Asynchronous Parallel Block-Coordinate Frank-Wolfe

Yu-Xiang Wang, Veeranjaneyulu Sadhanala, Wei Dai,
Willie Neiswanger, Suvrit Sra, and Eric Xing
Machine Learning Department, Carnegie Mellon University, Pittsburgh, PA 15213
{yuxiangw,vsadhana,wdai,willie,epxing}@cs.cmu.edu

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 [18] 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 [27] has recently witnessed a surge of interest in machine learning [6, 13, 14, 2]. Its wide applicability and practical appeal have spurred several extensions, including to regularized optimization [29, 5, 11], to linearly convergent algorithms [17, 10], to stochastic/online versions [23, 12], and to a powerful randomized block-coordinate version [18].

Motivated by [18], we consider FW methods for the following convex optimization problem

$$\min_x \quad f(x) \qquad \text{s.t.} \quad x = [x_{(1)}, ..., x_{(n)}] \in \mathcal{M}_1 \times \cdots \times \mathcal{M}_n := \mathcal{M} \tag{1}$$

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 [18], routing problems [19], dual of the group fused-lasso [1, 4], nearest point projection onto convex sets [15], among others.

A typical approach to solving (1) is to use a block-coordinate descent method, which involves solving a projection subproblem in every iteration [21, 24, 3]. This can sometimes be very expensive, e.g., submodular minimization [9], or even computationally intractable [7].

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 of the form

$$\min_x \quad \langle x, \nabla f(\cdot)\rangle \qquad \text{s.t.} \quad x \in \mathcal{M}. \tag{2}$$

This simpler structure can bestow great computational advantages; moreover, it enables another practical advantage of FW methods: *sparsity* or *low-rank* of intermediate iterates [6, 14].

For $\mathcal{M} = \prod_i \mathcal{M}_i$, problem (2) decomposes into n independent subproblems

$$\min_{s_{(i)} \in \mathcal{M}_i} \quad \langle s_{(i)}, \nabla f(x_{(i)})\rangle, \quad 1 \leq i \leq n, \tag{3}$$

where $x_{(i)}$ denotes the restriction (projection) of x to $\mathcal{M}_i$. Clearly, these n subproblems can be solved in parallel. However, updating all the coordinates at each iteration (whether in serial or parallel) may correspond to going through the entire input data, which handicaps the applicability of FW to true big-data problems. This drawback was partially ameliorated by the recent randomized Block-Coordinate Frank-Wolfe (BCFW) method of [18]. 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 block-coordinate Frank-Wolfe algorithm that allows asynchronous computation within each minibatch. The algorithm is robust to stragglers and faulty workers.
- 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.

Related works. The general prospect of the research is closely related to the recent effort in parallelizing sequential algorithms while adapting to the strengths and limitations of the modern computer systems, including that for stochastic gradient methods [25, 22], coordinate descent [24, 20] and so on. Building upon these predecessors, this work is novel in the problem scope (Frank-Wolfe), the asynchronous system design and parts of our theoretical analysis. We invite the readers to the detailed comparisons in the full paper.

Notation. The vector $x \in \mathbb{R}^m$ denotes the parameter vector, possibly split into n coordinate blocks, each $x_{(i)} \in \mathbb{R}^{m_i}$. $x_{[i]} \in \mathbb{R}^m$ denotes the projection of x onto the ith block. We denote the size of a minibatch by τ, and the number of parallel workers (threads) by T.

2 Algorithm

In this section, we develop and analyze an asynchronous parallel block-coordinate Frank-Wolfe algorithm, hereafter AP-BCFW, to solve (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 [22, 24, 20], and most modern multicore machines permit such an arrangement.

At a high-level, AP-BCFW (Algorithm 1) 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, AP-BCFW divides the workload of τ 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 τ 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.

2.1 Convergence results

Our convergence results rely on the notion of **set curvature**

$$C_f^{(S)} := \sup_{\substack{x\in\mathcal{M}, s_{(S)}\in\mathcal{M}^{(S)},\\ \gamma\in[0,1],\, y=x+\gamma(s_{[S]}-x_{[S]})}} \frac{2}{\gamma^2}\big(f(y) - f(x) - \langle y_{(S)} - x_{(S)}, \nabla_{(S)} f(x)\rangle\big). \tag{4}$$

This is a natural interpolation of the standard **curvature** in the affine-invariant proof of batch FW [14] and the **coordinate curvature** used in BCFW [18]. This is also closedly related to the coordinate Lipschitz constant in the analysis of coordinate descent methods.

Lemma 1 (Curvature relations). *Suppose $S \subseteq [n]$ with cardinality $|S| = \tau$ and $i \in S$. Also let $C_f^\tau := \mathbb{E}_S C_f^{(S)}$. Then, (i) $\quad C_f^{(i)} \leq C_f^{(S)} \leq C_f$; $\quad$ (ii) $\quad \frac{1}{n} C_f^{\otimes} = C_f^1 \leq C_f^\tau \leq C_f^n = C_f$.*

This curvature assumption allows us to inherit the typical traits of FW methods including:

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

```
Input: An initial feasible $x^{(0)}$, mini-batch size $\tau$, number of workers $T$.
for $k$ = 1,2,... ($k$ is the epoch.) do
    1. Randomly pick $S \subset [n]$ such that $|S| = \tau$ and set the index set $\tilde{S} = S$.
    2. [IN PARALLEL] While $\tilde{S} \neq \emptyset$, idle workers choose a $j \in \tilde{S}$ randomly, solves (3)
    "approximately" for $s_{(j)}$, optionally determine the line-search stepsizes $\gamma_j$ and set $\tilde{S} = \tilde{S} - \{j\}$.
    3. Update $x^{(k+1)} = x^{(k)} + \gamma \sum_{i \in S}(s_{[i]} - x^{(k)}_{[i]})$ with $\gamma = \frac{2n\tau}{\tau^2 k + 2n}$
    4. (Optionally) Compute $x^{(k+1)}_{\text{line-search}} = x^{(k)} + \sum_{i \in S} \gamma_i (s_{[i]} - x^{(k)}_{[i]})$.
    Update $x^{(k+1)}$ to $x^{(k+1)}_{\text{line-search}}$ if line-search improves the objective value.
end for
Output: limit point $\bar{x}$.
```

Affine invariance. No pre-conditioning is needed to get the best convergence.

Approximate oracle. The linear oracle allows an additive or multiplicative error.

Primal-Dual convergence. $O(nC/\tau^2 K)$ convergence guarantee of the duality gap, with consistent estimate of the duality gap "for free" in every iteration. (See the arxiv version for details.)

Due to space constraint, we describe only the theorem for primal convergence.

Theorem 1 (Primal Convergence)**.** *For each $k \geq 0$, the iterations in Algorithm 1 and its line search variant obey*

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

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

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 [18, Theorem 2] and if we take $\tau = n$, we can drop $f(x^{(0)}) - f(x^*)$ from C and it reduces to the classic batch guarantee as in [14].

Speedup. The careful reader may have noticed that the undesirable n^2 dependence, which could be prohibitive for large n. The saving grace in BCFW is that when $\tau = 1$, C_f^τ is as small as $O(n^{-2})$ (see [18, Lemmas A1 and A2]). What really matters is how much speedup one can achieve over BCFW, and this speedup critically relies on how C_f^τ depends on τ. Basically, if $C_f^\tau = o(\tau^2)$, we gain in convergence speed. In the ideal case $C_f^\tau = O(\tau)$, the speed-up will be linear in τ.

2.2 The effect of parallelism

To further understand when mini-batching is meaningful and quantify its speedup, we analyze and present a set of insightful conditions that govern the relationship between C_f^τ and τ.

Theorem 2. *If problem* (1) *obeys B-expected boundedness and μ-expected incoherence. Then,*

$$C_f^\tau \leq 4(\tau B + \tau(\tau - 1)\mu) \qquad \textit{for any} \quad \tau = 1, ..., n. \tag{5}$$

We leave the exact definitions of B and μ in the full paper [26]. Intuitively, they measure respectively the expected coordinate-wise curvature and the expected pairwise interaction. The theorem implies that when μ is small, i.e., when each coordinate is close to independent on average, then we can expect a large speed-up with AP-BCFW. This is analogous to the analysis in parallel coordinate descent [24, 20]. In fact, one can view our condition as an affine-invariant version of the Expected Separable Overapproximation (ESO) condition [24].

SDD matrices and Graph Fused Lasso. If the $n \times n$ matrix formed by putting B_i on the diagonal and μ_{ij} on the off-diagonal is *symmetric diagonally dominant* (SDD), then C_f^τ is proportional to τ. For instance, the dual of the Group Fused Lasso problem studied in Wytock et al. [28]. Also, graph Laplacians are symmetric diagonal dominant.

Structural SVM. In the worst case, a simple generalization of Lacoste-Julien et al. [18, Lemmas A.1, A.2] shows that $\tau > 1$ offers no gain at all. If we consider a specific problem and the average case, using larger τ does make the algorithm converge faster (and this is the case according to our experiments).

Parallel block coordinate descent. We compare the rate of convergence in Theorem 1 with parallel BCD [24, 20] under the assumption of $\mu = O(B/\tau)$ — an equally favorable case to all methods. To facilitate comparison, we also convert the constants in all methods to block coordinate gradient Lipschitz constant L_i and $R := \max_x \|x - x^*\|$.

	AP-BCFW (Ours)	P-BCD [24, Theorem 19]	AP-BCD [20, Theorem 3]
Rate	$O_p\left(\frac{n\mathbb{E}_i(L_i)R^2}{\tau k}\right)$	$O_p\left(\frac{n\mathbb{E}_i(L_i)R^2}{\tau k}\right)$	$O_p\left(\frac{n \max_i L_i R^2}{\tau k}\right)$

The comparison illustrates that these methods have the same $O(1/k)$ rate and almost the same dependence on n and τ despite the fact that we use a much simpler linear oracle. We note that with Nesterov acceleration, we can actually get $O(1/k^2)$ rate for coordinate decent [8], while the matching minimax lower bound of FW is disappointingly $O(1/k)$. Yet, when projection is much harder to compute than (3) (e.g., nuclear norm balls), or when the structures in intermediate steps matter, our method becomes a valuable plug-in solution for solving such problems in large-scale.

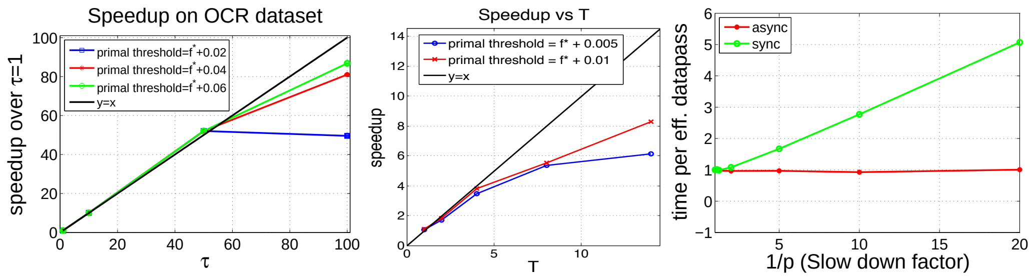
3 Experiments

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

Performance gain due to minibatch size τ: We simulated AP-BCFW on a part of OCR dataset[16] ($n = 6251, d = 4082$), which is a sequence labeling task where the subproblem can be solved using the Viterbi algorithm. In each iteration, we solve τ subproblems, thus simulating the parallel setting with τ workers and each worker solving one subproblem. Figure 1a shows that the speedup(compared to BCFW) is almost ideal until a reasonably large value for τ. The speedup gets worse if τ is too large as there are too many stale updates. We implemented AP-BCFW in a multi-core shared-memory system and applied to the full OCR dataset($n = 6877$). Figure 1b shows the speedup in terms of *wall-clock time* that is attained by the algorithm over BCFW when T workers are available.

Performance gain with asynchronous nature: We compare AP-BCFW with a synchronous version of the algorithm (SP-BCFW) where in each iteration, the server assigns τ/T subproblems to each worker, then waits for and accumulates the solutions. We simulate workers of varying speeds in our shared-memory setup by assigning each worker w_i a *return probability* $p_i \in (0, 1]$ with which they return a solution.

We 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 1c 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. AP-BCFW performs in a similarly superior manner when the workers have varied speeds, see arXiv version.



(a) Structural SVM on OCR (b) Wall clock-time speed up (c) AP-BCFW vs SP-BCFW

Figure 1: Speedup in a simulation, a real cluster, and performance with stragglers.

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.

References

[1] C. M. Alaíz, Á. Barbero, and J. R. Dorronsoro. Group fused lasso. In *Artificial Neural Networks and Machine Learning–ICANN 2013*, pages 66–73. Springer, 2013.

[2] F. Bach. Conditional gradients everywhere. 2013.

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

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

[5] 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.

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

[7] 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.

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

[9] 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.

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

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

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

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

[14] 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.

[15] 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.

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

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

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

[19] 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.

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

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

[22] 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.

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

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

[25] 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.

[26] Y.-X. Wang, V. Sadhanala, W. Dai, W. Neiswanger, S. Sra, and E. P. Xing. Asynchronous parallel block-coordinate frank-wolfe. *arXiv preprint arXiv:1409.6086*, 2014.

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

[28] M. Wytock, S. Sra, and J. Z. Kolter. Fast newton methods for the group fused lasso. In *Uncertainties in Artificial Intelligence*, 2014.

[29] 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.