Blended Conditional Gradients: The Unconditioning of Conditional Gradients

Gábor Braun [1] **Sebastian Pokutta** [1] **Dan Tu** [1] **Stephen Wright** [2]

Abstract

We present a *blended conditional gradient* approach for minimizing a smooth convex function over a polytope P, combining the Frank–Wolfe algorithm (also called conditional gradient) with gradient-based steps, different from away steps and pairwise steps, but still achieving linear convergence for strongly convex functions, along with good practical performance. Our approach retains all favorable properties of conditional gradient algorithms, notably avoidance of projections onto P and maintenance of iterates as sparse convex combinations of a limited number of extreme points of P. The algorithm is *lazy*, making use of inexpensive inexact solutions of the linear programming subproblem that characterizes the conditional gradient approach. It decreases measures of optimality rapidly, both in the number of iterations and in wall-clock time, outperforming even the lazy conditional gradient algorithms of (Braun et al., 2017). We also present a streamlined version of the algorithm that applies when P is the probability simplex.

1. Introduction

A common paradigm in convex optimization is minimization of a smooth convex function f over a polytope P. The conditional gradient (CG) algorithm, also known as "Frank–Wolfe" (Frank & Wolfe, 1956), (Levitin & Polyak, 1966) is enjoying renewed popularity because it can be implemented efficiently to solve important problems in data analysis. It is a first-order method, requiring access to gradients $\nabla f(x)$ and function values $f(x)$. In its original form, CG employs a linear programming (LP) oracle to minimize a linear function over the polytope P at each iteration. The cost of this operation depends on the complexity of P.

In this work, we describe a *blended conditional gradient (BCG)* approach, which takes one of several types of steps on the basis of the gradient ∇f at the current point. Our approach maintains an "active vertex set," consisting of some solutions from previous iterations. Building on (Braun et al., 2017), BCG uses a "weak-separation oracle" to find a vertex of P for which the linear objective attains some fraction of the reduction in f promised by the LP oracle, typically by searching among the current set of active vertices. If no vertex yielding acceptable reduction can be found, the LP oracle used in the original FW algorithm may be deployed. On other iterations, BCG employs a "simplex descent oracle," which takes a step within the convex hull of the active vertices, yielding progress either via reduction in function value (a "descent step") or via culling of the active vertex set (a "drop step"). The size of the active vertex set typically remains small, which benefits both the efficiency of the method and the "sparsity" of the final solution (i.e., its representation as a convex combination of a relatively small number of vertices).

BCG has similar theoretical convergence rates to several other variants of CG that have been studied recently, including pairwise-step and away-step variants and the lazy variants of (Braun et al., 2017). In several cases, we observe better empirical convergence for BCG than for these other variants. While the lazy variant of (Braun et al., 2017) has an advantage over baseline CG when the LP oracle is expensive, our BCG approach consistently outperforms the other variants in more general circumstances, both in per-iteration progress and in wall-clock time.

Related work

There has been an extensive body of work on conditional gradient algorithms; see the excellent overview of (Jaggi, 2013). Here we review only those papers most closely related to our work.

Our main inspiration comes from (Braun et al., 2017; Lan et al., 2017), which introduces the weak-separation oracle

[1]ISyE, Georgia Institute of Technology, Atlanta, GA [2]Computer Sciences Department, University of Wisconsin, Madison, WI. Correspondence to: Gábor Braun <gabor.braun@isye.gatech.edu>, Sebastian Pokutta <sebastian.pokutta@isye.gatech.edu>, Dan Tu <dan.tu@gatech.edu>, Stephen Wright <swright@cs.wisc.edu>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

as a lazy alternative to calling the LP oracle in every iteration. It is influenced too by the method of (Rao et al., 2015), which maintains an active vertex set, using projected descent steps to improve the objective over the convex hull of this set, and culling the set on some steps to keep its size under control. While the latter method is a heuristic with no proven convergence bounds beyond those inherited from the standard Frank–Wolfe method, our BCG algorithm employs a criterion for *optimal trade-off between the various steps*, with a *proven* convergence rate equal to state-of-the-art Frank–Wolfe variants up to a constant factor.

Our main result shows linear convergence of BCG for strongly convex functions. Linearly convergent variants of CG were studied as early as (Guélat & Marcotte, 1986) for special cases and (Garber & Hazan, 2013) for the general case (though the latter work involves very large constants). More recently, linear convergence has been established for various pairwise-step and away-step variants of CG in (Lacoste-Julien & Jaggi, 2015), where the concept of an active vertex set is used to improve performance. Other memory-efficient decomposition-invariant variants were described in (Garber & Meshi, 2016) and (Bashiri & Zhang, 2017). Modification of descent directions and step sizes, reminiscent of the drop steps used in BCG, have been considered by (Freund & Grigas, 2016; Freund et al., 2017). The use of an inexpensive oracle based on a subset of the vertices of P, as an alternative to the full LP oracle, has been considered in (Kerdreux et al., 2018b). (Garber et al., 2018) proposes a fast variant of conditional gradients for matrix recovery problems.

BCG is quite distinct from the fully-corrective Frank–Wolfe algorithm (FCFW) (see, for example, (Holloway, 1974; Lacoste-Julien & Jaggi, 2015)). Both approaches maintain active vertex sets, generate iterates that lie in the convex hulls of these sets, and alternate between Frank–Wolfe steps generating new vertices and correction steps optimizing within the current active vertex set. However, convergence analyses of the FCFW algorithm assume that the correction steps have unit cost, though they can be quite expensive in practice, requiring multiple evaluations of the gradient ∇f. For BCG, by contrast, we assume only a *single* step of gradient descent type having unit cost (disregarding cost of line search). For further explanation of the differences between BCG and FCFW, see computational results in Figure 12 and discussion in Appendix D.

Contribution

Our contribution can be summarized as follows:

Blended Conditional Gradients (BCG). The BCG approach blends different types of descent steps: the traditional CG steps of (Frank & Wolfe, 1956), the lazified CG steps of (Braun et al., 2017), and gradient descent steps over the convex hull of the current active vertex set. It avoids projections onto P or onto the convex hull of the active vertices, and does not use away steps and pairwise steps, which are elements of other popular variants of CG. It achieves linear convergence for strongly convex functions (see Theorem 3.1), and $O(1/t)$ convergence after t iterations for general smooth functions. While the linear convergence proof of the Away-step Frank–Wolfe Algorithm (Lacoste-Julien & Jaggi, 2015, Theorem 1, Footnote 4) requires the objective function f to be defined on the Minkowski sum $P - P + P$, BCG does not need f to be defined outside the polytope P. The algorithm has complexity comparable to pairwise-step or away-step variants of conditional gradients, both in per-iteration running time and in the space required to store vertices and iterates. It is affine-invariant and parameter-free; estimates of such parameters as smoothness, strong convexity, or the diameter of P are not required. It maintains iterates as (often sparse) convex combinations of vertices, typically much sparser than the baseline CG methods, a property that is important for some applications. Such sparsity is due to the aggressive reuse of active vertices, and the fact that new vertices are added only as a kind of last resort. In wall-clock time as well as per-iteration progress, our computational results show that BCG can be orders of magnitude faster than competimg CG methods on some problems.

Simplex Gradient Descent (SiGD). In Section 4, we describe a new projection-free gradient descent procedure for minimizing a smooth function over the probability simplex, which can be used to implement the "simplex descent oracle" required by BCG.

Computational Experiments. We demonstrate the excellent computational behavior of BCG compared to other CG algorithms on standard problems, including video co-localization, sparse regression, structured SVM training, and structured regression. We observe significant computational speedups and in several cases empirically better convergence rates.

Outline

We summarize preliminary material in Section 2, including the two oracles that are the foundation of our BCG procedure. BCG is described and analyzed in Section 3, establishing linear convergence rates. The simplex gradient descent routine, which implements the simplex descent oracle, is described in Section 4. Our computational experiments are summarized in Section 5; more extensive experiments appear in Appendix D. Variants on the analysis and other auxiliary materials are relegated to the appendix. We mention in particular a variant of BCG that applies when P is the probability simplex, a special case that admits several simplifications and improvements to the analysis.

2. Preliminaries

We use the following notation: e_i is the i-th coordinate vector, $\mathbb{1} := (1, \ldots, 1) = e_1 + e_2 + \cdots$ is the all-ones vector, $\|\cdot\|$ denotes the Euclidean norm (ℓ_2-norm), $D = \operatorname{diam}(P) = \sup_{u,v \in P} \|u - v\|_2$ is the ℓ_2-diameter of P, and $\operatorname{conv} S$ denotes the convex hull of a set S of points. The *probability simplex* $\Delta^k := \operatorname{conv}\{e_1, \ldots, e_k\}$ is the convex hull of the coordinate vectors in dimension k.

Let f be a differentiable convex function. Recall that f is L-*smooth* if $f(y) - f(x) - \nabla f(x)(y - x) \leq L\|y - x\|^2/2$ for all $x, y \in P$. The function f has *curvature* C if $f(\gamma y + (1 - \gamma)x) \leq f(x) + \gamma \nabla f(x)(y - x) + C\gamma^2/2$, for all $x, y \in P$ and $0 \leq \gamma \leq 1$. (Note that an L-smooth function always has curvature $C \leq LD^2$.) Finally, f is *strongly convex* if for some $\alpha > 0$ we have $f(y) - f(x) - \nabla f(x)(y - x) \geq \alpha\|y - x\|^2/2$, for all $x, y \in P$. We use the following fact about strongly convex function when optimizing over P.

Fact 2.1 (Geometric strong convexity guarantee)**.** *(Lacoste-Julien & Jaggi, 2015, Theorem 6 and Eq. (28)) Given a strongly convex function f, there is a value $\mu > 0$ called the* geometric strong convexity *such that*

$$f(x) - \min_{y \in P} f(y) \leq \frac{\left(\max_{y \in S, z \in P} \nabla f(x)(y - z)\right)^2}{2\mu}$$

for any $x \in P$ and for any subset S of the vertices of P for which x lies in the convex hull of S.

The value of μ depends both on f and the geometry of P.

2.1. Simplex Descent Oracle

Given a convex objective function f and an ordered finite set $S = \{v_1, \ldots, v_k\}$ of points, we define $f_S \colon \Delta^k \to \mathbb{R}$ as follows:

$$f_S(\lambda) := f\left(\sum_{i=1}^{k} \lambda_i v_i\right). \tag{1}$$

When f_S is L_{f_S}-smooth, Oracle 1 returns an improving point x' in $\operatorname{conv} S$ together with a vertex set $S' \subseteq S$ such that $x' \in \operatorname{conv} S'$.

Oracle 1 Simplex Descent Oracle $\mathrm{SiDO}(x, S, f)$

Input: finite set $S \subseteq \mathbb{R}^n$, point $x \in \operatorname{conv} S$, convex smooth function $f \colon \operatorname{conv} S \to \mathbb{R}^n$;

Output: finite set $S' \subseteq S$, point $x' \in \operatorname{conv} S'$ satisfying either

drop step: $f(x') \leq f(x)$ and $S' \neq S$

descent step:

$$f(x) - f(x') \geq [\max_{u,v \in S} \nabla f(x)(u - v)]^2/(4L_{f_S})$$

In Section 4 we provide an implementation (Algorithm 2) of this oracle via a *single descent step*, which avoids projection and does not require knowledge of the smoothness parameter L_{f_S}.

2.2. Weak-Separation Oracle

Oracle 2 Weak-Separation Oracle $\mathrm{LPsep}_P(c, x, \Phi, K)$

Input: linear objective $c \in \mathbb{R}^n$, point $x \in P$, accuracy $K \geq 1$, gap estimate $\Phi > 0$;

Output: Either (1) vertex $y \in P$ with $c(x - y) \geq \Phi/K$, or (2) **false**: $c(x - z) \leq \Phi$ for all $z \in P$.

The weak-separation oracle Oracle 2 was introduced in (Braun et al., 2017) to replace the LP oracle traditionally used in the CG method. Provided with a point $x \in P$, a linear objective c, a target reduction value $\Phi > 0$, and an inexactness factor $K \geq 1$, it decides whether there exists $y \in P$ with $cx - cy \geq \Phi/K$, or else certifies that $cx - cz \leq \Phi$ for all $z \in P$. In our applications, $c = \nabla f(x)$ is the gradient of the objective at the current iterate x. Oracle 2 could be implemented simply by the standard LP oracle of minimizing cz over $z \in P$. However, it allows more efficient implementations, including the following. (1) *Caching*: testing previously obtained vertices $y \in P$ (specifically, vertices in the current active vertex set) to see if one of them satisfies $cx - cy \geq \Phi/K$. If not, the traditional LP oracle could be called to either find a new vertex of P satisfying this bound, or else to certify that $cx - cz \leq \Phi$ for all $z \in P$, and (2) *Early Termination*: Terminating the LP procedure as soon as a vertex of P has been discovered that satisfies $cx - cy \geq \Phi/K$. (This technique requires an LP implementation that generates vertices as iterates.) If the LP procedure runs to termination without finding such a point, it has certified that $cx - cz \leq \Phi$ for all $z \in P$. In (Braun et al., 2017) these techniques resulted in orders-of-magnitude speedups in wall-clock time in the computational tests, as well as sparse convex combinations of vertices for the iterates x_t, a desirable property in many contexts.

3. Blended Conditional Gradients

Our BCG approach is specified as Algorithm 1. We discuss the algorithm in this section and establish its convergence rate. The algorithm expresses each iterate x_t, $t = 0, 1, 2, \ldots$ as a convex combination of the elements of the active vertex set, denoted by S_t, as in the Pairwise and Away-step variants of CG. At each iteration, the algorithm calls either Oracle 1 or Oracle 2 in search of the next iterate, whichever promises the smaller function value, using a test in Line 6 based on an estimate of the dual gap. The same greedy principle is used in the Away-step CG approach, and its lazy variants. A critical role in the algorithm (and particularly in the test of Line 6) is played by the value

Φ_t, which is a current estimate of the primal gap — the difference between the current function value $f(x_t)$ and the optimal function value over P. When Oracle 2 returns **false**, the curent value of Φ_t is discovered to be an overestimate of the dual gap, so it is halved (Line 13) and we proceed to the next iteration. In subsequent discussion, we refer to Φ_t as the "gap estimate."

Algorithm 1 Blended Conditional Gradients (BCG)

Input: smooth convex function f, start vertex $x_0 \in P$, weak-separation oracle LPsep_P, accuracy $K \geq 1$
Output: points x_t in P for $t = 1, \ldots, T$

1: $\Phi_0 \leftarrow \max_{v \in P} \nabla f(x_0)(x_0 - v)/2$ {Initial gap estimate}
2: $S_0 \leftarrow \{x_0\}$
3: **for** $t = 0$ **to** $T - 1$ **do**
4: $\quad v_t^A \leftarrow \mathrm{argmax}_{v \in S_t} \nabla f(x_t) v$
5: $\quad v_t^{FW-S} \leftarrow \mathrm{argmin}_{v \in S_t} \nabla f(x_t) v$
6: $\quad$ **if** $\nabla f(x_t)(v_t^A - v_t^{FW-S}) \geq \Phi_t$ **then**
7: $\quad\quad x_{t+1}, S_{t+1} \leftarrow \mathrm{SiDO}(x_t, S_t)$ {either a drop step or a descent step}
8: $\quad\quad \Phi_{t+1} \leftarrow \Phi_t$
9: $\quad$ **else**
10: $\quad\quad v_t \leftarrow \mathrm{LPsep}_P(\nabla f(x_t), x_t, \Phi_t, K)$
11: $\quad\quad$ **if** $v_t =$ **false then**
12: $\quad\quad\quad x_{t+1} \leftarrow x_t$
13: $\quad\quad\quad \Phi_{t+1} \leftarrow \Phi_t/2$ {gap step}
14: $\quad\quad\quad S_{t+1} \leftarrow S_t$
15: $\quad\quad$ **else**
16: $\quad\quad\quad x_{t+1} \leftarrow \mathrm{argmin}_{x \in [x_t, v_t]} f(x)$ {FW step, with line search}
17: $\quad\quad\quad$ Choose $S_{t+1} \subseteq S_t \cup \{v_t\}$ minimal such that $x_{t+1} \in \mathrm{conv}\, S_{t+1}$.
18: $\quad\quad\quad \Phi_{t+1} \leftarrow \Phi_t$
19: $\quad\quad$ **end if**
20: $\quad$ **end if**
21: **end for**

In Line 17, the active set S_{t+1} is required to be minimal. By Caratheodory's theorem, this requirement ensures that $|S_{t+1}| \leq \dim P + 1$. In practice, the S_t are invariably small and no explicit reduction in size is necessary. The key requirement, in theory and practice, is that if after a call to Oracle SiDO the new iterate x_{t+1} lies on a face of the convex hull of the vertices in S_t, then at least one element of S_t is dropped to form S_{t+1}. This requirement ensures that the local pairwise gap in Line 6 is not too large due to stale vertices in S_t, which can block progress. Small size of the sets S_t is crucial to the efficiency of the algorithm, in rapidly determining the maximizer and minimizer of $\nabla f(x_t)$ over the active set S_t in Lines 4 and 5.

The constants in the convergence rate described in our main theorem (Theorem 3.1 below) depend on a modified curvature-like parameter of the function f. Given a vertex set S of P, recall from Section 2.1 the smoothness parameter L_{f_S} of the function $f_S \colon \Delta^k \to \mathbb{R}$ defined by (1). Define the *simplicial curvature* C^Δ to be

$$C^\Delta := \max_{S \colon |S| \leq 2 \dim P} L_{f_S} \tag{2}$$

to be the maximum of the L_{f_S} over all possible active sets. This affine-invariant parameter depends both on the shape of P and the function f. This is the relative smoothness constant $L_{f,A}$ from the predecessor of (Gutman & Peña, 2019), namely (Gutman & Peña, 2018, Definiton 2a), with an additional restriction: the simplex is restricted to faces of dimension at most $2 \dim P$, which appears as a bound on the size of S in our formulation. This restriction improves the constant by removing dependence on the number of vertices of the polytope, and can probably replace the original constant in convergence bounds. We can immediately see the effect in the common case of L-smooth functions, that the simplicial curvature is of reasonable magnitude, specifically,

$$C^\Delta \leq \frac{LD^2(\dim P)}{2},$$

where D is the diameter of P. This result follows from (2) and the bound on L_{f_S} from Lemma A.1 in the appendix. This bound is not directly comparable with the upper bound $L_{f,A} \leq LD^2/4$ in (Gutman & Peña, 2018, Corollary 2), because the latter uses the 1-norm on the standard simplex, while we use the 2-norm, the norm used by projected gradients and our simplex gradient descent. The additional factor $\dim P$ is explained by the n-dimensional standard simplex having constant minimum width 2 in 1-norm, but having minimum width dependent on the dimension n (specifically, $\Theta(1/\sqrt{n})$) in the 2-norm. Recall that the minimum width of a convex body $P \subseteq \mathbb{R}^n$ in norm $\|\cdot\|$ is $\min_\phi \max_{u,v \in P} \phi(u - v)$, where ϕ runs over all linear maps $\mathbb{R}^n \to \mathbb{R}$ having dual norm $\|\phi\|_* = 1$. For the 2-norm, this is just the minimum distance between parallel hyperplanes such that P lies between the two hyperplanes.

For another comparison, recall the curvature bound $C \leq LD^2$. Note, however, that the algorithm and convergence rate below are affine invariant, and the only restriction on the function f is that it has finite simplicial curvature. This restriction readily provides the curvature bound

$$C \leq 2C^\Delta, \tag{3}$$

where the factor 2 arises as the square of the diameter of the standard simplex Δ^k. (See Lemma A.2 in the appendix for details.) Note that S is allowed to be large enough so that every point of P is in the convex hull of some vertex subset S, by Caratheodory's theorem, and that the simplicial curvature provides an upper bound on the curvature

We describe the convergence of BCG (Algorithm 1) in the following theorem.

Theorem 3.1. *Let f be a strongly convex, smooth function over the polytope P with simplicial curvature C^Δ and geometric strong convexity μ. Then Algorithm 1 ensures $f(x_T) - f(x^*) \leq \varepsilon$, where x^* is an optimal solution to f in P for some iteration index T that satisfies*

$$T \leq \left\lceil \log \frac{2\Phi_0}{\varepsilon} \right\rceil + 8K \left\lceil \log \frac{\Phi_0}{2KC^\Delta} \right\rceil + \frac{64K^2C^\Delta}{\mu} \left\lceil \log \frac{4KC^\Delta}{\varepsilon} \right\rceil = O\left(\frac{C^\Delta}{\mu} \log \frac{\Phi_0}{\varepsilon} \right), \quad (4)$$

where $\log$ *denotes logarithms to the base* 2.

For smooth but not necessarily strongly convex functions f, the algorithm ensures $f(x_T) - f(x^*) \leq \varepsilon$ after $T = O(\max\{C^\Delta, \Phi_0\}/\varepsilon)$ iterations by a similar argument, which is omitted.

Proof. The proof tracks that of (Braun et al., 2017). We divide the iteration sequence into epochs that are demarcated by the *gap steps*, that is, the iterations for which the weak-separation oracle (Oracle 2) returns the value **false**, which results in Φ_t being halved for the next iteration. We then bound the number of iterates within each epoch. The result is obtained by aggregating across epochs.

We start by a well-known bound on the function value using the Frank–Wolfe point $v_t^{FW} := \operatorname{argmin}_{v \in P} \nabla f(x_t) v$ at iteration t, which follows from convexity:

$$f(x_t) - f(x^*) \leq \nabla f(x_t)(x_t - x^*) \leq \nabla f(x_t)(x_t - v_t^{FW}).$$

If iteration $t-1$ is a gap step, we have using $x_t = x_{t-1}$ and $\Phi_t = \Phi_{t-1}/2$ that

$$f(x_t) - f(x^*) \leq \nabla f(x_t)(x_t - v_t^{FW}) \leq 2\Phi_t. \quad (5)$$

This bound also holds at $t = 0$, by definition of Φ_0. Thus Algorithm 1 is guaranteed to satisfy $f(x_T) - f(x^*) \leq \varepsilon$ at some iterate T such that $T-1$ is a gap step and $2\Phi_T \leq \varepsilon$. Therefore, the total number of gap steps N_Φ required to reach this point satisfies

$$N_\Phi \leq \left\lceil \log \frac{2\Phi_0}{\varepsilon} \right\rceil, \quad (6)$$

which is also a bound on the total number of epochs. The next stage of the proof finds bounds on the number of iterations of each type within an individual epoch.

If iteration $t-1$ is a gap step, we have $x_t = x_{t-1}$ and $\Phi_t = \Phi_{t-1}/2$, and because the condition is false at Line 6 of Algorithm 1, we have

$$\nabla f(x_t)(v_t^A - x_t) \leq \nabla f(x_t)(v_t^A - v_t^{FW-S}) \leq 2\Phi_t. \quad (7)$$

This condition also holds trivially at $t = 0$, since $v_0^A = v_0^{FW-S} = x_0$. By summing (5) and (7), we obtain $\nabla f(x_t)(v_t^A - v_t^{FW}) \leq 4\Phi_t$, so it follows from Fact 2.1 that $f(x_t) - f(x^*) \leq \frac{[\nabla f(x_t)(v_t^A - v_t^{FW})]^2}{2\mu} \leq \frac{8\Phi_t^2}{\mu}$. By combining this inequality with (5), we obtain

$$f(x_t) - f(x^*) \leq \min\left\{8\Phi_t^2/\mu, 2\Phi_t\right\}, \quad (8)$$

for all t such that either $t = 0$ or else $t-1$ is a gap step. In fact, (8) holds for *all* t, because (1) the sequence of function values $\{f(x_s)\}_s$ is non-increasing; and (2) $\Phi_s = \Phi_t$ for all s in the epoch that starts at iteration t.

We now consider the epoch that starts at iteration t, and use s to index the iterations within this epoch. Note that $\Phi_s = \Phi_t$ for all s in this epoch.

We distinguish three types of iterations besides gap step. The first type is a *Frank–Wolfe* step, taken when the weak-separation oracle returns an improving vertex $v_s \in P$ such that $\nabla f(x_s)(x_s - v_s) \geq \Phi_s/K = \Phi_t/K$ (Line 16). Using the definition of curvature C, we have by standard Frank–Wolfe arguments that (c.f., (Braun et al., 2017)).

$$\begin{aligned} f(x_s) - f(x_{s+1}) &\geq \frac{\Phi_s}{2K} \min\left\{1, \frac{\Phi_s}{KC}\right\} \\ &\geq \frac{\Phi_t}{2K} \min\left\{1, \frac{\Phi_t}{2KC^\Delta}\right\}, \end{aligned} \quad (9)$$

where we used $\Phi_s = \Phi_t$ and $C \leq 2C^\Delta$ (from (3)). We denote by N_{FW}^t the number of Frank–Wolfe iterations in the epoch starting at iteration t.

The second type of iteration is a *descent step*, in which Oracle SiDO (Line 7) returns a point x_{s+1} that lies in the relative interior of $\operatorname{conv} S_s$ and with strictly smaller function value. We thus have $S_{s+1} = S_s$ and, by the definition of Oracle SiDO, together with (2), it follows that

$$\begin{aligned} f(x_s) - f(x_{s+1}) &\geq \frac{[\nabla f(x_s)(v_s^A - v_s^{FW-S})]^2}{4C^\Delta} \\ &\geq \frac{\Phi_s^2}{4C^\Delta} = \frac{\Phi_t^2}{4C^\Delta}. \end{aligned} \quad (10)$$

We denote by N_{desc}^t the number of descent steps that take place in the epoch that starts at iteration t.

The third type of iteration is one in which Oracle 1 returns a point x_{s+1} lying on a face of the convex hull of S_s, so that S_{s+1} is strictly smaller than S_s. Similarly to the Away-step Frank–Wolfe algorithm of (Lacoste-Julien & Jaggi, 2015), we call these steps *drop steps*, and denote by N_{drop}^t the number of such steps that take place in the epoch that starts at iteration t. Note that since S_s is expanded only at Frank–Wolfe steps, and then only by at most one element, the *total* number of drop steps across the whole algorithm cannot exceed the total number of Frank–Wolfe steps. We

use this fact and (6) in bounding the total number of iterations T required for $f(x_T) - f(x^*) \leq \varepsilon$:

$$\begin{aligned} T &\leq N_\Phi + N_{\text{desc}} + N_{\text{FW}} + N_{\text{drop}} \\ &\leq \left\lceil \log \frac{2\Phi_0}{\varepsilon} \right\rceil + N_{\text{desc}} + 2N_{\text{FW}} \\ &= \left\lceil \log \frac{2\Phi_0}{\varepsilon} \right\rceil + \sum_{t:\text{epoch start}} (N_{\text{desc}}^t + 2N_{\text{FW}}^t). \end{aligned} \tag{11}$$

Here N_{desc} denotes the total number of descent steps, N_{FW} the total number of Frank–Wolfe steps, and N_{drop} the total number of drop steps, which is bounded by N_{FW}, as just discussed.

Next, we seek bounds on the iteration counts N_{desc}^t and N_{FW}^t within the epoch starting with iteration t. For the total decrease in function value during the epoch, Equations (9) and (10) provide a lower bound, while $f(x_t) - f(x^*)$ is an obvious upper bound, leading to the following estimate using (8).

- If $\Phi_t \geq 2KC^\Delta$ then

$$\begin{aligned} 2\Phi_t \geq f(x_t) - f(x^*) &\geq N_{\text{desc}}^t \frac{\Phi_t^2}{4C^\Delta} + N_{\text{FW}}^t \frac{\Phi_t}{2K} \\ &\geq N_{\text{desc}}^t \frac{\Phi_t K}{2} + N_{\text{FW}}^t \frac{\Phi_t}{2K} \geq (N_{\text{desc}}^t + 2N_{\text{FW}}^t) \frac{\Phi_t}{4K}, \end{aligned}$$

  hence

$$N_{\text{desc}}^t + 2N_{\text{FW}}^t \leq 8K. \tag{12}$$

- If $\Phi_t < 2KC^\Delta$, a similar argument provides

$$\begin{aligned} \frac{8\Phi_t^2}{\mu} \geq f(x_t) - f(x^*) &\geq N_{\text{desc}}^t \frac{\Phi_t^2}{4C^\Delta} + N_{\text{FW}}^t \frac{\Phi_t^2}{4K^2C^\Delta} \\ &\geq (N_{\text{desc}}^t + 2N_{\text{FW}}^t) \frac{\Phi_t^2}{8K^2C^\Delta}, \end{aligned}$$

  leading to

$$N_{\text{desc}}^t + 2N_{\text{FW}}^t \leq \frac{64K^2C^\Delta}{\mu}. \tag{13}$$

There are at most

$$\left\lceil \log \frac{\Phi_0}{2KC^\Delta} \right\rceil \text{ epochs in the regime with } \Phi_t \geq 2KC^\Delta,$$
$$\left\lceil \log \frac{2KC^\Delta}{\varepsilon/2} \right\rceil \text{ epochs in the regime with } \Phi_t < 2KC^\Delta.$$

Combining (11) with the bounds (12) and (13) on N_{FW}^t and N_{desc}^t, we obtain (4). □

4. Simplex Gradient Descent

Here we describe the Simplex Gradient Descent approach (Algorithm 2), an implementation of the SiDO oracle (Oracle 1). Algorithm 2 requires only $O(|S|)$ operations beyond the evaluation of $\nabla f(x)$ and the cost of line search. (It is assumed that x is represented as a convex combination of vertices of P, which is updated during Oracle 1.) Apart from the (trivial) computation of the projection of $\nabla f(x)$ onto the linear space spanned by Δ^k, no projections are computed. Thus, Algorithm 2 is typically faster even than a step of Frank–Wolfe, for typical small sets S.

Alternative implementations of Oracle 1 are described in Section C.1. Section C.2 describes the special case in which P itself is a probability simplex. Here, BCG and its oracles are combined into a single, simple method with better constants in the convergence bounds.

In the algorithm, the form $c\mathbb{1}$ denotes the scalar product of c and $\mathbb{1}$, i.e., the sum of entries of c.

Algorithm 2 Simplex Gradient Descent Step (SiGD)

Input: polyhedron P, smooth convex function $f\colon P \to \mathbb{R}$, subset $S = \{v_1, v_2, \ldots, v_k\}$ of vertices of P, point $x \in \operatorname{conv} S$

Output: set $S' \subseteq S$, point $x' \in \operatorname{conv} S'$

1: Decompose x as a convex combination $x = \sum_{i=1}^k \lambda_i v_i$, with $\sum_{i=1}^k \lambda_i = 1$ and $\lambda_i \geq 0$, $i = 1, 2, \ldots, k$
2: $c \leftarrow [\nabla f(x)v_1, \ldots, \nabla f(x)v_k]$ $\{c = \nabla f_S(\lambda)$; see (1)$\}$
3: $d \leftarrow c - (c\mathbb{1})\mathbb{1}/k$ {Projection onto the hyperplane of Δ^k}
4: **if** $d = 0$ **then**
5: **return** $x' = v_1$, $S' = \{v_1\}$ {Arbitrary vertex}
6: **end if**
7: $\eta \leftarrow \max\{\eta \geq 0 : \lambda - \eta d \geq 0\}$
8: $y \leftarrow x - \eta \sum_i d_i v_i$
9: **if** $f(x) \geq f(y)$ **then**
10: $x' \leftarrow y$
11: Choose $S' \subseteq S$, $S' \neq S$ with $x' \in \operatorname{conv} S'$.
12: **else**
13: $x' \leftarrow \operatorname{argmin}_{z \in [x,y]} f(z)$
14: $S' \leftarrow S$
15: **end if**
16: **return** x', S'

To verify the validity of Algorithm 2 as an implementation of Oracle 1, note first that since y lies on a face of $\operatorname{conv} S$ by definition, it is always possible to choose a proper subset $S' \subseteq S$ in Line 11, for example, $S' := \{v_i : \lambda_i > \eta d_i\}$. The following lemma shows that with the choice $h := f_S$, Algorithm 2 correctly implements Oracle 1.

Lemma 4.1. *Let Δ^k be the probability simplex in k dimensions and suppose that $h\colon \Delta^k \to \mathbb{R}$ is an L_h-smooth*

function. Given some $\lambda \in \Delta^k$*, define* $d := \nabla h(\lambda) - (\nabla h(\lambda)\mathbb{1}/k)\mathbb{1}$ *and let* $\eta \geq 0$ *be the largest value for which* $\tau := \lambda - \eta d \geq 0$*. Let* $\lambda' := \operatorname{argmin}_{z \in [\lambda, \tau]} h(z)$*. Then either* $h(\lambda) \geq h(\tau)$ *or*

$$h(\lambda) - h(\lambda') \geq \frac{[\max_{1 \leq i,j \leq k} \nabla h(\lambda)(e_i - e_j)]^2}{4L_h}.$$

Proof. Let $g(\zeta) := h(\zeta - (\zeta\mathbb{1})\mathbb{1}/k)$, then $\nabla g(\zeta) = \nabla h(\zeta - (\zeta\mathbb{1})\mathbb{1}/k) - (\nabla h(\zeta - (\zeta\mathbb{1})\mathbb{1}/k)\mathbb{1})\mathbb{1}/k$, and g is clearly L_h-smooth, too. In particular, $\nabla g(\lambda) = d$.

From standard gradient descent bounds, not repeated here, we have the following inequalities, for $\gamma \leq \min\{\eta, 1/L_h\}$:

$$\begin{aligned} h(\lambda) - h(\lambda - \gamma d) &= g(\lambda) - g(\lambda - \gamma \nabla g(\lambda)) \\ &\geq \gamma \frac{\|\nabla g(\lambda)\|_2^2}{2} \geq \gamma \frac{[\max_{1 \leq i,j \leq k} \nabla g(\lambda)(e_i - e_j)]^2}{4} \\ &= \gamma \frac{[\max_{1 \leq i,j \leq k} \nabla h(\lambda)(e_i - e_j)]^2}{4}, \end{aligned} \tag{14}$$

where the second inequality uses that the ℓ_2-diameter of the Δ^k is 2, and the last equality follows from $\nabla g(\lambda)(e_i - e_j) = \nabla h(\lambda)(e_i - e_j)$.

When $\eta \geq 1/L_h$, we conclude that $h(\lambda') \leq h(\lambda - (1/L_h)d) \leq h(\lambda)$, hence

$$h(\lambda) - h(\lambda') \geq \frac{[\max_{i,j \in \{1,2,\ldots,k\}} \nabla h(\lambda)(e_i - e_j)]^2}{4L_h},$$

which is the second case of the lemma. When $\eta < 1/L_h$, then setting $\gamma = \eta$ in (14) clearly provides $h(\lambda) - h(\tau) \geq 0$, which is the first case of the lemma. □

5. Computational Experiments (Summary)

To compare our experiments to previous work, we used problems and instances similar to those in (Lacoste-Julien & Jaggi, 2015; Garber & Meshi, 2016; Rao et al., 2015; Braun et al., 2017; Lan et al., 2017). These include structured regression, sparse regression, video co-localization, sparse signal recovery, matrix completion, and Lasso. We compared various algorithms denoted by the following acronyms: our algorithm (BCG), the Away-step Frank–Wolfe algorithm (ACG) and the Pairwise Frank–Wolfe algorithm (PCG) from (Lacoste-Julien & Jaggi, 2015; Garber & Meshi, 2016), the vanilla Frank–Wolfe algorithm (CG), as well as their lazified versions from (Braun et al., 2017). We add a prefix 'L' for the lazified versions. Figure 1 summarizes our results on four test problems. Further details and more extensive computational results are reported in Appendix D.

Performance Comparison

We implemented Algorithm 1 as outlined above and used SiGD (Algorithm 2) for the descent steps as described in Section 4. For line search in Line 13 of Algorithm 2, we perform standard backtracking, and for Line 16 of Algorithm 1, we do ternary search. In Figure 1, each of the four plots itself contains four subplots depicting results of four variants of CG on a single instance. The two subplots in each upper row measure progress in the logarithm (to base 2) of the function value, while the two subplots in each lower row report the logarithm of the gap estimate Φ_t from Algorithm 1. The subplots in the left column of each plot report performance in terms of number of iterations, while the subplots in the right column report wall-clock time.

As discussed earlier, $2\Phi_t$ upper bounds the primal gap (the difference between the function value at the current iterate and the optimal function value). The lazified algorithms (including BCG) halve Φ_t occasionally, which provides a stair-like appearance in the graphs. In implementations, if a stronger bound on the primal gap is available (e.g., from an LP oracle call), we reset Φ_t to half of that value, thus removing unnecessary successive halving steps. For the non-lazified algorithms, we plot the dual gap $\max_{v \in P} \nabla f(x_t)(x_t - v)$ as a gap estimate. The dual gap does not necessarily decrease in a monotone fashion (though of course the primal gap is monotone decreasing), so the plots have a zigzag appearance in some instances.

6. Final Remarks

In (Lan et al., 2017), an accelerated method based on weak separation and conditional gradient sliding was described. This method provided optimal tradeoffs between (stochastic) first-order oracle calls and weak-separation oracle calls. An open question is whether the same tradeoffs and acceleration could be realized by replacing SiGD (Algorithm 2) by an accelerated method.

After an earlier version of our work appeared online, (Kerdreux et al., 2018a) introduced the *Hölder Error Bound condition* (also known as *sharpness* or the *Łojasiewicz growth condition*). This is a family of conditions parameterized by $0 < p \leq 1$, interpolating between strongly convex ($p = 0$) and convex functions ($p = 1$). For such functions, convergence rate $O(1/\varepsilon^p)$ has been shown for Away-step Frank–Wolfe algorithms, among others. Our analysis can be similarly extended to objective functions satisfying this condition, leading to similar convergence rates.

Acknowledgements

We are indebted to Swati Gupta for the helpful discussions. Research reported in this paper was partially supported by NSF CAREER award CMMI-1452463, and also NSF Awards 1628384, 1634597, and 1740707; Subcontract 8F-30039 from Argonne National Laboratory; and Award N660011824020 from the DARPA Lagrange Program.

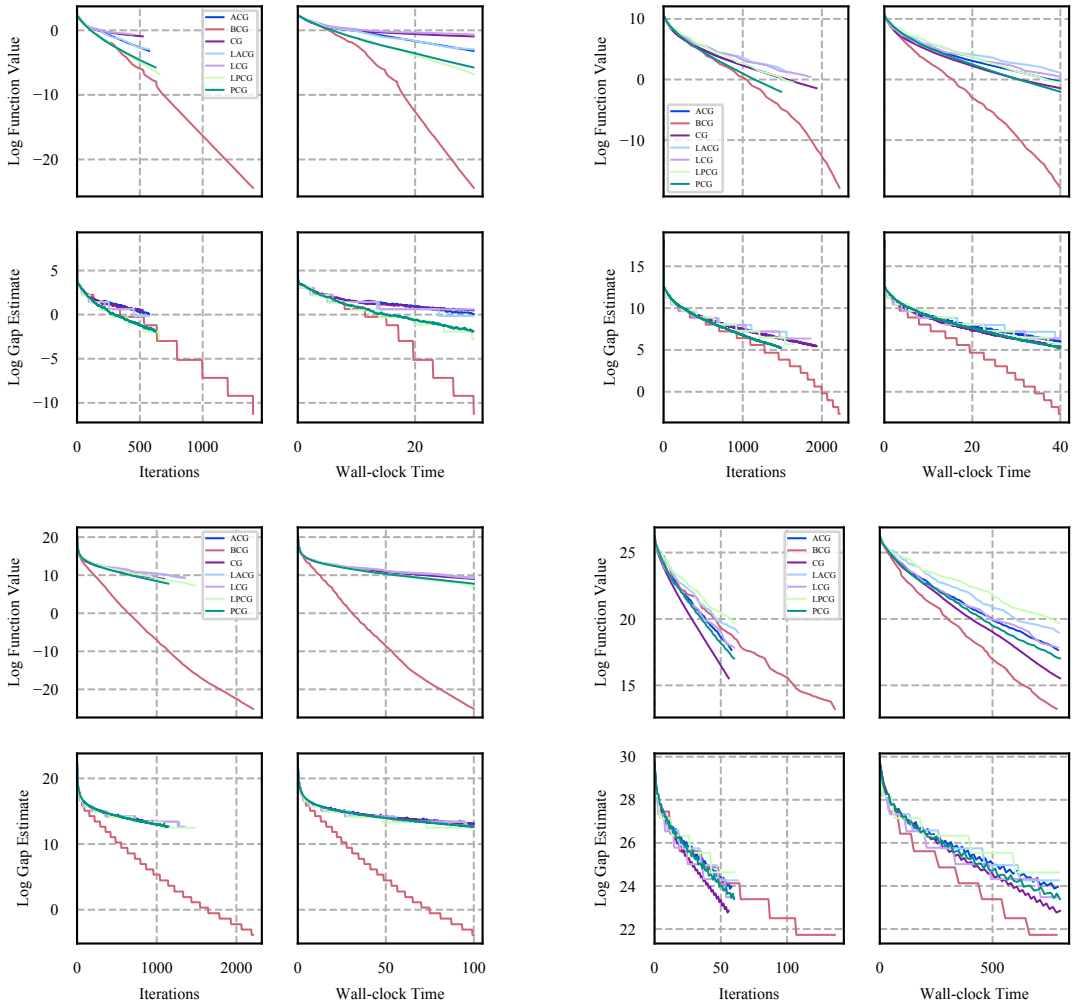

Figure 1. Four representative examples. (Upper-left) Sparse signal recovery: $\min_{x \in \mathbb{R}^n : \|x\|_1 \leq \tau} \|y - \Phi x\|_2^2$, where Φ is of size 1000×3000 with density 0.05. BCG made 1402 iterations with 155 calls to the weak-separation oracle LPsep_P. The final solution is a convex combination of 152 vertices. (Upper-right) Lasso. We solve $\min_{x \in P} \|Ax - b\|^2$ with P being the (scaled) ℓ_1-ball. A is a 400×2000 matrix with 100 non-zeros. BCG made 2130 iterations, calling LPsep_P 477 times, with the final solution being a convex combination of 462 vertices. (Lower-left) Structured regression over the Birkhoff polytope of dimension 50. BCG made 2057 iterations with 524 calls to LPsep_P. The final solution is a convex combination of 524 vertices. (Lower-right) Video co-localization over `netgen_12b` polytope with an underlying 5000-vertex graph. BCG made 140 iterations, with 36 calls to LPsep_P. The final solution is a convex combination of 35 vertices.

References

Bashiri, M. A. and Zhang, X. Decomposition-invariant conditional gradient for general polytopes with line search. In *Advances in Neural Information Processing Systems*, pp. 2687–2697, 2017.

Braun, G., Pokutta, S., and Zink, D. Lazifying conditional gradient algorithms. *Proceedings of ICML*, 2017.

Frank, M. and Wolfe, P. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2): 95–110, 1956.

Freund, R. M. and Grigas, P. New analysis and results for the Frank–Wolfe method. *Mathematical Programming*, 155(1):199–230, 2016. ISSN 1436-4646. doi: 10.1007/s10107-014-0841-6. URL http://dx.doi.org/10.1007/s10107-014-0841-6.

Freund, R. M., Grigas, P., and Mazumder, R. An extended Frank–Wolfe method with "in-face" directions, and its application to low-rank matrix completion. *SIAM Journal on Optimization*, 27(1):319–346, 2017.

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

Garber, D. and Meshi, O. Linear-memory and decomposition-invariant linearly convergent conditional gradient algorithm for structured polytopes. *arXiv preprint, arXiv:1605.06492v1*, May 2016.

Garber, D., Sabach, S., and Kaplan, A. Fast generalized conditional gradient method with applications to matrix recovery problems. *arXiv preprint arXiv:1802.05581*, 2018.

Guélat, J. and Marcotte, P. Some comments on wolfe's 'away step'. *Mathematical Programming*, 35(1):110–119, 1986.

Gurobi Optimization. Gurobi optimizer reference manual version 6.5, 2016. URL https://www.gurobi.com/documentation/6.5/refman/.

Gutman, D. H. and Peña, J. F. The condition of a function relative to a polytope. *arXiv preprint arXiv:1802.00271*, February 2018.

Gutman, D. H. and Peña, J. F. The condition of a function relative to a set. *arXiv preprint arXiv:1901.08359*, January 2019.

Holloway, C. A. An extension of the Frank and Wolfe method of feasible directions. *Mathematical Programming*, 6(1):14–27, 1974.

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

Joulin, A., Tang, K., and Fei-Fei, L. Efficient image and video co-localization with frank-wolfe algorithm. In *European Conference on Computer Vision*, pp. 253–268. Springer, 2014.

Kerdreux, T., d'Aspremont, A., and Pokutta, S. Restarting Frank–Wolfe. *arXiv preprint arXiv:1810.02429*, 2018a.

Kerdreux, T., Pedregosa, F., and d'Aspremont, A. Frank–Wolfe with subsampling oracle. *arXiv preprint arXiv:1803.07348*, 2018b.

Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R. E., Danna, E., Gamrath, G., Gleixner, A. M., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D. E., and Wolter, K. MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163, 2011. doi: 10.1007/s12532-011-0025-9. URL http://mpc.zib.de/index.php/MPC/article/view/56/28.

Lacoste-Julien, S. and Jaggi, M. On the global linear convergence of Frank–Wolfe optimization variants. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 28, pp. 496–504. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5925-on-the-global-linear-convergence-of-frank-wolfe-optimization-variants.pdf.

Lan, G., Pokutta, S., Zhou, Y., and Zink, D. Conditional accelerated lazy stochastic gradient descent. *Proceedings of ICML*, 2017.

Lan, G. G. *Lectures on Optimization for Machine Learning*. ISyE, April 2017.

Levitin, E. S. and Polyak, B. T. Constrained minimization methods. *USSR Computational mathematics and mathematical physics*, 6(5):1–50, 1966.

Nemirovski, A. and Yudin, D. *Problem complexity and method efficiency in optimization*. Wiley, 1983. ISBN 0-471-10345-4.

Rao, N., Shah, P., and Wright, S. Forward–backward greedy algorithms for atomic norm regularization. *IEEE Transactions on Signal Processing*, 63(21):5798–5811, 2015.