Lazifying Conditional Gradient Algorithms

Gábor Braun [*1] **Sebastian Pokutta** [*1] **Daniel Zink** [*1]

Abstract

Conditional gradient algorithms (also often called Frank-Wolfe algorithms) are popular due to their simplicity of only requiring a linear optimization oracle and more recently they also gained significant traction for online learning. While simple in principle, in many cases the actual implementation of the linear optimization oracle is costly. We show a general method to *lazify* various conditional gradient algorithms, which in actual computations leads to several orders of magnitude of speedup in wall-clock time. This is achieved by using a faster separation oracle instead of a linear optimization oracle, relying only on *few* linear optimization oracle calls.

1. Introduction

Convex optimization is an important technique both from a theoretical and an applications perspective. Gradient descent based methods are widely used due to their simplicity and easy applicability to many real-world problems. We are interested in solving constraint convex optimization problems of the form

$$\min_{x \in P} f(x), \tag{1}$$

where f is a smooth convex function and P is a polytope, with access to f being limited to first-order information, i.e., we can obtain $\nabla f(v)$ and $f(v)$ for a given $v \in P$ and access to P via a linear minimization oracle which returns $x = \operatorname{argmin}_{v \in P} cx$ for a given linear objective c.

When solving Problem (1) using gradient descent approaches in order to maintain feasibility, typically a projection step is required. This projection back into the feasible region P is potentially computationally expensive, especially for complex feasible regions in very large dimensions. As such projection-free methods gained a lot of attention recently, in particular the Frank-Wolfe algorithm (Frank & Wolfe, 1956) (also known as conditional gradient descent (Levitin & Polyak, 1966); see also (Jaggi, 2013) for an overview) and its online version (Hazan & Kale, 2012) due to their simplicity. We recall the basic Frank-Wolfe algorithm in Algorithm 1. These methods eschew the projection step and rather use a linear optimization oracle to stay within the feasible region. While convergence rates and regret bounds are often suboptimal, in many cases the gain due to only having to solve a *single* linear optimization problem over the feasible region in every iteration still leads to significant computational advantages (see e.g., (Hazan & Kale, 2012, Section 5)). This led to conditional gradients algorithms being used for e.g., online optimization and more generally machine learning and the property that these algorithms naturally generate sparse distributions over the extreme points of the feasible region (sometimes also refereed to as atoms) is often helpful. Further increasing the relevance of these methods, it was shown recently that conditional gradient methods can also achieve linear convergence (see e.g., (Garber & Hazan, 2013; Lacoste-Julien & Jaggi, 2015; Garber & Meshi, 2016)) as well as that the number of total gradient evaluations can be reduced while maintaining the optimal number of oracle calls as shown in (Lan & Zhou, 2014).

Algorithm 1 Frank-Wolfe Algorithm (Frank & Wolfe, 1956)

Input: smooth convex f function with curvature C, $x_1 \in P$ start vertex, LP_P linear minimization oracle
Output: x_t points in P

1: **for** $t = 1$ **to** $T - 1$ **do**
2: $\quad v_t \leftarrow \mathrm{LP}_P(\nabla f(x_t))$
3: $\quad x_{t+1} \leftarrow (1 - \gamma_t)x_t + \gamma_t v_t$ with $\gamma_t := \frac{2}{t+2}$
4: **end for**

Oracle 1 Weak Separation Oracle $\mathrm{LPsep}_P(c, x, \Phi, K)$

Input: $c \in \mathbb{R}^n$ linear objective, $x \in P$ point, $K \geq 1$ accuracy, $\Phi > 0$ objective value;
Output: Either (1) $y \in P$ vertex with $c(x - y) > \Phi/K$, or (2) **false**: $c(x - z) \leq \Phi$ for all $z \in P$.

Unfortunately, for complex feasible regions even solving the linear optimization problem might be time-consuming and as such the cost of solving the LP might be non-negligible.

[1]ISyE, Georgia Institute of Technology, Atlanta, GA. Correspondence to: Daniel Zink <daniel.zink@gatech.edu>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

This could be the case, e.g., when linear optimization over the feasible region is a hard problem or when solving large-scale optimization problems or learning problems. As such it is natural to ask the following questions:

(i) Does the linear optimization oracle have to be called in every iteration?

(ii) Does one need approximately optimal solutions for convergence?

(iii) Can one reuse information across iteration?

We will answer these questions in this work, showing that (i) the LP oracle is not required to be called in every iteration, that (ii) much weaker guarantees are sufficient, and that (iii) we can reuse information. To significantly reduce the cost of oracle calls *while* maintaining identical convergence rates up to small constant factors, we replace the linear optimization oracle by a *(weak) separation oracle* (see Oracle 1) which approximately solves a certain *separation problem* within a multiplicative factor and returns improving vertices (or atoms). We stress that the weak separation oracle is significantly weaker than approximate minimization, which has been already considered in (Jaggi, 2013). In fact, if the oracle returns an improving vertex then this vertex *does not* imply any guarantee in terms of solution quality with respect to the linear minimization problem. It is this relaxation of the dual guarantees that will provide a significant speedup as we will see later. At the same time, in case that the oracle returns *false*, we directly obtain a dual bound via convexity.

A (weak) separation oracle can be realized by a single call to a linear optimization oracle, however with two important differences. It allows for *caching* and *early termination*: Previous solutions are cached, and first it is verified whether any of the cached solutions satisfy the oracle's separation condition. The underlying linear optimization oracle has to be called, only when none of the cached solutions satisfy the condition, and the linear optimization can be stopped as soon as a satisfactory solution with respect to the separation condition is found. See Algorithm 2 for pseudo-code; early termination is implicit in line 4.

We call this technique *lazy optimization* and we will demonstrate significant speedups in wall-clock performance (see e.g., Figure 1), while maintaining identical theoretical convergence rates.

To exemplify our approach we provide conditional gradient algorithms employing the weak separation oracle for the standard Frank-Wolfe algorithm as well as the variants in (Hazan & Kale, 2012; Garber & Meshi, 2016; Garber & Hazan, 2013), which have been chosen due to requiring modified convergence arguments that go beyond those required for the vanilla Frank-Wolfe algorithm. Complementing the theoretical analysis we report computational results demonstrating effectiveness of our approach via a significant reduction in wall-clock running time compared to their linear optimization counterparts.

Oracle 2 $\mathrm{LPsep}_P(c, x, \Phi, K)$ via LP oracle

Input: $c \in \mathbb{R}^n$ linear objective, $x \in P$ point, $K \geq 1$ accuracy, $\Phi > 0$ objective value;
Output: Either (1) $y \in P$ vertex with $c(x-y) > \Phi/K$, or (2) **false**: $c(x-z) \leq \Phi$ for all $z \in P$.

1: **if** $y \in P$ cached with $c(x-y) > \Phi/K$ exists **then**
2: **return** y {Cache call}
3: **else**
4: **compute** $y \leftarrow \operatorname{argmax}_{x \in P} c(x)$ {LP call}
5: **if** $c(x-y) > \Phi/K$ **then**
6: **return** y and **add** y to cache
7: **else**
8: **return false**
9: **end if**
10: **end if**

Related Work

There has been extensive work on Frank-Wolfe algorithms and conditional gradient descent algorithms and we will be only able to review work most closely related to ours. The Frank-Wolfe algorithm was originally introduced in (Frank & Wolfe, 1956) (also known as conditional gradient descent (Levitin & Polyak, 1966) and has been intensely studied in particular in terms of achieving stronger convergence guarantees as well as affine-invariant versions. We demonstrate our approach for the vanilla Frank-Wolfe algorithm (Frank & Wolfe, 1956) (see also (Jaggi, 2013)) as an introductory example. We then consider more complicated variants that require non-trivial changes to the respective convergence proofs to demonstrate the versatility of our approach. This includes the linearly convergent variant via local linear optimization (Garber & Hazan, 2013) as well as the pairwise conditional gradient variant of (Garber & Meshi, 2016), which is especially efficient in terms of implementation. However, our technique also applies to the *Away-Step Frank-Wolfe* algorithm, the *Fully-Corrective Frank-Wolfe* algorithm, as well as the *Block-Coordinate Frank-Wolfe* algorithm. Recently, in (Freund & Grigas, 2016) guarantees for arbitrary step-size rules were provided and an analogous analysis can be also performed for our approach. On the other hand, the analysis of the inexact variants, e.g., with approximate linear minimization does not apply to our case as our oracle is significantly weaker than approximate minimization as pointed out earlier. For more information, we refer the interested reader to the excellent overview in (Jaggi, 2013) for Frank-Wolfe methods in general as well as (Lacoste-Julien & Jaggi, 2015) for an overview with respect to global linear convergence.

It was also recently shown in (Hazan & Kale, 2012) that the Frank-Wolfe algorithm can be adjusted to the online learning setting and here we provide a lazy version of this algorithm. Combinatorial convex online optimization has been investigated in a long line of work (see e.g., (Kalai & Vempala, 2005; Audibert et al., 2013; Neu & Bartók, 2013)). It is important to note that our regret bounds hold in the structured online learning setting, i.e., our bounds depend on the ℓ_1-diameter or sparsity of the polytope, rather than its ambient dimension for arbitrary convex functions (see e.g., (Cohen & Hazan, 2015; Gupta et al., 2016)). We refer the interested reader to (Hazan, 2016) for an extensive overview.

A key component of the new oracle is the ability to cache and reuse old solutions, which accounts for the majority of the observed speed up. The idea of caching of oracle calls was already explored in various other contexts such as cutting plane methods (see e.g., (Joachims et al., 2009)) as well as the *Block-Coordinate Frank-Wolfe* algorithm in (Shah et al., 2015; Osokin et al., 2016). Our lazification approach (which uses caching) is different however in the sense that our weak separation oracle does not resemble an approximate linear optimization oracle with a multiplicative approximation guarantee; see (Osokin et al., 2016, Proof of Theorem 3. Appendix F) and (Lacoste-Julien et al., 2013) for comparison to our setup. In fact, our weaker oracle does not imply any approximation guarantee and differs from approximate minimization as done e.g., in (Jaggi, 2013) substantially.

Contribution

The main technical contribution of this paper is a new approach, whereby instead of finding the optimal solution, the oracle is used only to find a *good enough solution* or a *certificate* that such a solution does not exist, both ensuring the desired convergence rate of the conditional gradient algorithms.

Our contribution can be summarized as follows:

(i) *Lazifying approach.* We provide a general method to lazify conditional gradient algorithms. For this we replace the linear optimization oracle with a weak separation oracle, which allows us to reuse feasible solutions from previous oracle calls, so that in many cases the oracle call can be skipped. In fact, once a simple representation of the underlying feasible region is learned no further oracle calls are needed. We also demonstrate how parameter-free variants can be obtained.

(ii) *Lazified conditional gradient algorithms.* We exemplify our approach by providing lazy versions of the vanilla Frank-Wolfe algorithm as well as of the conditional gradient methods in (Hazan & Kale, 2012; Garber & Hazan, 2013; Garber & Meshi, 2016).

(iii) *Weak separation through augmentation.* We show in the case of 0/1 polytopes how to implement a weak separation oracle with at most k calls to an augmentation oracle that on input $c \in \mathbb{R}^n$ and $x \in P$ provides either an improving solution $\overline{x} \in P$ with $c\overline{x} < cx$ or ensures optimality, where k denotes the ℓ_1-diameter of P. This is useful when the solution space is sparse.

(iv) *Computational experiments.* We demonstrate computational superiority by extensive comparisons of the weak separation based versions with their original versions. In all cases we report significant speedups in wall-clock time often of several orders of magnitude.

It is important to note that in all cases, we inherit the same requirements, assumptions, and properties of the baseline algorithm that we lazify. This includes applicable function classes, norm requirements, as well as smoothness and (strong) convexity requirements. We also maintain identical convergence rates up to (small!) constant factors.

Outline

We briefly recall notation and notions in Section 2 and consider conditional gradients algorithms in Section 3. In Section 4 we explain how parameter-free variants of the proposed algorithms can be obtained. Finally, in Section 5 we provide some experimental results. In the supplemental material we consider two more variants of conditional gradients algorithms (Sections B and C), we show that we can realize a weak separation oracle with an even weaker oracle in the case of combinatorial problem (Section D) and we provide additional computational results (Section E).

2. Preliminaries

Let $\|\cdot\|$ be an arbitrary norm on $\mathbb{R}^n$, and let $\|\cdot\|^*$ denote the dual norm of $\|\cdot\|$. We will specify the applicable norm in the later sections. A function f is *L-Lipschitz* if $|f(y) - f(x)| \leq L\|y - x\|$ for all $x, y \in \operatorname{dom} f$. A convex function f is *smooth* with *curvature* at most C if $f(\gamma y + (1-\gamma)x) \leq f(x) + \gamma \nabla f(x)(y - x) + C\gamma^2/2$ for all $x, y \in \operatorname{dom} f$ and $0 \leq \gamma \leq 1$. A function f is *S-strongly convex* if $f(y) - f(x) \geq \nabla f(x)(y - x) + \frac{S}{2}\|y - x\|^2$ for all $x, y \in \operatorname{dom} f$. Unless stated otherwise Lipschitz continuity and strong convexity will be measured in the norm $\|\cdot\|$. Moreover, let $\mathbb{B}_r(x) \coloneqq \{y \mid \|x - y\| \leq r\}$ be the ball around x with radius r with respect to $\|.\|$. In the following, P will denote the feasible region, a polytope and the vertices of P will be denoted by $v_1, \ldots, v_N$.

3. Lazy Conditional Gradients

We start with the most basic Frank-Wolfe algorithm as a simple example how a conditional gradient algorithm can be lazified by means of a *weak separation oracle*. We will also use the basic variant to discuss various properties and implications. We then show how the more complex Frank-Wolfe algorithms in (Garber & Hazan, 2013) and (Garber & Meshi, 2016) can be lazified. Throughout this section $\|\cdot\|$ denotes the ℓ_2-norm.

3.1. Lazy Conditional Gradients: a basic example

We start with lazifying the original Frank-Wolfe algorithm (arguably the simplest Conditional Gradients algorithm), adapting the baseline argument from (Jaggi, 2013, Theorem 1). While the vanilla version has suboptimal convergence rate $O(1/T)$, its simplicity makes it an illustrative example of the main idea of lazification. The lazy algorithm (Algorithm 2) maintains an upper bound Φ_t on the convergence rate, guiding its eagerness for progress when searching for an improving vertex v_t. If the oracle provides an improving vertex v_t we refer to this as a *positive call* and we call it a *negative call* otherwise.

Algorithm 2 Lazy Conditional Gradients (LCG)

Input: smooth convex f function with curvature C, $x_1 \in P$ start vertex, LPsep_P weak linear separation oracle, accuracy $K > 1$, initial upper bound Φ_0
Output: x_t points in P

1: **for** $t = 1$ **to** $T - 1$ **do**
2: $\quad\Phi_t \leftarrow \frac{\Phi_{t-1} + \frac{C\gamma_t^2}{2}}{1+\frac{\gamma_t}{K}}$
3: $\quad v_t \leftarrow \text{LPsep}_P(\nabla f(x_t), x_t, \Phi_t, K)$
4: $\quad$**if** $v_t =$ **false then**
5: $\quad\quad x_{t+1} \leftarrow x_t$
6: $\quad$**else**
7: $\quad\quad x_{t+1} \leftarrow (1-\gamma_t)x_t + \gamma_t v_t$
8: $\quad$**end if**
9: **end for**

The step size γ_t is chosen to (approximately) minimize Φ_t in Line 2; roughly Φ_{t-1}/KC.

Theorem 3.1. *Assume f is convex and smooth with curvature C. Then Algorithm 2 with $\gamma_t = \frac{2(K^2+1)}{K(t+K^2+2)}$ has convergence rate*

$$f(x_t) - f(x^*) \leq \frac{2\max\{C, \Phi_0\}(K^2+1)}{t+K^2+2},$$

where x^* is a minimum point of f over P.

Proof. We prove by induction that $f(x_t) - f(x^*) \leq \Phi_{t-1}$. The claim is clear for $t = 1$ by the choice of Φ_0. Assuming the claim is true for t, we prove it for $t+1$. We distinguish two cases depending on the return value of the weak separation oracle in Line 3.

When the oracle returns an improving solution v_t, which we call the positive case, then $\nabla f(x_t)(x_t - v_t) \geq \Phi_t/K$, which is used in the second inequality below. The first inequality follows by smoothness of f, and the third inequality by the induction hypothesis:

$$\begin{aligned} &f(x_{t+1}) - f(x^*) \\ &\quad\leq f(x_t) - f(x^*) + \gamma_t \nabla f(x_t)(v_t - x_t) + \frac{C\gamma_t^2}{2} \\ &\quad\leq f(x_t) - f(x^*) - \gamma_t \frac{\Phi_t}{K} + \frac{C\gamma_t^2}{2} \\ &\quad\leq \Phi_{t-1} - \gamma_t \frac{\Phi_t}{K} + \frac{C\gamma_t^2}{2} = \Phi_t, \end{aligned}$$

When the oracle returns no improving solution, then in particular $\nabla f(x_t)(x_t - x^*) \leq \Phi_t$, hence by Line 5 $f(x_{t+1}) - f(x^*) = f(x_t) - f(x^*) \leq \nabla f(x_t)(x_t - x^*) = \Phi_t$.

Finally, using the specific values of γ_t we prove the upper bound

$$\Phi_{t-1} \leq \frac{2\max\{C, \Phi_0\}(K^2+1)}{t+K^2+2}$$

by induction on t. The claim is obvious for $t = 1$. The induction step is an easy computation relying on the definition of Φ_t on Line 2:

$$\begin{aligned} \Phi_t &= \frac{\Phi_{t-1} + \frac{C\gamma_t^2}{2}}{1+\frac{\gamma_t}{K}} \leq \frac{\frac{2\max\{C,\Phi_0\}(K^2+1)}{t+K^2+2} + \frac{\max\{C,\Phi_0\}\gamma_t^2}{2}}{1+\frac{\gamma_t}{K}} \\ &= 2\max\{C,\Phi_0\}(K^2+1)\frac{1+\frac{\gamma_t}{2K}}{\left(1+\frac{\gamma_t}{K}\right)(t+K^2+2)} \\ &\leq \frac{2\max\{C,\Phi_0\}(K^2+1)}{t+K^2+3}. \end{aligned}$$

Here the second equation follows via plugging-in the choice for γ_t for one of the γ_t in the quadratic term and last inequality follows from $t \geq 1$ and the concrete choice of γ_t. □

Remark 3.2 (Discussion of the weak separation oracle). A few remarks are in order:

(i) *Interpretation of weak separation oracle.* The weak separation oracle provides new *extreme points* (or vertices) v_t that ensure necessary progress to converge at the proposed rate Φ_t or it certifies that we are already Φ_t-close to the optimal solution. It is important to note that the two cases in Oracle 1 are not mutually exclusive: the oracle might return $y \in P$ with $c(x - y) > \Phi/K$ (positive call: returning a vertex y with improvement Φ/K), while still $c(x - z) \leq \Phi$ for all $z \in P$ (negative call: certifying that there is no vertex

z that can improve by Φ). This a desirable property as it makes the separation problem much easier and the algorithm works with either answer in the ambiguous case.

(ii) *Choice of K.* The K parameter can be used to bias the oracle towards positive calls, i.e., returning improving directions. We would also like to point out that the algorithm above as well as those below will also work for $K = 1$, however we show in supplemental material (Section D) that we can use an even weaker oracle to realize a weak separation oracle if $K > 1$ and for consistency, we require $K > 1$ throughout. In the case $K = 1$ the two cases in the oracle are mutually exclusive.

(iii) *Effect of caching and early termination.* When realizing the weak separation oracle, the actual linear optimization oracle has to be only called if none of the previously seen vertices (or atoms) satisfies the separation condition. Moreover, the weak separation oracle has to only produce a satisfactory solution and not an approximately optimal one. These two properties are responsible for the observed speedup (see Figure 1). Moreover, the convex combinations of vertices of P that represent the solutions x_t are extremely sparse as we reuse (cached) vertices whenever possible.

(iv) *Dual certificates.* By not computing an approximately optimal solution, we give up dual optimality certificates. For a given point $x \in P$, let $g(x) \coloneqq \max_{v \in P} \nabla f(x)(x - v)$ denote the *Wolfe gap*. We have $f(x) - f(x^*) \leq g(x)$ where $x^* = \operatorname{argmin}_{x \in P} f(x)$ by convexity. In those rounds t where we obtain an improving vertex we have no information about $g(x_t)$. However, if the oracle returns *false* in round t, then we obtain the dual certificate $f(x_t) - f(x^*) \leq g(x_t) \leq \Phi_t$.

(v) *Rate of convergence.* A close inspection of the algorithm utilizing the weak separation oracle suggests that the algorithm converges only at the worst-case convergence rate that we propose with the Φ_t sequence. This however is only an artefact of the simplified presentation for the proof of the worst-case rate. We can easily adjust the algorithm to implicitly perform a search over the rate Φ_t combined with line search for γ. This leads to a parameter-free variant of Algorithm 2as given in Section 4 and comes at the expense of a (small!) constant factor deterioration of the worst-case rate guarantee; see also Supplementary Material A.(iii) for an in-depth discussion.

We discuss potential implementation improvements in Supplementary Material A.

3.2. Lazy Pairwise Conditional Gradients

In this section we provide a lazy variant (Algorithm 3) of the Pairwise Conditional Gradient algorithm from (Garber & Meshi, 2016), using separation instead of linear optimization. We make identical assumptions: the feasible region is a $0/1$ polytope given in the form $P = \{x \in \mathbb{R}^n \mid 0 \leq x \leq \mathbf{1}, Ax = b\}$, where $\mathbf{1}$ denotes the all-one vector of compatible dimension; in particular all vertices of P have only $0/1$ entries.

Algorithm 3 Lazy Pairwise Conditional Gradients (LPCG)

Input: polytope P, smooth and S-strongly convex function f with curvature C, accuracy $K > 1$, η_t non-increasing step-sizes
Output: x_t points

1: $x_1 \in P$ arbitrary and $\Phi_0 \geq f(x_1) - f(x^*)$
2: **for** $t = 1, \ldots, T$ **do**
3: define $\tilde{\nabla} f(x_t) \in \mathbb{R}^m$ as follows:

$$\tilde{\nabla} f(x_t)_i \coloneqq \begin{cases} \nabla f(x_t)_i & \text{if } (x_t)_i > 0 \\ -\infty & \text{if } (x_t)_i = 0 \end{cases}$$

4: $\Phi_t \leftarrow \frac{2\Phi_{t-1} + \eta_t^2 C}{2 + \frac{\eta_t}{K \Delta_t}}$
5: $c_t \leftarrow \left(\nabla f(x_t), -\tilde{\nabla} f(x_t)\right)$
6: $(v_t^+, v_t^-) \leftarrow \mathrm{LPsep}_{P \times P}\left(c_t, (x_t, x_t), \frac{\Phi_t}{\Delta_t}, K\right)$
7: **if** $(v_t^+, v_t^-) =$ **false then**
8: $x_{t+1} \leftarrow x_t$
9: **else**
10: $\tilde{\eta}_t \leftarrow \max\{2^{-\delta} \mid \delta \in \mathbb{Z}_{\geq 0},\ 2^{-\delta} \leq \eta_t\}$
11: $x_{t+1} \leftarrow x_t + \tilde{\eta}_t (v_t^+ - v_t^-)$
12: **end if**
13: **end for**

Observe that Algorithm 3 calls LPsep on the cartesian product of P with itself. Choosing the objective function as in Line 5 allows us to simultaneously find an improving direction and an away-step direction.

Theorem 3.3. *Let x^* be a minimum point of f in P, and Φ_0 an upper bound of $f(x_1) - f(x^*)$. Furthermore, let $M_1 \coloneqq \sqrt{\frac{S}{8 \operatorname{card}(x^*)}}$, $M_2 \coloneqq KC/2$, $\kappa \coloneqq \min\{\frac{M_1}{2M_2}, 1/\sqrt{\Phi_0}\}$, $\eta_t \coloneqq \kappa\sqrt{\Phi_{t-1}}$ and $\Delta_t \coloneqq \sqrt{\frac{2 \operatorname{card}(x^*)\Phi_{t-1}}{S}}$, then Algorithm 3 has convergence rate*

$$f(x_{t+1}) - f(x^*) \leq \Phi_t \leq \Phi_0 \left(\frac{1 + B}{1 + 2B}\right)^t,$$

where $B \coloneqq \kappa \cdot \frac{M_1}{2K}$.

We recall a technical lemma for the proof.

Lemma 3.4 ((Garber & Meshi, 2016, Lemma 2))**.** *Let $x, y \in P$. There exists vertices v_i of P such that $x = \sum_{i=1}^k \lambda_i v_i$ and $y = \sum_{i=1}^k (\lambda_i - \gamma_i) v_i + \left(\sum_{i=1}^k \gamma_i\right) z$ with $\gamma_i \in [0, \lambda_i]$, $z \in P$ and $\sum_{i=1}^k \gamma_i \leq \sqrt{\operatorname{card}(y)} \|x - y\|$.*

Proof of Theorem 3.3. The feasibility of the iterates x_t is ensured by Line 10 and the monotonicity of the sequence $\{\eta_t\}_{t\geq 1}$ with the same argument as in (Garber & Meshi, 2016, Lemma 1 and Observation 2).

We first show by induction that $f(x_{t+1}) - f(x^*) \leq \Phi_t$. For $t = 0$ we have $\Phi_0 \geq f(x_1) - f(x^*)$. Now assume the statement for some $t \geq 0$. In the negative case (Line 8), we use the guarantee of Oracle 1 to get $c_t((x_t, x_t)-(z_1, z_2)) \leq \frac{\Phi_t}{\Delta_t}$ for all $z_1, z_2 \in P$, which is equivalent to (as $c_t(x_t, x_t) = 0$) $\tilde{\nabla} f(x_t)z_2 - \nabla f(x_t)z_1 \leq \frac{\Phi_t}{\Delta_t}$ and therefore

$$\nabla f(x_t)(\tilde{z}_2 - z_1) \leq \frac{\Phi_t}{\Delta_t},$$

for all $\tilde{z}_2, z_1 \in P$ with $\operatorname{supp}(\tilde{z}_2) \subseteq \operatorname{supp}(x_t)$. We further use Lemma 3.4 to write $x_t = \sum_{i=1}^k \lambda_i v_i$ and $x^* = \sum_{i=1}^k (\lambda_i - \gamma_i)v_i + \sum_{i=1}^k \gamma_i z$ with $\gamma_i \in [0, \lambda_i]$, $z \in P$ and $\sum_{i=1}^k \gamma_i \leq \sqrt{\operatorname{card}(x^*)}\|x_t - x^*\| \leq \sqrt{\frac{2\operatorname{card}(x^*)\Phi_{t-1}}{S}} = \Delta_t$, using the induction hypothesis and the strong convexity in the second inequality. Then $f(x_{t+1}) - f(x^*) = f(x_t) - f(x^*) \leq \nabla f(x_t)(x_t - x^*) = \sum_{i=1}^k \gamma_i (v_i - z) \cdot \nabla f(x_t) \leq \Phi_t$, where we used Equation 3.2 for the last inequality.

For the positive case (Lines 10 and 11) we get, using first smoothness of f, then $\eta_t/2 < \tilde{\eta}_t \leq \eta_t$ and $\nabla f(x_t)(v_t^+ - v_t^-) \leq -\Phi_t/(\Delta_t K)$, and finally the definition of Φ_t:

$$\begin{aligned} f(x_{t+1}) - f(x^*) &= f(x_t + \tilde{\eta}_t(v_t^+ - v_t^-)) - f(x^*) \\ &\leq \Phi_{t-1} + \tilde{\eta}_t \nabla f(x_t)(v_t^+ - v_t^-) + \frac{\tilde{\eta}_t^2 C}{2} \\ &\leq \Phi_{t-1} - \frac{\eta_t}{2} \cdot \frac{\Phi_t}{\Delta_t K} + \frac{\eta_t^2 C}{2} = \Phi_t. \end{aligned}$$

Plugging in the values of η_t and Δ_t to the definition of Φ_t gives the desired bound.

$$\begin{aligned} \Phi_t &= \frac{2\Phi_{t-1} + \eta_t^2 C}{2 + \frac{\eta_t}{K\Delta_t}} = \Phi_{t-1} \frac{1 + \kappa^2 M_2/K}{1 + \kappa M_1/K} \\ &\leq \Phi_{t-1} \frac{1+B}{1+2B} \leq \Phi_0 \left(\frac{1+B}{1+2B}\right)^t. \end{aligned}$$ □

4. Parameter-free Conditional Gradients via Weak Separation

We now provide a parameter-free variant of the Lazy Frank-Wolfe Algorithm. We stress that the worst-case convergence rate is identical up to a small constant factor. Here we find a tight initial bound Φ_0 with a single extra LP call, which can be also done approximately as long as Φ_0 is a valid upper bound. Alternatively, one can perform binary search via the weak separation oracle as described earlier.

Note that the accuracy parameter K in Algorithm 4 is a parameter of the oracle and not of the algorithm itself. We will show now that Algorithm 4 converges in the worst-case at a rate identical to Algorithm 2 (up to a small constant factor).

Algorithm 4 Parameter-free Lazy Conditional Gradients (LCG)

Input: smooth convex function f, $x_1 \in P$ start vertex, LPsep_P weak linear separation oracle, accuracy $K > 1$
Output: x_t points in P

1: $\Phi_0 \leftarrow \max_{x\in P} \nabla f(x_1)(x_1 - x)/2$ {Initial bound}
2: **for** $t = 1$ **to** $T - 1$ **do**
3: $v_t \leftarrow \mathrm{LPsep}_P(\nabla f(x_t), x_t, \Phi_{t-1}, K)$
4: **if** $v_t =$ **false then**
5: $x_{t+1} \leftarrow x_t$
6: $\Phi_t \leftarrow \frac{\Phi_{t-1}}{2}$ {Update Φ}
7: **else**
8: $\gamma_t \leftarrow \operatorname{argmin}_{0\leq\gamma\leq 1} f((1-\gamma)x_t + \gamma v_t)$
9: $x_{t+1} \leftarrow (1-\gamma_t)x_t + \gamma_t v_t$ {Update iterate}
10: $\Phi_t \leftarrow \Phi_{t-1}$
11: **end if**
12: **end for**

Theorem 4.1. *Let f be a smooth convex function with curvature C. Algorithm 4 converges at a rate proportional to $1/t$. In particular to achieve a bound $f(x_t) - f(x^*) \leq \varepsilon$, given an initial upper bound $f(x_1) - f(x^*) \leq 2\Phi_0$, the number of required steps is upper bounded by*

$$t \leq \lceil \log \Phi_0/\varepsilon \rceil + 1 + 4K\lceil \log \Phi_0/KC \rceil + \frac{16K^2C}{\varepsilon}.$$

Proof. The main idea of the proof is that while negative answers to oracle calls halve the dual upper bound $2\Phi_t$, positive oracle calls significantly decrease the function value of the current point.

We analyze iteration t of the algorithm. If Oracle 1 in Line 3 returns a negative answer (i.e., **false**, case (2)), then this guarantees $\nabla f(x_t)(x_t - x) \leq \Phi_{t-1}$ for all $x \in P$, in particular, using convexity, $f(x_{t+1}) - f(x^*) = f(x_t) - f(x^*) \leq \nabla f(x_t)(x_t - x^*) \leq \Phi_{t-1} = 2\Phi_t$.

If Oracle 1 returns a positive answer (case (1)), then we have $f(x_t) - f(x_{t+1}) \geq \gamma_t \Phi_{t-1}/K - (C/2)\gamma_t^2$ by smoothness of f. By minimality of γ_t, therefore $f(x_t) - f(x_{t+1}) \geq \min_{0\leq\gamma\leq 1}(\gamma\Phi_{t-1}/K - (C/2)\gamma^2)$, which is $\Phi_{t-1}^2/(2CK^2)$ if $\Phi_{t-1} < KC$, and $\Phi_{t-1}/K - C/2 \geq \frac{C}{2}$ if $\Phi_{t-1} \geq KC$.

Now we bound the number t' of consecutive positive oracle calls immediately following an iteration t with a negative oracle call. Note that the same argument bounds the number of initial consecutive positive oracle calls with the choice $t = 0$, as we only use $f(x_{t+1}) - f(x^*) \leq 2\Phi_t$ below.

Note that $\Phi_t = \Phi_{t+1} = \cdots = \Phi_{t+t'}$. Therefore

$$2\Phi_t \geq f(x_{t+1}) - f(x^*) \geq \sum_{\tau=t+1}^{t+t'} (f(x_\tau) - f(x_{\tau+1}))$$
$$\geq \begin{cases} t' \frac{\Phi_t^2}{2CK^2} & \text{if } \Phi_{t-1} < KC \\ t' \left(\frac{\Phi_{t-1}}{K} - \frac{C}{2} \right) & \text{if } \Phi_{t-1} \geq KC \end{cases},$$

which gives in the case $\Phi_t < KC$ that $t' \leq 4CK^2/\Phi_t$, and in the case $\Phi_t \geq KC$ that

$$t' \leq \frac{2\Phi_t}{\frac{\Phi_t}{K} - \frac{C}{2}} = \frac{4K\Phi_t}{2\Phi_t - KC} \leq \frac{4K\Phi_t}{2\Phi_t - \Phi_t} = 4K.$$

Thus iteration t is followed by at most $4K$ consecutive positive oracle calls as long as $\Phi_t \geq KC$, and $4CK^2/\Phi_t < 2^{\ell+1} \cdot 4K$ ones for $2^{-\ell-1}KC < \Phi_t \leq 2^{-\ell}KC$ with $\ell \geq 0$.

Adding up the number of oracle calls gives the desired rate: in addition to the positive oracle calls we also have at most $\lceil \log(\Phi_0/\varepsilon) \rceil + 1$ negative oracle calls, where $\log(\cdot)$ is the binary logarithm and ε is the (additive) accuracy. Thus after a total of

$$\lceil \log \Phi_0/\varepsilon \rceil + 1 + 4K \lceil \log \Phi_0/KC \rceil + \sum_{\ell=0}^{\lceil \log KC/\varepsilon \rceil} 2^{\ell+1} \cdot 4K$$
$$\leq \lceil \log \Phi_0/\varepsilon \rceil + 1 + 4K \lceil \log \Phi_0/KC \rceil + \frac{16K^2C}{\varepsilon}$$

iterations (or equivalently oracle calls) we have $f(x_t) - f(x^*) \leq \varepsilon$. □

Remark 4.2. Observe that Algorithm 4 might converge much faster due to the aggressive halving of the rate. In fact, Algorithm 4 convergences at a rate that is at most a factor $4K^2$ slower than the rate that the vanilla (non-lazy) Frank-Wolfe algorithm would realize for the same problem. In actual wall-clock time Algorithm 4 is much faster though due to the use of the weaker oracle; see Figure 2 and 4 for a comparison and Section E.1.2 for more experimental results.

Negative oracle calls tend to be significantly more expensive time-wise than positive oracle calls due to proving dual bounds. The following corollary is an immediate consequence of the argumentation from above:

Corollary 4.3. *Algorithm 4 makes at most* $\lceil \log \Phi_0/\varepsilon \rceil + 1$ *negative oracle calls.*

If line search is too expensive we can choose $\gamma_t = \min(1, \Phi_t/KC)$ in Algorithm 4. In this case an estimate of the curvature C is required, though no explicit knowledge of the sequence Φ_t is needed as compared to the textbook variant in Section 3.1.

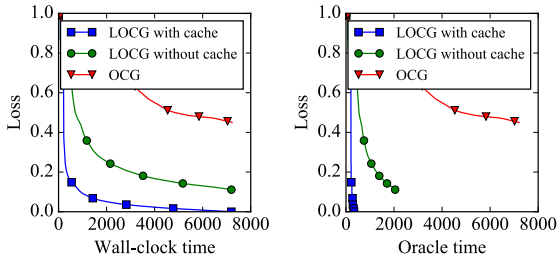

Figure 1. Performance gain due to caching and early termination for stochastic optimization over a maximum cut problem with linear losses. The red line is the OCG baseline, the green one is the lazy variant using only early termination, and the blue one uses caching and early termination. Left: loss vs. wall-clock time. Right: loss vs. total time spent in oracle calls. Time limit was 7200 seconds. Caching allows for a significant improvement in loss reduction in wall-clock time. The effect is even more obvious in oracle time as caching cuts out a large number of oracle calls.

5. Experiments

As mentioned before, lazy algorithms have two improvements: caching and early termination. Here we depict the effect of caching in Figure 1, comparing OCG (no caching, no early termination), LOCG (caching and early termination) and LOCG (only early termination) (see Algorithm 7). We did not include a caching-only OCG variant, because caching without early termination does not make much sense: in each iteration a new linear optimization problem has to be solved; previous solutions can hardly be reused as they are unlikely to be optimal for the new linear optimization problem.

5.1. Effect of K

If the parameter K of the oracle can be chosen, which depends on the actual oracle implementation, then we can increase K to bias the algorithm towards performing more positive calls. At the same time the steps get shorter. As such there is a natural trade-off between the cost of many positive calls vs. a negative call. We depict the impact of the parameter choice for K in Figure 6.

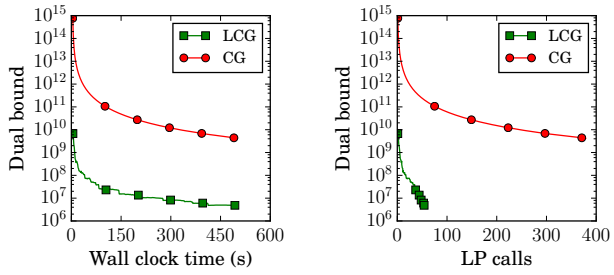

Figure 2. Performance on an instance of the video colocalization problem. We solve quadratic minimization over a flow polytope and report the achieved dual bound (or Wolfe-gap) over wall-clock time in seconds in logscale on the left and over the number of actual LP calls on the right. We used the parameter-free variant of the Lazy CG algorithm, which performs in both measures significantly better than the non-lazy counterpart. The performance difference is more prominent in the number of LP calls.

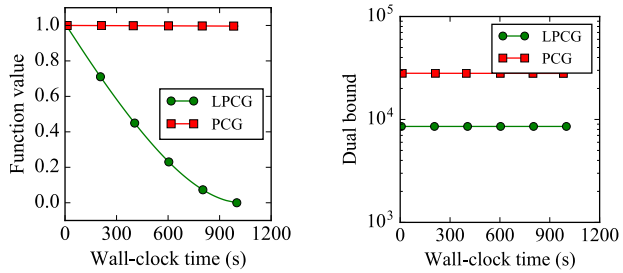

Figure 3. Performance on a large instance of the video colocalization problem using PCG and its lazy variant. We observe that lazy PCG is significantly better both in terms of function value and dual bound. Recall that the function value is normalized between $[0, 1]$.

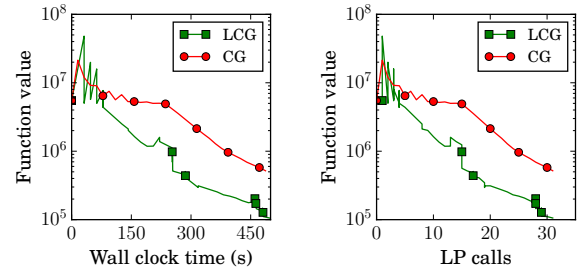

Figure 4. Performance on a matrix completion instance. More information about this problem can be found in the supplemental material (Section E). The performance is reported as the objective function value over wall-clock time in seconds on the left and over LP calls on the right. In both measures after an initial phase the function value using LCG is much lower than with the non-lazy algorithm.

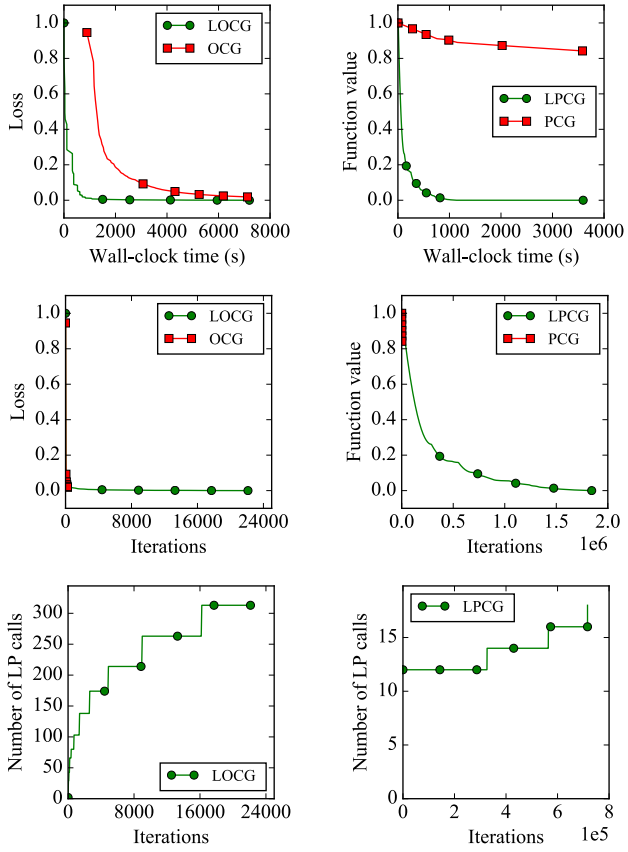

Figure 5. Performance of the two lazified variants LOCG (left column) and LPCG (right column). The feasible regions are a cut polytope on the left and the MIPLIB instance `air04` on the right. The objective functions are in both cases quadratic, on the left randomly chosen in every step. We show the performance over wall clock time in seconds (first row) and over iterations (second row). The last row shows the number of call to the linear optimization oracle. The lazified versions perform significantly better in wall clock time compared to the non-lazy counterparts.

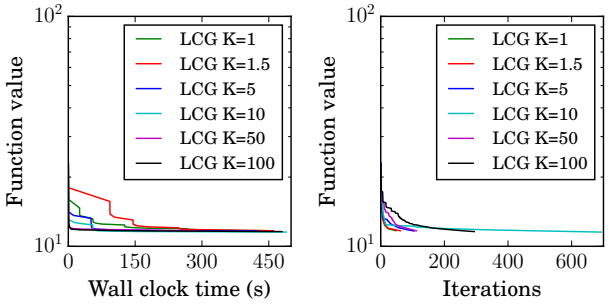

Figure 6. Impact of the oracle approximation parameter K depicted for the Lazy CG algorithm. We can see that increasing K leads to a deterioration of progress in iterations but improves performance in wall-clock time. The behavior is similar for other algorithms.

Acknowledgements

We are indebted to Alexandre D'Aspremont, Simon Lacoste-Julien, and George Lan for the helpful discussions and for providing us with relevant references. Research reported in this paper was partially supported by NSF CAREER award CMMI-1452463.

References

Achterberg, Tobias, Koch, Thorsten, and Martin, Alexander. MIPLIB 2003. *Operations Research Letters*, 34(4):361–372, 2006. doi: 10.1016/j.orl.2005.07.009. URL http://www.zib.de/Publications/abstracts/ZR-05-28/.

Audibert, Jean-Yves, Bubeck, Sébastien, and Lugosi, Gábor. Regret in online combinatorial optimization. *Mathematics of Operations Research*, 39(1):31–45, 2013.

Bodic, Pierre Le, Pavelka, Jeffrey W, Pfetsch, Marc E, and Pokutta, Sebastian. Solving MIPs via scaling-based augmentation. *arXiv preprint arXiv:1509.03206*, 2015.

Cohen, Alon and Hazan, Tamir. Following the perturbed leader for online structured learning. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1034–1042, 2015.

Dash, Sanjeeb. A note on QUBO instances defined on Chimera graphs. *preprint arXiv:1306.1202*, 2013.

Frank, András and Tardos, Éva. An application of simultaneous Diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987.

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

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

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

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

Grötschel, Martin and Lovász, László. Combinatorial optimization: A survey, 1993.

Gupta, Swati, Goemans, Michel, and Jaillet, Patrick. Solving combinatorial games using products, projections and lexicographically optimal bases. *arXiv preprint arXiv:1603.00522*, 2016.

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

Hazan, Elad. Introduction to online convex optimization. *Foundations and Trends in Optimization*, 2(3–4):157–325, 2016. doi: 10.1561/2400000013. URL http://ocobook.cs.princeton.edu/.

Hazan, Elad and Kale, Satyen. Projection-free online learning. *arXiv preprint arXiv:1206.4657*, 2012.

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

Joachims, Thorsten, Finley, Thomas, and Yu, Chun-Nam John. Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59, 2009.

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

Kalai, Adam and Vempala, Santosh. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3):291–307, 2005.

Koch, Thorsten, Achterberg, Tobias, Andersen, Erling, Bastert, Oliver, Berthold, Timo, Bixby, Robert E., Danna, Emilie, Gamrath, Gerald, Gleixner, Ambros M., Heinz, Stefan, Lodi, Andrea, Mittelmann, Hans, Ralphs, Ted, Salvagnin, Domenico, Steffy, Daniel E., and Wolter, Kati. 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, Simon and Jaggi, Martin. 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.

Lacoste-Julien, Simon, Jaggi, Martin, Schmidt, Mark, and Pletscher, Patrick. Block-coordinate frank-wolfe optimization for structural svms. In *ICML 2013 International Conference on Machine Learning*, pp. 53–61, 2013.

Lan, Guanghui and Zhou, Yi. Conditional gradient sliding for convex optimization. *Optimization-Online preprint (4605)*, 2014.

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

Neu, Gergely and Bartók, Gábor. An efficient algorithm for learning with semi-bandit feedback. In *Algorithmic Learning Theory*, pp. 234–248. Springer, 2013.

Oertel, Timm, Wagner, Christian, and Weismantel, Robert. Integer convex minimization by mixed integer linear optimization. *Oper. Res. Lett.*, 42(6-7):424–428, 2014.

Osokin, Anton, Alayrac, Jean-Baptiste, Lukasewitz, Isabella, Dokania, Puneet K, and Lacoste-Julien, Simon. Minding the gaps for block frank-wolfe optimization of structured svms. *ICML 2016 International Conference on Machine Learning / arXiv preprint arXiv:1605.09346*, 2016.

Schulz, Andreas S and Weismantel, Robert. The complexity of generic primal algorithms for solving general integer programs. *Mathematics of Operations Research*, 27(4): 681–692, 2002.

Schulz, Andreas S., Weismantel, Robert, and Ziegler, Günter M. 0/1-integer programming: Optimization and augmentation are equivalent. In *Algorithms – ESA '95, Proceedings*, pp. 473–483, 1995.

Shah, Neel, Kolmogorov, Vladimir, and Lampert, Christoph H. A multi-plane block-coordinate frank-wolfe algorithm for training structural svms with a costly max-oracle. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2737–2745, 2015.