O. Fercoq and P. Richtárik. Accelerated, parallel and proximal coordinate descent. arXiv preprint arXiv:1312.5799, 2013.

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

[12] D. Garber and E. Hazan. A linearly convergent conditional gradient algorithm with applications to online and stochastic optimization. arXiv preprint arXiv:1301.4666, 2013.

[13] Z. Harchaoui, A. Juditsky, and A. Nemirovski. Conditional gradient algorithms for norm-regularized smooth convex optimization. arXiv preprint arXiv:1302.2325, 2013.

[14] E. Hazan and S. Kale. Projection-free online learning. In ICML, 2012.

[15] M. Jaggi. Sparse convex optimization methods for machine learning. PhD thesis, Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 20013, 2011.

[16] M. Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In Proceedings of the 30th International Conference on Machine Learning (ICML-13), pages 427–435, 2013.

[17] S. Jegelka, F. Bach, and S. Sra. Reflection methods for user-friendly submodular optimization. In Advances in Neural Information Processing Systems, pages 1313–1321, 2013.

[18] J. A. Kelner, L. Orecchia, A. Sidford, and Z. A. Zhu. A simple, combinatorial algorithm for solving sdd systems in nearly-linear time. In Proceedings of the 45th annual ACM symposium on Symposium on theory of computing, pages 911–920. ACM, 2013.

[19] D. Koller, B. Taskar, and C. Guestrin. Max-margin Markov networks. 2003.

[20] I. Koutis, G. L. Miller, and R. Peng. A nearly-m log n time solver for sdd linear systems. In Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on, pages 590–598. IEEE, 2011.

[21] S. Lacoste-Julien and M. Jaggi. An affine invariant linear convergence analysis for Frank-Wolfe algorithms. arXiv preprint arXiv:1312.7864, 2013.

[22] S. Lacoste-Julien, M. Jaggi, M. Schmidt, and P. Pletscher. Block-coordinate Frank-Wolfe optimization for structural SVMs. arXiv preprint arXiv:1207.4747, 2012.

[23] L. J. LeBlanc, E. K. Morlok, and W. P. Pierskalla. An efficient approach to solving the road network equilibrium traffic assignment problem. Transportation Research, 9(5):309–318, 1975.

[24] M. Li, L. Zhou, Z. Yang, A. Li, F. Xia, D. G. Andersen, and A. Smola. Parameter server for distributed machine learning. In NIPS Workshop: Big Learning, 2013.

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

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

[27] F. Niu, B. Recht, C. Ré, and S. J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *arXiv preprint arXiv:1106.5730*, 2011.

[28] H. Ouyang and A. G. Gray. Fast stochastic Frank-Wolfe algorithms for nonlinear SVMs. In *SDM*, 2010.

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

[30] J. N. Tsitsiklis, D. P. Bertsekas, M. Athans, et al. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE transactions on automatic control*, 31(9):803–812, 1986.

[31] P. Wolfe. Convergence theory in nonlinear programming. *Integer and nonlinear programming*, 1970.

[32] C.-N. J. Yu and T. Joachims. Learning structural SVMs with latent variables. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1169–1176. ACM, 2009.

[33] X. Zhang, Y. Yu, and D. Schuurmans. Accelerated training for matrix-norm regularization: A boosting approach. In *NIPS*, pages 2915–2923, 2012.

[34] X. Zhang, Y.-L. Yu, and D. Schuurmans. Polar operators for structured sparse estimation. In *Advances in Neural Information Processing Systems*, pages 82–90, 2013.