A Block-Coordinate Frank-Wolfe Splitting Algorithm: Breaking the Complexity Barrier

JMU CSM Symposium

Nicholas Harsell
harselnw[at]dukes.jmu.edu
James Madison University (JMU)
Optimization and Algorithmic Theory Lab
NSF Grant #DMS-2532423

July 2026

Split Block-Coordinate Frank-Wolfe Algorithm

What is Constrained Optimization?

In data science, physics, and engineering, we frequently solve problems of the form:

$$\min_{x \in \mathcal{C}} f(x)$$

- **The Objective Function ($f(x)$):** A mathematical formula measuring "cost," "error," or "energy." Our goal is to find a solution x that makes this value as small as possible.
- **The Constraint Set ($\mathcal{C}$):** The set of valid solutions. The constraint set represents physical limitations or requirements (e.g., probabilities must sum to 1, energy must be positive).

The Goal of Constrained Optimization

How do we systematically step downhill toward the minimum of $f(x)$ without accidentally stepping outside the boundaries of $\mathcal{C}$?

A Limitation of Projection Methods

Standard iterative optimization algorithms (like Projected Gradient Descent) handle constraints using a two-step process at each iteration:

1. Take a standard downhill step along the gradient: $x_t - \gamma_t \nabla f(x_t)$.
2. If that step lands outside the valid set $\mathcal{C}$, **project** it back to the closest boundary point.

A Computational Bottleneck

Finding the "closest boundary point" requires solving a quadratic distance equation:

$$\text{Proj}_{\mathcal{C}}(x) := \arg\min_{v \in \mathcal{C}} \|v - x\|^2$$

For complex scientific data (like large matrices or networks), this projection step can be prohibitively expensive (e.g., requiring full Singular Value Decompositions).

A Cheaper Alternative (The LMO)

Instead of taking an unconstrained step and requiring a projection to remain in the constraint set, we replace Euclidean projection with a **Linear Minimization Oracle (**LMO**)**.

The Intuition: Asking a Simpler Question

Instead of asking: *"What is the exact closest valid point?"* We ask: *"If the landscape were completely flat, which corner of our boundary should we run toward?"*

$$\mathrm{LMO}_{\mathcal{C}}(\nabla f(x_t)) \in \arg\min_{v \in \mathcal{C}} \langle \nabla f(x_t) \mid v \rangle$$

LMO's are the computational basis of Frank-Wolfe algorithms.

Frank-Wolfe (Conditional Gradient) Methods

The **Frank-Wolfe (FW)** algorithm uses the LMO to optimize without ever needing to project. FW methods typically require f to be a *smooth* function; we call its smoothness constant L_f.

How It Works (Three Steps)

At each iteration t:

1. **Look Downhill:** Compute the local slope (gradient $\nabla f(x_t)$).
2. **Find a Target Corner:** Compute the LMO to find which valid boundary point v_t is furthest downhill along that linear slope.
3. **Take a Safe Step:** Move a percentage $\gamma_t \in [0, 1]$ of the way toward that target:

$$x_{t+1} = x_t + \gamma_t(v_t - x_t)$$

Frank-Wolfe: Pacing and Measuring Success

To guarantee we reach the true minimum, we must control our step sizes (γ_t) and know when to stop running the algorithm. In this work, we mostly consider **short-step**:

$$\gamma_t = \min\left\{1, \frac{\langle \nabla f(x_t) \mid x_t - v_t \rangle}{L_f \|x_t - v_t\|^2}\right\}.$$

How do we know we are done?

We track the **Frank-Wolfe Gap**:

$$G(x_t) := \langle \nabla f(x_t) \mid x_t - v_t \rangle$$

Convergence: for an objective function which is allowed to be nonconvex, a typical convergence theorem will prove that a subsequence of these Frank-Wolfe Gaps converges to zero.

Split Block-Coordinate Frank-Wolfe Algorithm

The Problem: Overlapping Constraints

Let $\mathcal{H}$ denote a Hilbert space (which gives us nice norm and inner product properties). In real-world applications, a valid solution often has to satisfy multiple complex requirements simultaneously:

$$\min_{x \in \mathcal{H}} f(x) \qquad \text{subject to} \qquad x \in \bigcap_{i=1}^{m} \mathcal{C}_i \tag{$*$}$$

Example: A signal processing model where the output must be both sparse ($\mathcal{C}_1$) and smooth ($\mathcal{C}_2$).

The LMO Intersect Bottleneck

Even if we have fast Linear Minimization Oracles for each individual constraint set $\mathcal{C}_i$, there is generally **no simple formula** for the oracle over their intersection:

$$\text{LMO}_{\mathcal{C}_1 \cap \mathcal{C}_2 \cap \cdots \cap \mathcal{C}_m}(\mathbf{G}) = \text{Hard!}$$

Product Space Formulation

To bypass the intersection bottleneck, we create a larger workspace $\boldsymbol{\mathcal{H}} = \mathcal{H}^m$ where each constraint set gets its own independent copy of the variable:

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_m) \in \underbrace{\bigtimes_{i=1}^{m} \mathcal{C}_i = \mathcal{C}_1 \times \mathcal{C}_2 \times \cdots \times \mathcal{C}_m}_{\text{“Product Space”}} \subset \boldsymbol{\mathcal{H}}$$

We then define a weighted averaging operator $A(\boldsymbol{x}) = \sum_{i=1}^{m} \omega_i x_i$ (where weights sum to 1) to combine them back together. Instead of solving one impossible oracle over an intersection, we can now run m simple, parallel oracles over each individual $\mathcal{C}_i$.

The Feasibility Guarantee

If we have some $\boldsymbol{x}^* = (x, x, \ldots, x)$ (i.e., $x \in \bigcap_{i=1}^{m} \mathcal{C}_i$) and their average $A(\boldsymbol{x}^*)$ minimizes our objective f, then that vector x is a solution to our original problem $(*)$!

Split Block-Coordinate Frank-Wolfe Algorithm

Product Space Problem: The Relaxation

The Split Conditional Gradient (SCG) algorithm defines the "consensus set" $\mathcal{D} = \{(x_1, x_2, \ldots, x_m) \in \mathcal{H} \mid x_1 = x_2 = \cdots = x_m\}$. Defined then is the penalized product-space objective:

$$F_{\lambda_t}(\boldsymbol{x}_t) = \underbrace{f(A\boldsymbol{x}_t)}_{\text{Optimality}} + \underbrace{\frac{\lambda_t}{2}\text{dist}^2_{\mathcal{D}}(\boldsymbol{x}_t)}_{\text{Feasibility Penalty}} . \qquad (**)$$

The Split Conditional Gradient (SCG) Loop (Woodstock 2023, [3])

By iteratively increasing λ_t, SCG converges for both convex and non-convex functions without needing intersection oracles:

1. **Update Penalty & Step Size:** Increase λ_t and choose step size γ_t.
2. **Evaluate Gradients:** Compute the gradient of the joint objective $\nabla F_{\lambda_t}(\boldsymbol{x}_t)$.
3. **Parallel FW Updates:** Run m independent Frank-Wolfe updates (one for each constraint set $\mathcal{C}_i$).

Where SCG Falls Short

While the SCG algorithm successfully avoids complex intersection projections, it introduces a new computational burden:

The Simultaneous Update Cost

At every single iteration, standard SCG requires running m separate Linear Minimization Oracles: one for **every** constraint set $\mathcal{C}_i$.

Why this can be inefficient in practice:

- **Asymmetric Costs:** In real scientific models, some constraints have trivial LMOs which can be computed very quickly, while others are far more computationally demanding.
- **Wasted Work:** Forcing every variable to update simultaneously at every step slows the entire algorithm down to the speed of the slowest oracle.

Block Updates & Epoch Size K

Instead of updating the entire vector $\boldsymbol{x} = (x_1, \ldots, x_m)$ at once, we update only a selected **block-coordinate subset** $I_t \subset \{1, 2, \ldots, m\}$ at iteration t.

Assumption of Epoch Limit K

To ensure convergence, we cannot ignore any constraint forever. We assume there exists a fixed integer $K \geqslant 1$ such that every constraint is updated **at least once** within any window of K iterations:

$$\bigcup_{k=0}^{K-1} I_{t+k} = \{1, 2, \ldots, m\} \quad \text{for all } t \in \mathbb{N}.$$

We define every K consecutive iterations as one **epoch**.

- This is the basis of a *block-coordinate* updating scheme for an algorithm, proven to converge for m constraints (Woodstock 2026, [1]).

Split Block-Coordinate Frank-Wolfe Algorithm

A New Algorithm

Our theoretical contribution is as follows: by merging block-coordinate updates with the penalized product-space relaxation defined in the SCG algorithm, we arrive at the **Split Block-Coordinate Frank-Wolfe (SBCFW)** algorithm.

Per-Iteration Complexity Improvement

- **Standard SCG:** Requires m LMO evaluations per step.
- **SBCFW:** Requires only 1 **LMO evaluation** per iteration!

Exploiting Cheap Oracles for "Bonus Progress":

- Because our epoch window K allows flexible scheduling, we allow for updating computationally cheap constraints multiple times per epoch.
- This flexibility generates continuous **bonus progress** toward consensus and optimality without increasing the overall runtime!

Split Block-Coordinate Frank-Wolfe Algorithm

Key Definitions: FW Gap and Bonus Progress

To keep track of progress:

Partial Smoothed Frank-Wolfe Gap ($G_{J,\lambda}(\boldsymbol{x})$) and Accumulated "Bonus Progress" ($A_{t,\lambda} \geqslant 0$)

For any subset of constraints $J \subseteq \{1, \dots, m\}$, we define:

$$G_{J,\lambda}(\boldsymbol{x}) := \sum_{i \in J} \Big(\underbrace{\langle \nabla f(A\boldsymbol{x}) \mid \boldsymbol{x}^i - \boldsymbol{v}^i \rangle}_{\text{Optimality Gap}} + \underbrace{\lambda \langle \boldsymbol{x}^i - (A^* A \boldsymbol{x})^i \mid \boldsymbol{x}^i - \boldsymbol{v}^i \rangle}_{\text{Consensus Gap}} \Big)$$

Because we can update cheap constraints multiple times per epoch, we accumulate guaranteed non-negative bonus progress:

$$A_{t,\lambda} := \sum_{k=1}^{K-1} G_{I_{t+k-1} \cap (I_{t+k} \cup \cdots \cup I_{t+K-1}), \lambda_{t+k}}(\boldsymbol{x}_{t+k}) \geqslant 0$$

Mathematical Tools: The Huber Loss

Our progress bounds take the form of the **Huber loss function**:

The Huber Function $\rho(x, b)$

For inputs $x \in \mathbb{R}$ and $b \in \mathbb{R}^+$:

$$\rho(x, b) := \begin{cases} |x| - \frac{b}{2}, & \text{if } x \geqslant b \quad \text{(Linear regime)} \\ \frac{x^2}{2b}, & \text{if } x \leqslant b \quad \text{(Quadratic regime)} \end{cases}$$

For our analysis:

- We use the fact that $\rho(x, b) \geqslant 0$ for all allowed inputs x and b.

Next, we define two penalty terms here that show up in proofs. Both penalties implicitly (or explicitly) go to zero, since $\text{dist}_{\mathcal{D}}(\boldsymbol{x}_t) \to 0$.

Penalty Definitions

Distance Penalties from Increasing λ

Let $m \in \mathbb{N}$. For epoch m, define the following:

$$c_{t+k} := \langle \boldsymbol{x}_{t+k}^{J_k \setminus I_{t+k-1}} - (A^* A \boldsymbol{x}_{t+k})^{J_k \setminus I_{t+k-1}} \mid \boldsymbol{x}_{t+k}^{J_k \setminus I_{t+k-1}} - \mathrm{LMO}(\nabla^{J_k \setminus I_{t+k-1}} F_{\lambda_{t+k-1}}(\boldsymbol{x}_{t+k})) \rangle,$$

$$b_{t+k} := (1+\gamma_{t+k})\mathrm{dist}_{\mathcal{D}}^2(\boldsymbol{x}_{t+k}) + R\gamma_{t+k}(\mathrm{dist}_{\mathcal{D}}(\boldsymbol{x}_{t+k}) + 1) + \frac{1}{2}(1-\gamma_{t+k})\mathrm{dist}_{\mathcal{D}}(\boldsymbol{x}_{t+k}),$$

$$C_t = \sum_{k=1}^{K-1} (\lambda_{t+k} - \lambda_{t+k-1}) c_{t+k},$$

$$B_t = \frac{1}{2} \sum_{k=0}^{K-1} (\lambda_{t+k+1} - \lambda_{t+k}) b_{t+k}.$$

We note that $B_t \geqslant 0$ for all t, but C_t can be positive or negative (or 0).

Building the Theory: Step-by-Step Progress

How much closer to the solution do we get during a single iteration, and over a full K-step epoch?

Lemma 1: Progress Through P_t (Single Iteration & Epoch Sums)

P_t is defined as a measure of our one-step descent. Using our short-step rules γ_t^i, we prove two fundamental lower bounds:

- **One-Step Bound:** For any un-updated constraints J, progress is bounded by a Huber function of their partial gap.
- **Full Epoch Bound:** Summing over K iterations (one epoch) captures the total global progress plus our accumulated bonus:

$$\sum_{k=0}^{K-1} P_{t+k} \geqslant \rho \left(G_{I,\lambda_t}(\boldsymbol{x}_t) + A_{t,\lambda} + C_t,\ \sum_{k=1}^{K} (L_f + \lambda_{t+k}) R^2 \right)$$

Building the Theory: The Epoch Objective Drop

Because our penalty parameter λ_t increases over time, the objective landscape is changing as we move through an epoch.

Lemma 2: Per-Epoch Progress Through Smoothness

The actual drop in our penalized objective over an epoch satisfies:

$$F_{\lambda_t}(\boldsymbol{x}_t) - F_{\lambda_{t+K}}(\boldsymbol{x}_{t+K}) \geqslant \frac{1}{2}\rho \left(\left| G_{I,\lambda_t}(\boldsymbol{x}_t) + A_{t,\lambda} - C_t \right|, \sum_{k=1}^{K} (L_f + \lambda_{t+k}) R^2 \right) - B_t$$

- **Trade-off:** The first term guarantees nonnegative descent from Frank-Wolfe steps. The negative term (B_t) is the cost of increasing λ_t to enforce feasibility.
- **Key Question:** Can we prove the descent dominates the penalty?

Main Result: Non-Convex Convergence of SBCFW

Let $n \in \mathbb{N}$, R be the diameter of $\bigtimes_{i=1}^{m} C_i$, $D = 3R^2 + R$, $q \in (0, \frac{1}{2})$, λ_0 be a constant, $\lambda_p = \lambda_0(p+1)^q$, and $H_0 := F_{\lambda_0}(\boldsymbol{x}_0) - \inf_{\boldsymbol{x} \in \bigtimes_{i=1}^{m} C_i} f(A\boldsymbol{x})$.

Theorem (Convergence under Constant Epoch Limit K and λ_p Constant Through an Epoch)

For a smooth (possibly non-convex) objective f, summing our epoch progress and applying a Huber inversion yields the suboptimality rate for sufficiently large n:

$$\min_{0 \leqslant p \leqslant n-1} G_{I,\lambda_{pK}}(\boldsymbol{x}_{pK}) \leqslant 2R \sqrt{\frac{K(L_f + \lambda_0(n+1)^q)\left(H_0 + \frac{\lambda_0 D}{2}((n+1)^q - 1)\right)}{n}} - \frac{\sum_{p=0}^{n-1} A_{pK,\lambda}}{n}.$$

In consequence, $\liminf_{n \to \infty} G_{I,\lambda_n}(\boldsymbol{x}_n) = 0$ *for* $q < 0.5$.

Ongoing analysis (Woodstock 2026, [2]) suggests the existence of a subsequence of smoothed FW gaps in $\mathcal{O}(n^{-3/8})$, approaching the non-split FW rate $\mathcal{O}(n^{-1/2})$. This also gives a potential convergence rate of $\mathcal{O}(n^{-1/3})$ on the product space problem $(**)$.

Closing the Loop: Back to the Original Problem

How SBCFW Resolve the Original Intersection Objective

As $\lambda_n \to \infty$, the penalty term $\mathrm{dist}^2_{\mathcal{D}}(\boldsymbol{x}_n)$ is driven to zero, forcing all independent variable copies to merge into a single consensus vector x^* guaranteed to lie in the intersection $\bigcap_{i=1}^m \mathcal{C}_i$.

Concurrently, the vanishing Frank-Wolfe gap ensures that x^* represents an optimal stationary point of the original objective $(*)$.

In conclusion, the **SBCFW** algorithm:

- Bypasses Euclidean projection and repeated LMO costs by requiring only 1 **LMO evaluation per iteration** (down from m), enabling the algorithm to farm "cheap" constraints without bottlenecking on expensive ones.
- Delivers polynomial convergence guarantees for complex constraint intersections that apply identically to both **convex and non-convex** objectives.

Thank you for your attention!

References

[1] Gábor Braun, Jannis Halbey, Sebastian Pokutta, , and Zev Woodstock, *Flexible block-iterative analysis for the frank-wolfe algorithm*, Journal on Optimization Theory and Applications (to appear) (2026).

[2] Antonio Silveti-Falls, Cesare Molinari, and Zev Woodstock, *Frank-wolfe with moreau envelope smoothing for nonsmooth nonconvex problems*, 2026.

[3] Zev Woodstock and Sebastian Pokutta, *Splitting the conditional gradient algorithm*, SIAM Journal on Optimization **35** (2024), no. 1, 347–368.