Journal of Machine Learning Research 20 (2019) 1-42 Submitted 3/18; Revised 2/19; Published 3/19

Lazifying Conditional Gradient Algorithms

Gábor Braun GABOR.BRAUN@ISYE.GATECH.EDU
Sebastian Pokutta SEBASTIAN.POKUTTA@ISYE.GATECH.EDU
Daniel Zink DANIEL.ZINK@GATECH.EDU
School of Industrial & Systems Engineering
Georgia Institute of Technology
Atlanta, GA 30332, USA

Editor: Mark Schmidt

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.

Keywords: Frank–Wolfe algorithm, conditional gradient, caching, linear optimization oracle, convex optimization

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 $\nabla f(x)$ and $f(x)$ for a given $x \in P$ and access to P via a linear minimization oracle, which returns $\mathrm{LP}_P(c) = \operatorname{argmin}_{x \in P} cx$ 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 and Wolfe (1956) (also known as conditional gradient descent Levitin and Polyak (1966); see also Jaggi (2013) for an overview) and its online version Hazan and 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

Algorithm 1 Frank–Wolfe Algorithm Frank and Wolfe (1956)

Input: smooth convex function f with curvature C, start vertex $x_1 \in P$, linear minimization oracle LP_P
Output: points x_t 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 $\gamma_t \coloneqq \frac{2}{t+2}$
4: **end for**

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 and Kale, 2012, Section 5)). This led to conditional gradient algorithms being used for e.g., online optimization and more generally machine learning. Also the property that these algorithms naturally generate sparse distributions over the extreme points of the feasible region 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 and Hazan (2013); Lacoste-Julien and Jaggi (2015); Garber and 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 and Zhou (2014).

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. This could be the case, e.g., when linear optimization over the feasible region is a hard problem or when solving large-scale optimization 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 iterations?

We will answer these questions in this work, showing that (i) the LP oracle is not required to be called in every iteration, (ii) much weaker guarantees are sufficient, and (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* (Oracle 1) which approximately solves a *separation problem*

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

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

within a multiplicative factor and returns improving vertices. We stress that the weak separation oracle is significantly weaker than approximate minimization, which has been

already considered in Jaggi (2013). In fact, there is no guarantee that the improving vertices returned by the oracle are near to the optimal solution to the linear minimization problem. It is this relaxation of dual bounds and approximate optimality that will provide a significant speedup as we will see later. However, if the oracle does not return an improving vertex (returns **false**), then this fact can be used to derive a reasonably small dual bound of the form: $f(x_t) - f(x^*) \leq \nabla f(x_t)(x_t - x^*) \leq \Phi_t$ for some $\Phi_t > 0$. While the accuracy K is presented here as a formal argument of the oracle, an oracle implementation might restrict to a fixed value $K > 1$, which often makes implementation easier. We point out that the cases (1) and (2) potentially overlap if $K > 1$. This is intentional and in this case it is unspecified which of the cases the oracle should choose (and it does not matter for the algorithms).

This new oracle encapsulates the smart use of the *original* linear optimization oracle, even though for some problems it could potentially be implemented directly without relying on a linear programming oracle. Concretely, a weak separation oracle can be realized by a single call to a linear optimization oracle and as such is no more complex than the original oracle. However it has two important advantages: it allows for *caching* and *early termination*. Caching refers to storing previous solutions, and first searching among them to satisfy the oracle's separation condition. The underlying linear optimization oracle is called only, when none of the cached solutions satisfy the condition. Algorithm 2 formalizes this process. Early termination is the technique to stop the linear optimization algorithm before it finishes at an appropriate stage, when from its internal data a suitable oracle answer can be easily recovered; this is clearly an implementation dependent technique. The two techniques can be combined, e.g., Algorithm 2 could use an early terminating linear oracle or other implementation of the weak separation oracle in line 4.

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

Input: linear objective $c \in \mathbb{R}^n$, point $x \in P$, accuracy $K \geq 1$, objective value $\Phi > 0$;
Output: Either (1) vertex $y \in P$ 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: $y \leftarrow \text{argmax}_{x \in P}\, cx$ {LP call}
5: **if** $c(x - y) > \Phi/K$ **then**
6: add y to cache
7: **return** y
8: **else**
9: **return false**
10: **end if**
11: **end if**

We call *lazification* the technique of replacing a linear programming oracle with a much weaker one, and we will demonstrate significant speedups in wall-clock performance (see e.g., Figure 12), 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 and Kale (2012); Garber and Meshi (2016); Garber and 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 time compared to their linear optimization counterparts.

Related Work

There has been extensive work on Frank–Wolfe algorithms and conditional gradient algorithms, so we will restrict to review work most closely related to ours. The Frank–Wolfe algorithm was originally introduced in Frank and Wolfe (1956) (also known as conditional gradient descent Levitin and 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 and 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 and Hazan (2013) as well as the pairwise conditional gradient variant of Garber and 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, the *Pairwise Conditional Gradient* algorithm, as well as the *Block-Coordinate Frank–Wolfe* algorithm. Recently, in Freund and 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 and Jaggi (2015) for an overview with respect to global linear convergence.

It was also recently shown in Hazan and Kale (2012) that the Frank–Wolfe algorithm can be adjusted to the online learning setting and in this work 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 and Vempala (2005); Audibert et al. (2013); Neu and 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 and 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 however much more lazy, requiring no 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.

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 and Kale (2012); Garber and Hazan (2013); Garber and 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.

A previous version of this work appeared as extended abstract in Braun et al. (2017); this version has been significantly revised over the conference version including a representative subset of more extensive computational results, full proofs for all described variants, as well as a variant that uses an augmentation oracle instead of linear optimization oracle (see Section 7).

Outline

We briefly recall notation and notions in Section 2 and consider conditional gradient algorithms in Section 3. In Section 4 we consider parameter-free variants of the proposed algorithms, and in Section 5 we examine online versions. Finally, in Section 7 we show a realization of a weak separation oracle with an even weaker oracle in the case of combinatorial problem and we provide extensive computational results in Section 8.

2. Preliminaries

Let $\|\cdot\|$ be an arbitrary norm on $\mathbb{R}^n$, and let $\|\cdot\|^*$ denote the dual norm of $\|\cdot\|$. 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 Gradient

We start with the most basic Frank–Wolfe algorithm as a simple example for lazifying by means of a *weak separation oracle*. We then lazify more complex Frank–Wolfe algorithms in Garber and Hazan (2013) and Garber and Meshi (2016). Throughout this section $\|\cdot\|$ denotes the ℓ_2-norm.

3.1. Lazy Conditional Gradient: a basic example

We start with lazifying the original Frank–Wolfe algorithm (arguably the simplest Conditional Gradient 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 3) 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 weak separation oracle provides an improving vertex v_t we refer to this as a *positive call* and if the oracle claims there are no improving vertices we call it a *negative call*.

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

Theorem 1 *Assume f is convex and smooth with curvature C. Then Algorithm 3 with $\gamma_t = \frac{2(K^2+1)}{K(t+K^2+2)}$ and $f(x_1) - f(x^*) \leq \Phi_0$ 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*

$$f(x_t) - f(x^*) \leq \Phi_{t-1}.$$

Algorithm 3 Lazy Conditional Gradient

Input: smooth convex function f with curvature C, start vertex $x_1 \in P$, weak linear separation oracle LPsep_P, accuracy $K \geq 1$, step sizes γ_t, initial upper bound Φ_0

Output: points x_t 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: $\quad v_t \leftarrow \mathrm{LPsep}_P(\nabla f(x_t), x_t, \Phi_t, K)$
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 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.

In case of a positive call, i.e., when the oracle returns an improving solution v_t, 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 second inequality by the induction hypothesis and the fact that v_t is an improving solution:

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

In case of a negative call, i.e., when the oracle returns no improving solution, then in particular $\nabla f(x_t)(x_t - x^*) \leq \Phi_t$, hence by Line 5

$$f(x_{t+1}) - f(x^*) = f(x_t) - f(x^*) \leq \nabla f(x_t)(x_t - x^*) \leq \Phi_t.$$

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:

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

Here the last inequality follows from the concrete value of γ_t. ■

Note that by design, the algorithm converges at the worst-case rate that we postulate due to the negative calls when it does not move. Clearly, this is highly undesirable, therefore the algorithm should be understood as the *textbook variant* of lazy conditional gradient. We will present an improved, parameter-free variant of Algorithm 3 in Section 4 that converges at the best possible rate that the non-lazy variant would achieve (up to a small constant factor).

3.2. Lazy Pairwise Conditional Gradient

In this section we provide a lazy variant (Algorithm 4) of the Pairwise Conditional Gradient algorithm from Garber and Meshi (2016), using separation instead of linear optimization. We make identical assumptions: the feasible region is a 0/1 polytope, i.e., all vertices of P have only 0/1 entries, and moreover it is given in the form $P = \{x \in \mathbb{R}^n \mid 0 \leq x \leq \mathbb{1}, Ax = b\}$, where $\mathbb{1}$ denotes the all-one vector.

Algorithm 4 Lazy Pairwise Conditional Gradient (LPCG)

Input: polytope P, smooth and S-strongly convex function f with curvature C, accuracy $K \geq 1$, non-increasing step-sizes η_t, eagerness Δ_t

Output: points x_t

1: $x_1 \in P$ arbitrary and $\Phi_0 \geq f(x_1) - f(x^*)$
2: **for** $t = 1, \ldots, T$ **do**
3: $\quad \widetilde{\nabla} f(x_t)_i \coloneqq \begin{cases} \nabla f(x_t)_i & \text{if } x_{t,i} > 0 \\ -\infty & \text{if } x_{t,i} = 0 \end{cases}$
4: $\quad \Phi_t \leftarrow \frac{2\Phi_{t-1} + \eta_t^2 C}{2 + \frac{\eta_t}{K\Delta_t}}$
5: $\quad c_t \leftarrow \left(\nabla f(x_t), -\widetilde{\nabla} f(x_t)\right)$
6: $\quad (v_t^+, v_t^-) \leftarrow \mathrm{LPsep}_{P \times P}\left(c_t,\, (x_t, x_t),\, \frac{\Phi_t}{\Delta_t},\, K\right)$
7: $\quad$ **if** $(v_t^+, v_t^-) =$ **false then**
8: $\quad\quad x_{t+1} \leftarrow x_t$
9: $\quad$ **else**
10: $\quad\quad \tilde{\eta}_t \leftarrow \max\{2^{-\delta} \mid \delta \in \mathbb{Z}_{\geq 0},\ 2^{-\delta} \leq \eta_t\}$
11: $\quad\quad x_{t+1} \leftarrow x_t + \tilde{\eta}_t (v_t^+ - v_t^-)$
12: $\quad$ **end if**
13: **end for**

Observe that Algorithm 4 calls the linear separation oracle 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.

Let $\operatorname{card} x$ denote the number of non-zero entries of the vector x.

Theorem 2 *Let x^* be a minimum point of f in P, and Φ_0 an upper bound of $f(x_1) - f(x^*)$. Furthermore, let $\operatorname{card}(x^*) \leq \alpha$, $M_1 \coloneqq \sqrt{\frac{S}{8\alpha}}$, $\kappa \coloneqq \min\left\{\frac{M_1}{KC}, 1/\sqrt{\Phi_0}\right\}$, $\eta_t \coloneqq \kappa\sqrt{\Phi_{t-1}}$ and*

$\Delta_t \coloneqq \sqrt{\frac{2\alpha\Phi_{t-1}}{S}}$, *then Algorithm 4 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 ((Garber and Meshi, 2016, Lemma 2)) *Let* $x, y \in P$. *Then* x *is a liner combination* $x = \sum_{i=1}^k \lambda_i v_i$ *of some vertices* v_i *of* P *(in particular,* $\sum_{i=1}^k \lambda_i = 1$*) with* $x - y = \sum_{i=1}^k \gamma_i (v_i - z)$ *for some* $0 \leq \gamma_i \leq \lambda_i$ *and* $z \in P$ *such that* $\sum_{i=1}^k \gamma_i \leq \sqrt{\operatorname{card}(y)}\|x-y\|$.

Proof [Proof of Theorem 2] 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 and 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 case of a negative call (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 $c_t(x_t, x_t) = 0$)

$$\widetilde{\nabla} f(x_t) z_2 - \nabla f(x_t) z_1 \leq \frac{\Phi_t}{\Delta_t}$$

and therefore

$$\nabla f(x_t)(\tilde{z}_2 - z_1) \leq \frac{\Phi_t}{\Delta_t}, \tag{2}$$

for all $\tilde{z}_2, z_1 \in P$ with $\operatorname{supp}(\tilde{z}_2) \subseteq \operatorname{supp}(x_t)$, where $\operatorname{supp}(x)$ denotes the set of non-zero coordinates of x. We use Lemma 3 for the decompositions $x_t = \sum_{i=1}^k \lambda_i v_i$ and $x_t - x^* = \sum_{i=1}^k \gamma_i (v_i - z)$ with $0 \leq \gamma_i \leq \lambda_i$, $z \in P$ and

$$\sum_{i=1}^k \gamma_i \leq \sqrt{\operatorname{card}(x^*)}\|x_t - x^*\| \leq \sqrt{\frac{2\operatorname{card}(x^*)\Phi_{t-1}}{S}} \leq \Delta_t,$$

using the induction hypothesis and strong convexity in the second inequality. Then

$$f(x_{t+1}) - f(x^*) = f(x_t) - f(x^*) \leq \nabla f(x_t)(x_t - x^*) = \sum_{i=1}^k \gamma_i \cdot \underbrace{\nabla f(x_t)(v_i - z)}_{\leq \Phi_t/\Delta_t} \leq \Phi_t,$$

where we used Equation (2) for the last inequality.

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

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

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

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

■

3.3. Lazy Local Conditional Gradient

In this section we provide a lazy version (Algorithm 5) of the conditional gradient algorithm from Garber and Hazan (2013). Let $P \subseteq \mathbb{R}^n$ be any polytope, D denote an upper bound on the ℓ_2-diameter of P, and $\mu \geq 1$ be an affine invariant parameter of P satisfying Lemma 4 below, see (Garber and Hazan, 2013, Section 2) for a possible definition. As the algorithm is not affine invariant by nature, we need a non-invariant version of smoothness: Recall that a convex function f is *β-smooth* if

$$f(y) - f(x) \leq \nabla f(x)(y - x) + \beta \|y - x\|^2/2.$$

Algorithm 5 Lazy Local Conditional Gradient (LLCG)

Input: feasible polytope P, β-smooth and S-strongly convex function f, parameters K, S, β, μ; diameter D
Output: points x_t

1: $x_1 \in P$ arbitrary and $\Phi_0 \geq f(x_1) - f(x^*)$
2: $\alpha \leftarrow \frac{S}{2K\beta n\mu^2}$
3: **for** $t = 1, \ldots, T$ **do**
4: $\quad r_t \leftarrow \sqrt{\frac{2\Phi_{t-1}}{S}}$
5: $\quad \Phi_t \leftarrow \frac{\Phi_{t-1} + \frac{\beta}{2}\alpha^2 \min\{n\mu^2 r_t^2, D^2\}}{1+\alpha/K}$
6: $\quad p_t \leftarrow \mathrm{LLPsep}_P\left(\nabla f(x_t), x_t, r_t, \Phi_t, K\right)$
7: $\quad$ **if** $p_t =$ **false then**
8: $\quad\quad x_{t+1} \leftarrow x_t$
9: $\quad$ **else**
10: $\quad\quad x_{t+1} \leftarrow x_t + \alpha(p_t - x_t)$
11: $\quad$ **end if**
12: **end for**

Algorithm 6 Weak Local Separation $\mathrm{LLPsep}_P(c, x, r, \Phi, K)$

Input: polytope P together with invariant μ, linear objective $c \in \mathbb{R}^n$, point $x \in P$, radius $r > 0$, objective value $\Phi > 0$, accuracy $K \geq 1$

Output: Either (1) $y \in P$ with $\|x - y\| \leq \sqrt{n}\mu r$ and $c(x - y) > \Phi/K$, or (2) **false**: $c(x - z) \leq \Phi$ for all $z \in P \cap \mathbb{B}_r(x)$.

1: $\Delta \leftarrow \min\left\{\frac{\sqrt{n}\mu}{D} r, 1\right\}$
2: Decompose x: $x = \sum_{j=1}^{M} \lambda_j v_j$, $\lambda_j > 0$, $\sum_j \lambda_j = 1$.
3: Sort vertices: $i_1, \ldots, i_M \quad cv_{i_1} \geq \cdots \geq cv_{i_M}$.
4: $k \leftarrow \min\{k : \sum_{j=1}^{k} \lambda_{i_j} \geq \Delta\}$
5: $p_- \leftarrow \sum_{j=1}^{k-1} \lambda_{i_j} v_{i_j} + \left(\Delta - \sum_{j=1}^{k-1} \lambda_{i_j}\right) v_{i_k}$
6: $v^* \leftarrow \mathrm{LPsep}_P\left(c, \frac{p_-}{\Delta}, \frac{\Phi}{\Delta}\right)$
7: **if** $v^* =$ **false then**
8: **return false**
9: **else**
10: **return** $y \leftarrow x - p_- + \Delta v^*$
11: **end if**

As an intermediary step, we first implement a *local weak separation oracle* in Algorithm 6, a *local* version of Oracle 1, which finds improving points only in a small neighborhood of the original point, analogously to the local linear optimization oracle in Garber and Hazan (2013). To this end, we recall a technical lemma from Garber and Hazan (2013).

Lemma 4 *(Garber and Hazan, 2013, Lemma 7) Let $P \subseteq \mathbb{R}^n$ be a polytope and $v_1, \ldots, v_N$ be its vertices. Let $x, y \in P$ and $x = \sum_{i=1}^{N} \lambda_i v_i$ a convex combination of the vertices of P. Then there are numbers $0 \leq \gamma_i \leq \lambda_i$ and $z \in P$ satisfying*

$$x - y = \sum_{i \in [N]} \gamma_i (z - v_i)$$

$$\sum_{i \in [N]} \gamma_i \leq \frac{\sqrt{n}\mu}{D} \|x - y\|.$$

Now we prove the correctness of the weak local separation algorithm.

Lemma 5 *Algorithm 6 is correct. In particular* $\mathrm{LLPsep}_P(c, x, r, \Phi, K)$

(i) returns either an $y \in P$ with $\|x - y\| \leq \sqrt{n}\mu r$ and $c(x - y) \geq \Phi/K$,

*(ii) or returns **false**, and then $c(x - z) \leq \Phi$ for all $z \in P \cap \mathbb{B}_r(x)$.*

Proof *We first consider the case when the algorithm exits in Line 10. Observe that $y \in P$ since y is a convex combination of vertices of by construction: $y = \sum_{j=1}^{M} (\lambda_{i_j} - \gamma_j) v_{i_j} + \Delta v^*$ with $\Delta = \sum_{j=1}^{M} \gamma_j \leq \frac{\sqrt{n}\mu}{D} r$, where $\gamma_j = \lambda_{i_j}$ for $j < k$, and $\gamma_k = \Delta - \sum_{j=1}^{k-1} \lambda_{i_j}$, and $\gamma_j = 0$ for $j > k$. Therefore*

$$\|x - y\| = \left\| \sum_{j=1}^{M} \gamma_j (v_{i_j} - v^*) \right\| \leq \sum_{j=1}^{M} \gamma_j \|v_{i_j} - v^*\| \leq \sqrt{n}\mu r.$$

Finally using the guarantee of LPsep_P *we get*

$$c(x-y) = \Delta c\left(\frac{p_-}{\Delta} - v^*\right) \geq \frac{\Phi}{K}.$$

If the algorithm exits in Line 8, we use Lemma 4 to decompose any $y \in P \cap \mathbb{B}_r(x)$*:*

$$x - y = \sum_{i=1}^{M} \gamma_i (v_i - z),$$

with $z \in P$ *and* $\sum_{i=1}^{M} \gamma_i \leq \frac{\sqrt{n}\mu}{D}\|x-y\| \leq \Delta$*. Since* $\sum_{i=1}^{M} \lambda_i = 1 \geq \Delta$*, there are numbers* $\gamma_i \leq \eta_i^- \leq \lambda_i$ *with* $\sum_{i=1}^{M} \eta_i^- = \Delta$*. Let*

$$\tilde{p}_- \coloneqq \sum_{i=1}^{M} \eta_i^- v_i,$$

$$\tilde{p}_+ \coloneqq y - x + \tilde{p}_- = \sum_{i=1}^{M} (\eta_i^- - \gamma_i) v_i + \sum_{i=1}^{M} \gamma_i z,$$

so that $\tilde{p}_+/\Delta \in P$*. To bound the function value we first observe that the choice of* p_- *in the algorithm assures that* $cu \leq cp_-$ *for all* $u = \sum_{i=1}^{M} \eta_i v_i$ *with* $\sum_{i=1}^{M} \eta_i = \Delta$ *and all* $0 \leq \eta_i \leq \lambda_i$*. In particular,* $c\tilde{p}_- \leq cp_-$*. The function value of the positive part* $\tilde{p}_+$ *can be bounded with the guarantee of* LPsep_P*:*

$$c\left(\frac{p_-}{\Delta} - \frac{\tilde{p}_+}{\Delta}\right) \leq \frac{\Phi}{\Delta},$$

i.e., $c(p_- - \tilde{p}_+) \leq \Phi$*. Finally combining these bounds gives*

$$c(x-y) = c\left(\tilde{p}_- - \tilde{p}_+\right) \leq c(p_- - \tilde{p}_+) \leq \Phi$$

as desired. ■

We are ready to examine the Conditional Gradient Algorithm based on LLPsep_P:

Theorem 6 *Algorithm 5 converges with the following rate:*

$$f(x_{t+1}) - f(x^*) \leq \Phi_t \leq \Phi_0 \left(\frac{1+\alpha/(2K)}{1+\alpha/K}\right)^t.$$

Proof *The proof is similar to the proof of Theorem 2. We prove this rate by induction. For* $t = 0$ *the choice of* Φ_0 *guarantees that* $f(x_1) - f(x^*) \leq \Phi_0$*. Now assume the theorem holds for* $t \geq 0$*. With strong convexity and the induction hypothesis we get*

$$\|x_t - x^*\|^2 \leq \frac{2}{S}(f(x_t) - f(x^*)) \leq \frac{2}{S}\Phi_{t-1} = r_t^2,$$

i.e., $x^* \in P \cap \mathbb{B}_{r_t}(x_t)$*. In case of a negative call, i.e., when* $p_t =$ **false***, then case (ii) of Lemma 5 applies:*

$$f(x_{t+1}) - f(x^*) = f(x_t) - f(x^*) \leq \nabla f(x_t)(x_t - x^*) \leq \Phi_t.$$

In case of a positive call, i.e., when Line 10 is executed, we get the same inequality via:

$$\begin{aligned} f(x_{t+1}) - f(x^*) &\leq \Phi_{t-1} + \alpha \nabla f(x_t)(p_t - x_t) + \frac{\beta}{2}\alpha^2 \|x_t - p_t\|^2 \\ &\leq \Phi_{t-1} - \alpha \frac{\Phi_t}{K} + \frac{\beta}{2}\alpha^2 \min\{n\mu^2 r_t^2, D^2\} \\ &= \Phi_t. \end{aligned}$$

Therefore using the definition of α and r_t we get the desired bound:

$$\Phi_t \leq \frac{\Phi_{t-1} + \frac{\beta}{2}\alpha^2 r_t^2 n\mu^2}{1 + \alpha/K} = \Phi_{t-1}\left(\frac{1 + \alpha/(2K)}{1 + \alpha/K}\right) \leq \Phi_0 \left(\frac{1 + \alpha/(2K)}{1 + \alpha/K}\right)^t.$$

■

4. Parameter-free Conditional Gradient via Weak Separation

In this section we provide a parameter-free variant of the Lazy Frank–Wolfe Algorithm, which is inspired by Pokutta (2017) and which exhibits a very favorable behavior in computations; the same technique applies to all other variants from Section 3 as well. The idea is that instead of using predetermined values for progress parameters, like Φ_t and γ_t in Algorithm 3, to optimize worst-case progress, the parameters are adjusted adaptively, using data encountered by the algorithm, and avoiding hard-to-estimate parameters, like the curvature C. In practice, this leads to faster convergence, as usual for adaptive methods, while the theoretical convergence rate is worse only by a small constant factor. See Figure 14 for a comparison and Section 8.1.1 for more experimental results.

The occasional halving of the Φ_t is reminiscent of an adaptive restart strategy, considering successive iterates with the same Φ_t as an epoch. It ensures that Φ_t is always at least half of the primal gap, while quickly reducing it if it is too large for the algorithm to make progress, and as such it represents a reasonable amount of expected progress throughout the whole run of the algorithm, not just at the initial iterates.

Remark 7 (Additional LP call for initial bound) *Note that Algorithm 7 finds a tight initial bound Φ_0 with a single extra LP call. If this is undesired, this can be also done approximately as long as Φ_0 is a valid upper bound, for example by means of binary search via the weak separation oracle.*

Theorem 8 *Let f be a smooth convex function with curvature C. Algorithm 7 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*

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

Proof *The main idea of the proof is to maintain an approximate upper bound on the optimality gap. We then show that negative calls halve the upper bound $2\Phi_t$ and positive oracle calls make significant objective function improvement.*

Algorithm 7 Parameter-free Lazy Conditional Gradient (LCG)

Input: smooth convex function f, start vertex $x_1 \in P$, weak linear separation oracle LPsep_P, accuracy $K \geq 1$
Output: points x_t 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: $\gamma_t \leftarrow \mathrm{argmin}_{0 \leq \gamma \leq 1} f((1-\gamma)x_t + \gamma v_t)$ {Line search}
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**

*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

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

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

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

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

As seen from the proof, the algorithm receives few negative oracle calls by design; these are usually more expensive than positive ones as the oracle has to compute a certificate by, e.g., executing a full linear optimization oracle call.

Corollary 9 *Algorithm 7 receives at most* $\lceil \log \Phi_0/\varepsilon \rceil + 1$ *negative oracle answers.*

Remark 10 (Improved use of Linear Optimization oracle) *A possible improvement to Line 6 is* $\Phi_t \leftarrow \max_{x \in P} \nabla f(x_t)(x_t - x)/2$*, assuming that at a negative call the oracle also provides the dual gap* $\max_{x \in P} \nabla f(x_t)(x_t - x)$ *as well as the minimizer* $\bar{x} \in P$ *of the oracle call. This is the case e.g., when the weak separation oracle is implemented as in Algorithm 2. Clearly, the minimizer* $\bar{x}$ *can be also used to perform a progress step; albeit without guarantee w.r.t. to* Φ_t*.*

Remark 11 (Line Search) *If line search is too expensive we can choose* $\gamma_t = \min\{1, \Phi_t/KC\}$ *in Algorithm 7. In this case an estimate of the curvature* C *is required.*

5. Lazy Online Conditional Gradient

In this section we lazify the online conditional gradient algorithm of Hazan and Kale (2012) over arbitrary polytopes $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$, resulting in Algorithm 8. We slightly improve constant factors by replacing (Hazan and Kale, 2012, Lemma 3.1) with a better estimation via solving a quadratic inequality arising from strong convexity. In this section the norm $\|\cdot\|$ can be arbitrary.

Theorem 12 *Let* $0 \leq b, s < 1$*. Let* $K \geq 1$ *be an accuracy parameter. Assume* f_t *is L-Lipschitz, and smooth with curvature at most* Ct^{-b}*. Let* $D \coloneqq \max_{y_1, y_2 \in P} \|y_1 - y_2\|$ *denote the diameter of* P *in norm* $\|\cdot\|$*. Then the following hold for the points* x_t *computed by Algorithm 8 where* x_T^* *is the minimizer of* $\sum_{t=1}^T f_t$*:*

(i) With the choice

$$\gamma_t = t^{-(1-b)/2},$$

the x_t *satisfy*

$$\frac{1}{T} \sum_{t=1}^{T} (f_t(x_T) - f_t(x_T^*)) \leq AT^{-(1-b)/2},$$

where

$$A \coloneqq \frac{CK}{2(1-b)} + L(K+1)D.$$

(ii) Moreover, if all the f_t *are* St^{-s}*-strongly convex, then with the choice*

$$\gamma_t = t^{(b+s-2)/3},$$

Algorithm 8 Lazy Online Conditional Gradient (LOCG)

Input: functions f_t, start vertex $x_1 \in P$, weak linear separation oracle LPsep_P, parameters K, C, b, S, s; diameter D

Output: points x_t

1: **for** $t = 1$ **to** $T - 1$ **do**
2: $\quad \nabla_t \leftarrow \nabla f_t(x_t)$
3: $\quad$ **if** $t = 1$ **then**
4: $\quad\quad h_1 \leftarrow \min\{\|\nabla_1\|^* D, 2\|\nabla_1\|^{*2}/S\}$
5: $\quad$ **else**
6: $\quad\quad h_t \leftarrow \Phi_{t-1} + \min\left\{\|\nabla_t\|^* D, \frac{\|\nabla_t\|^{*2}}{St^{1-s}} + 2\sqrt{\frac{\|\nabla_t\|^{*2}}{2St^{1-s}}\left(\frac{\|\nabla_t\|^{*2}}{2St^{1-s}} + \Phi_{t-1}\right)}\right\}$
7: $\quad$ **end if**
8: $\quad \Phi_t \leftarrow \frac{h_t + \frac{Ct^{1-b}\gamma_t^2}{2(1-b)}}{1+\frac{\gamma_t}{K}}$
9: $\quad v_t \leftarrow \mathrm{LPsep}_P(\sum_{i=1}^t \nabla f_i(x_t), x_t, \Phi_t, K)$
10: $\quad$ **if** $v_t =$ **false then**
11: $\quad\quad x_{t+1} \leftarrow x_t$
12: $\quad$ **else**
13: $\quad\quad x_{t+1} \leftarrow (1-\gamma_t)x_t + \gamma_t v_t$
14: $\quad\quad \Phi_t \leftarrow h_t - \sum_{i=1}^t f_i(x_t) + \sum_{i=1}^t f_i(x_{t+1})$
15: $\quad$ **end if**
16: **end for**

the x_t satisfy

$$\frac{1}{T}\sum_{t=1}^{T}(f_t(x_T) - f_t(x_T^*)) \le AT^{-(2(1+b)-s)/3}, \tag{3}$$

where

$$A \coloneqq 2\left((K+1)(K+2)\frac{L^2}{S} + \frac{CK}{2(1-b)}\right).$$

Proof *We prove only Claim (ii), as the proof of Claim (i) is similar and simpler. Let $F_T \coloneqq \sum_{t=1}^T f_t$. Furthermore, let $\overline{h}_T \coloneqq AT^{1-(2(1+b)-s)/3}$ be T times the right-hand side of Equation (3). In particular, F_T is S_T-strongly convex, and smooth with curvature at most C_{F_T} where*

$$C_{F_T} \coloneqq \frac{CT^{1-b}}{1-b} \ge C\sum_{t=1}^{T} t^{-b}, \qquad S_T \coloneqq ST^{1-s} \le S\sum_{t=1}^{T} t^{-s}.$$

*We prove $F_t(x_t) - F_t(x_t^*) \le h_t \le \overline{h}_t$ by induction on t. The case $t = 1$ is clear. Let $\overline{\Phi}_t$ denote the value of Φ_t in Line 8, while we reserve Φ_t to denote its value as used in Line 6. We start by showing $F_t(x_{t+1}) - F_t(x_t^*) \le \Phi_t \le \overline{\Phi}_t$. We distinguish two cases depending on the oracle answer v_t from Line 9. For a negative oracle answer ($v_t =$ **false**), we have $\Phi_t = \overline{\Phi}_t$ and the weak separation oracle asserts $\max_{y\in P} \nabla F_t(x_t)(x_t - y) \le \Phi_t$, which combined with the convexity of F_t provides*

$$F_t(x_{t+1}) - F_t(x_t^*) = F_t(x_t) - F_t(x_t^*) \le \nabla F_t(x_t)(x_t - x_{t^*}) \le \Phi_t = \overline{\Phi}_t.$$

Otherwise, for a positive oracle answer, Line 14 and the induction hypothesis provides $F_t(x_{t+1}) - F_t(x_t^*) \leq h_t + F_t(x_{t+1}) - F_t(x_t) = \Phi_t$. *To prove* $\Phi_t \leq \overline{\Phi}_t$, *we apply the smoothness of* F_t *followed by the inequality provided by the choice of* v_t*:*

$$F_t(x_{t+1}) - F_t(x_t) - \frac{C_{F_t}\gamma_t^2}{2} \leq \nabla F_t(x_t)(x_{t+1} - x_t) = \gamma_t \nabla F_t(x_t)(v_t - x_t) \leq -\frac{\gamma_t \overline{\Phi}_t}{K}.$$

Rearranging provides the inequality:

$$\Phi_t = h_t + F_t(x_{t+1}) - F_t(x_t) \leq h_t - \frac{\gamma_t \overline{\Phi}_t}{K} + \frac{C_{F_t}\gamma_t^2}{2} = \overline{\Phi}_t.$$

For later use, we bound the difference between $\overline{h}_t$ *and* $\overline{\Phi}_t$ *using the value of parameters,* $h_t \leq \overline{h}_t$, *and* $\gamma_t \leq 1$*:*

$$\overline{h}_t - \overline{\Phi}_t \geq \overline{h}_t - \frac{\overline{h}_t + \frac{C_{F_t}\gamma_t^2}{2}}{1 + \frac{\gamma_t}{K}} = \frac{\frac{\overline{h}_t \gamma_t}{K} - \frac{C_{F_t}\gamma_t^2}{2}}{1 + \frac{\gamma_t}{K}} \geq \frac{\frac{\overline{h}_t \gamma_t}{K} - \frac{C_{F_t}\gamma_t^2}{2}}{1 + \frac{1}{K}} = \frac{A - \frac{CK}{2(1-b)}}{K+1} t^{[2s-(1+b)]/3}.$$

We now apply $F_t(x_{t+1}) - F_t(x_t^*) \leq \Phi_t$, *together with convexity of* f_{t+1}, *and the minimality* $F_t(x_t^*) \leq F_t(x_{t+1}^*)$ *of* x_t^*, *followed by strong convexity of* F_{t+1}*:*

$$\begin{aligned} F_{t+1}(x_{t+1}) - F_{t+1}(x_{t+1}^*) &\leq (F_t(x_{t+1}) - F_t(x_t^*)) + (f_{t+1}(x_{t+1}) - f_{t+1}(x_{t+1}^*)) \\ &\leq \Phi_t + \|\nabla_{t+1}\|^* \cdot \|x_{t+1} - x_{t+1}^*\| \\ &\leq \Phi_t + \|\nabla_{t+1}\|^* \sqrt{\frac{2}{S_{t+1}}(F_{t+1}(x_{t+1}) - F_{t+1}(x_{t+1}^*))}. \end{aligned} \tag{4}$$

Solving the quadratic inequality provides

$$F_{t+1}(x_{t+1}) - F_{t+1}(x_{t+1}^*) \leq \Phi_t + \frac{\|\nabla_{t+1}\|^{*2}}{S_{t+1}} + 2\sqrt{\frac{\|\nabla_{t+1}\|^{*2}}{2S_{t+1}}\left(\frac{\|\nabla_{t+1}\|^{*2}}{2S_{t+1}} + \Phi_t\right)}. \tag{5}$$

From Equation (4), *ignoring the last line, we also obtain* $F_{t+1}(x_{t+1}) - F_{t+1}(x_{t+1}^*) \leq \Phi_t + \|\nabla_{t+1}\|^* D$ *via the estimate* $\|x_{t+1} - x_{t+1}^*\| \leq D$. *Thus* $F_{t+1}(x_{t+1}) - F_{t+1}(x_{t+1}^*) \leq h_{t+1}$, *by Line 6, as claimed.*

Now we estimate the right-hand side of Equation (5) *by using the actual value of the parameters, the estimate* $\|\nabla_{t+1}\|^* \leq L$, *and the inequality* $s + b \leq 2$. *In fact, we estimate a proxy for the right-hand side. Note that* A *was chosen to satisfy the second inequality:*

$$\begin{aligned} \frac{L^2}{S_{t+1}} + 2\sqrt{\frac{L^2}{2S_{t+1}}\overline{h}_t} &\leq \frac{L^2}{St^{1-s}} + 2\sqrt{\frac{L^2}{2St^{1-s}}\overline{h}_t} \leq \frac{L^2}{S} t^{[2s-(1+b)]/3} + 2\sqrt{\frac{L^2}{2St^{1-s}}\overline{h}_t} \\ &= \left(\frac{L^2}{S} + \sqrt{2\frac{L^2}{S}A}\right) t^{[2s-(1+b)]/3} \leq \frac{A - \frac{CK}{2(1-b)}}{K+1} t^{[2s-(1+b)]/3} \\ &\leq \overline{h}_t - \overline{\Phi}_t \leq \overline{h}_t - \Phi_t. \end{aligned}$$

In particular, $\frac{L^2}{2S_{t+1}} + \Phi_t \leq \overline{h}_t$ *hence combining with Equation* (5) *we obtain*

$$\begin{aligned} h_{t+1} &\leq \Phi_t + \frac{L^2}{S_{t+1}} + 2\sqrt{\frac{L^2}{2S_{t+1}}\left(\frac{L^2}{2S_{t+1}} + \Phi_t\right)} \\ &\leq \Phi_t + \frac{L^2}{S_{t+1}} + 2\sqrt{\frac{L^2}{2S_{t+1}}\overline{h}_t} \\ &\leq \overline{h}_t \leq \overline{h}_{t+1}. \end{aligned}$$

■

5.1. Stochastic and Adversarial Versions

Complementing the offline algorithms from Section 3, we will now derive various online versions. The presented cases here are similar to those in Hazan and Kale (2012) and thus we state them without proof.

For stochastic cost functions f_t, we obtain bounds from Theorem 12 (i) similar to (Hazan and Kale, 2012, Theorems 4.1 and 4.3) (with δ replaced by δ/T in the bound to correct an inaccuracy in the original argument). The proof is analogous and hence omitted, but note that $\|y_1 - y_2\|_2 \leq \sqrt{\|y_1 - y_2\|_1 \|y_1 - y_2\|_\infty} \leq \sqrt{k}$ for all $y_1, y_2 \in P$.

Corollary 13 *Let* f_t *be convex functions sampled i.i.d. with expectation* $\mathbb{E}[f_t] = f^*$*, and* $\delta > 0$*. Assume that the* f_t *are* L*-Lipschitz in the* 2*-norm.*

(i) If all the f_t *are smooth with curvature at most* C*, then Algorithm 8 applied to the* f_t *(with* $b = 0$*) yields with probability* $1 - \delta$

$$\sum_{t=1}^{T} f^*(x_t) - \min_{x \in P} \sum_{t=1}^{T} f^*(x) \leq O\left(C\sqrt{T} + Lk\sqrt{nT \log(nT^2/\delta) \log T}\right).$$

(ii) Without any smoothness assumption, Algorithm 8 (applied to smoothenings of the f_t*) provides with probability* $1 - \delta$

$$\sum_{t=1}^{T} f^*(x_t) - \min_{x \in P} \sum_{t=1}^{T} f^*(x) \leq O\left(\sqrt{n}LkT^{2/3} + Lk\sqrt{nT \log(nT^2/\delta) \log T}\right).$$

Similar to (Hazan and Kale, 2012, Theorem 4.4), from Theorem 12 (ii) we obtain the following regret bound for adversarial cost functions with an analogous proof.

Corollary 14 *For any* L*-Lipschitz convex cost functions* f_t*, Algorithm 8 applied to the functions* $\tilde{f}_t(x) \coloneqq \nabla f_t(x_t)x + \frac{2L}{\sqrt{k}} t^{-1/4} \|x - x_1\|_2^2$ *(with* $b = s = 1/4$*,* $C = L\sqrt{k}$*,* $S = L/\sqrt{k}$*, and Lipschitz constant* $3L$*) achieving regret*

$$\sum_{t=1}^{T} f_t(x_t) - \min_{x \in P} \sum_{t=1}^{T} f_t(x) \leq O(L\sqrt{k}T^{3/4})$$

with at most T *calls to the weak separation oracle.*

Note that the gradient of the $\tilde{f}_t$ are easily computed via the formula $\nabla \tilde{f}_t(x) = \nabla f_t(x_t) + 4Lt^{-1/4}(x - x_1)/\sqrt{k}$, particularly because the gradient of the f_t need not be recomputed, so that we obtain a weak separation-based stochastic gradient descent algorithm, where we only have access to the f_t through a stochastic gradient oracle, while retaining all the favorable properties of the Frank–Wolfe algorithm with a convergence rate $O(T^{-1/4})$ (c.f., Garber and Hazan (2013)).

6. Non-polytopal domains

So far we have formulated our results for the polytopal case as most of the base methods that we lazify are usually formulated for polytopal domains. However, whenever a base method extends to general compact convex sets as domains then so does our lazification of the base method. In fact it is not even required to use vertices or extremal points as answers to either the LP oracle or weak-separation oracle calls; this is usually only required to obtain iterates as convex combinations of *extremal points* but it is not necessary for convergence.

Base methods that extend to general compact convex sets in particular include the vanilla Frank–Wolfe algorithm, the Away-Step Frank–Wolfe algorithm, and the Pairwise Frank–Wolfe algorithm. Note, however that for the Away-Step Frank–Wolfe algorithm and the Pairwise Frank–Wolfe algorithm (irrespective of lazification) it is not known whether a linear rate of convergence can be achieved for strongly convex functions over general compact convex sets. Base variants that do not readily apply to the non-polyhedral case are the variants in Garber and Meshi (2016) (see Section 3.2) and Garber and Hazan (2013) (see Section 3.3).

We further would like to mention, that often (but not always) for non-polyhedral domains the linear optimization oracle might be more expensive, so that lazification might offer attractive benefits in this case as the effect of caching and early termination might be even more pronounced. We present computational tests for non-polyedral domains for matrix completion, where the feasible domain is given via $\|X\|_* \leq R$, where $\|\cdot\|_*$ is the (non-polyhedral) nuclear norm (see Section 8.1).

7. Weak Separation through Augmentation

So far we realized the weak separation oracle via lazy optimization. We will now create a (weak) separation oracle for integral polytopes, employing an even weaker, so-called augmentation oracle, which only provides an improving solution but provides no guarantee with respect to optimality. We call this approach *lazy augmentation*. This is especially useful when a fast augmentation oracle is available or the vertices of the underlying polytope P are particularly sparse, i.e., $\|y_1 - y_2\|_1 \leq k \ll n$ for all $y_1, y_2 \in P$, where n is the ambient dimension of P. As before theoretical convergence rates are maintained.

For simplicity of exposition we restrict to 0/1 polytopes P here. For general integral polytopes, one considers a so-called *directed augmentation oracle*, which can be similarly linearized after splitting variables in positive and negative parts; we refer the interested reader to see Schulz and Weismantel (2002); Bodic et al. (2015) for an in-depth discussion.

Let k denote the ℓ_1-diameter of P. Upon presentation with a 0/1 solution x and a linear objective $c \in \mathbb{R}^n$, an augmentation oracle either provides an improving 0/1 solution $\bar{x}$ with $c\bar{x} < cx$ or asserts optimality for c:

Oracle 2 Linear Augmentation Oracle AUG_P(c, x)
Input: linear objective $c \in \mathbb{R}^n$, vertex $x \in P$
Output: vertex $\bar{x} \in P$ with $c\bar{x} < cx$ when exists, otherwise $\bar{x} = x$

Such an oracle is significantly weaker than a linear optimization oracle but also significantly easier to implement and much faster; we refer the interested reader to Grötschel and Lovász (1993); Schulz et al. (1995); Schulz and Weismantel (2002) for an extensive list of examples. While augmentation and optimization are polynomially equivalent (even for convex integer programming Oertel et al. (2014)) the current best linear optimization algorithms based on an augmentation oracle are slow for general objectives. While optimizing an *integral* objective $c \in \mathbb{R}^n$ needs $O(k \log\|c\|_\infty)$ calls to an augmentation oracle (see Schulz et al. (1995); Schulz and Weismantel (2002); Bodic et al. (2015)), a general objective function, such as the gradient in Frank–Wolfe algorithms has only an $O(kn^3)$ guarantee in terms of required oracle calls (e.g., via simultaneous diophantine approximations Frank and Tardos (1987)), which is not desirable for large n. In contrast, here we use an augmentation oracle to perform separation, without finding the optimal solution. Allowing a multiplicative error $K > 1$, we realize an augmentation-based weak separation oracle (see Algorithm 9), which decides given a linear objective function $c \in \mathbb{R}^n$, an objective value $\Phi > 0$, and a starting point $x \in P$, whether there is a $y \in P$ with $c(x - y) > \Phi/K$ or $c(x - y) \leq \Phi$ for all $y \in P$. In the former case, it actually provides a certifying $y \in P$, i.e., with $c(x - y) > \Phi/K$. Note that a constant accuracy K requires a linear number of oracle calls in the diameter k, e.g., $K = (1 - 1/e)^{-1} \approx 1.582$ needs at most $N \leq k$ oracle calls, which can be much smaller than the ambient dimension of the polytope.

At the beginning, in Line 2, the algorithm has to replace the input point x with an integral point x_0. If the point x is given as a convex combination of integral points, then a possible solution is to evaluate the objective c on these integral points, and choose x_0 the first one with $cx_0 \leq cx$. This can be easily arranged for Frank–Wolfe algorithms as they maintain convex combinations.

Proposition 15 *Assume $\|y_1 - y_2\|_1 \leq k$ for all $y_1, y_2 \in P$. Then Algorithm 9 is correct, i.e., it outputs either (1) $y \in P$ with $c(x - y) > \Phi/K$, or (2)* **false**. *In the latter case $c(x - y) \leq \Phi$ for all $y \in P$ holds. The algorithm calls AUG_P at most $N \leq \lceil \log(1 - 1/K)/\log(1 - 1/k) \rceil$ many times.*

Proof *First note that $(\mathbb{1} - 2x)v + \|x\|_1 = \|v - x\|_1$ for $x, v \in \{0, 1\}^n$, hence Line 7 is equivalent to $x_i \leftarrow \mathrm{AUG}_P(c + \frac{\Phi - c(x - x_{i-1})}{k}\|\cdot - x_{i-1}\|_1, x_{i-1})$.*

The algorithm obviously calls the oracle at most N times by design, and always returns a value, so we need to verify only the correctness of the returned value. We distinguish cases according to the output.

Algorithm 9 Augmenting Weak Separation $\mathrm{LPsep}_P(c, x, \Phi, K)$

Input: linear objective $c \in \mathbb{R}^n$, point $x \in P$, objective value $\Phi > 0$; accuracy $K > 1$
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: $N \leftarrow \lceil \log(1 - 1/K) / \log(1 - 1/k) \rceil$
2: Choose $x_0 \in P$ vertex with $cx_0 \leq cx$.
3: **for** $i = 1$ **to** N **do**
4: **if** $c(x - x_{i-1}) \geq \Phi$ **then**
5: **return** x_{i-1}
6: **end if**
7: $x_i \leftarrow \mathrm{AUG}_P(c + \frac{\Phi - c(x - x_{i-1})}{k}(\mathbb{1} - 2x_{i-1}), x_{i-1})$
8: **if** $x_i = x_{i-1}$ **then**
9: **return false**
10: **end if**
11: **end for**
12: **return** x_N

Clearly, Line 5 always returns an x_{i-1} with $c(x - x_{i-1}) \geq \Phi > [1 - (1 - 1/k)^N]\Phi$. When Line 9 is executed, the augmentation oracle just returned $x_i = x_{i-1}$, i.e., for all $y \in P$

$$cx_{i-1} \leq cy + \frac{\Phi - c(x - x_{i-1})}{k}\|y - x_{i-1}\|_1 \leq cy + \frac{\Phi - c(x - x_{i-1})}{k}k = c(y - x) + cx_{i-1} + \Phi,$$

so that $c(x - y) \leq \Phi$, as claimed.

Finally, when Line 12 is executed, the augmentation oracle has found an improving vertex x_i at every iteration, i.e.,

$$cx_{i-1} > cx_i + \frac{\Phi - c(x - x_{i-1})}{k}\|x_i - x_{i-1}\|_1 \geq cx_i + \frac{\Phi - c(x - x_{i-1})}{k},$$

using $\|x_i - x_{i-1}\|_1 \geq 1$ by integrality. Rearranging provides the convenient form

$$\Phi - c(x - x_i) < \left(1 - \frac{1}{k}\right)[\Phi - c(x - x_{i-1})],$$

which by an easy induction provides

$$\Phi - c(x - x_N) < \left(1 - \frac{1}{k}\right)^N [\Phi - c(x - x_0)] \leq \left(1 - \frac{1}{K}\right)\Phi,$$

i.e., $c(x - x_N) \geq \frac{\Phi}{K}$, finishing the proof. ■

8. Experiments

We implemented and compared the parameter-free variant of LCG (Algorithm 7) to the standard Frank–Wolfe algorithm (CG), then Algorithm 4 (LPCG) to the Pairwise Conditional

Gradient algorithm (PCG) of Garber and Meshi (2016), as well as Algorithm 8 (LOCG) to the Online Frank–Wolfe algorithm (OCG) of Hazan and Kale (2012). While we did implement the Local Conditional Gradient algorithm of Garber and Hazan (2013) as well, the very large constants in the original algorithms made it impractical to run. Unless stated otherwise the weak separation oracle is implemented as sketched in Algorithm 2 through caching and early termination of the original LP oracle.

We have used $K = 1.1$ and $K = 1$ as multiplicative factors for the weak separation oracle; for the impact of the choice of K see Section 8.2.2. For the baseline algorithms we use inexact variants, i.e., we solve linear optimization problems only approximately. This is a significant speedup in favor of non-lazy algorithms at the (potential) cost of accuracy, while neutral to lazy optimization as it solves an even more relaxed problem anyways. To put things in perspective, the non-lazy baselines could not complete even a single iteration for a significant fraction of the considered problems in the given time frame if we were to exactly solve the linear optimization problems. In terms of using line search, for all tests we treated all algorithms equally: either *all* or *none* used line search. If not stated otherwise, we used (simple backtracking) line search.

The linear optimization oracle over $P \times P$ for LPCG was implemented by calling the respective oracle over P twice: once for either component. Contrary to the non-lazy version, the lazy algorithms depend on the initial upper bound Φ_0. For the instances that need a very long time to solve the (approximate) linear optimization even once, we used a binary search for Φ_0 for the lazy algorithms: starting from a conservative initial value, using the update rule $\Phi_0 \leftarrow \Phi_0/2$ until the separation oracle returns an improvement for the first time and then we start the algorithm with $2\Phi_0$, which is an upper bound on the Wolfe gap and hence also on the primal gap. This initial phase is also included in the reported wall-clock time. Alternatively, if the linear optimization was less time consuming we used a single (approximate) linear optimization at the start to obtain an initial bound on Φ_0 (see e.g., Section 4).

In some cases, especially when the underlying feasible region has a high dimension and the (approximate) linear optimization can be solved relatively fast compared to the cost of computing an inner product, we observed that the costs of maintaining the cache was very high. In these cases we reduced the cache size every 100 steps by keeping only the 100 points that were used the most so far. Both the number of steps and the approximate size of the cache were chosen arbitrarily, however 100 for both worked very well for all our examples. Of course there are many different strategies for maintaining the cache, which could be used here and which could lead to further improvements in performance.

The stopping criteria for each of the experiments was a given wall clock time limit in seconds. The time limit was enforced separately for the main code and the oracle code, so in some cases the actual time used can be larger, when the last oracle call started before the time limit was reached and took longer than the time left.

We implemented all algorithms in Python 2.7 with critical functions *cythonized* for performance employing Numpy. We used these packages from the Anaconda 4.2.0 distribution as well as Gurobi 7.0 Gurobi Optimization (2016) as a black box solver for the linear optimization oracle. The weak separation oracle was implemented via a callback function to stop linear optimization as soon as a good enough feasible solution has been found in a schema as outlined in Algorithm 2. The parameters for Gurobi were kept at their

default settings except for enforcing the time limit of the tests and setting the acceptable duality gap to 10%, allowing Gurobi to terminate the linear optimization early avoiding the expensive *proof* of optimality. This is used to realize the inexact versions of the baseline algorithms. All experiments were performed on a 16-core machine with Intel Xeon E5-2630 v3 @ 2.40GHz CPUs and 128GB of main memory. While our code does not explicitly use multiple threads, both Gurobi and the numerical libraries use multiple threads internally.

8.1. Computational results

We performed computational tests on a large variety of different problems that are instances of the three machine learning tasks *video colocalization*, *matrix completion*, and *structured regression*.

Video colocalization. Video colocalization is the problem of identifying objects in a sequence of multiple frames in a video. In Joulin et al. (2014) it is shown that video colocalization can be reduced to optimizing a quadratic objective function over a flow or a path polytope, which is the problem we are going to solve. The resulting linear program is an instance of the minimum-cost network flow problem, see (Joulin et al., 2014, Eq. (3)) for the concrete linear program and more details. The quadratic functions are of the form $\|Ax - b\|^2$ where we choose the non-zero entries in A according to a density parameter at random and then each of these entries to be $[0, 1]$-uniformly distributed, while b is chosen as a linear combination of the columns of A with random multipliers from $[0, 1]$. For some of the instances we also use $\|x - b\|^2$ as the objective function with $b_i \in [0, 1]$ uniformly at random.

Matrix completion. The formulation of the matrix completion problem we are going to use is the following:

$$\min_X \sum_{(i,j)\in\Omega} |X_{i,j} - A_{i,j}|^2 \qquad \text{s.t.} \quad \|X\|_* \leq R, \tag{6}$$

where $\|\cdot\|_*$ denotes the nuclear norm, i.e., $\|A\|_* = \operatorname{Tr}(\sqrt{A^t A})$. The set Ω, the matrix A and R are given parameters. Similarly to Lan and Zhou (2014) we generate the $m \times n$ matrix A as the product of A_L of size $m \times r$ and A_R of size $r \times n$. The entries in A_L and A_R are chosen from a standard Gaussian distribution. The set Ω is chosen uniformly of size $s = \min\{5r(m+n-r), \lceil 0.99mn \rceil\}$. The linear optimization oracle is implemented in this case by a singular value decomposition of the linear objective function and we essentially solve the LP to (approximate) optimality. The matrix completion tests will only demonstrate the impact of caching solutions. Note that this test is also informative as due to the 'roundness' of the feasible region the solution of the actual LP oracle will induce a direction that is equal to the true gradient and as such it provides insight into how much per-iteration progress is lost due to working with gradient approximations from the weak separation oracle.

Structured regression. The structured regression problem consists of solving a quadratic function of the form $\|Ax - b\|^2$ over some structured feasible set or a polytope P, i.e., we solve $\min_{x\in P}\|Ax - b\|^2$. We construct the objective functions in the same way as for the video colocalization problem.

Tests. In the following two sections we will present our results for various problems grouped by the versions of the considered algorithms. Every figure contains two columns, each containing one experiment. We use different measures to report performance: we report progress of loss or function value in wall-clock time in the first row (including time spent by the oracle), in the number of iterations in the second row, and in the number of linear optimization calls in the last row. Obviously, the latter only makes sense for the lazy algorithms. In some other cases we report in another row the dual bound or Wolfe gap in wall-clock time. The red line denotes the non-lazy algorithm and the green line denotes the lazy variants. For each experiment we also report the cache hit rate, which is the number of oracle calls answered with a point from the cache over all oracle calls given in percent.

While we found convergence rates in the number of iterations quite similar (as expected!), we consistently observe a significant speedup in wall-clock time. In particular for many large-scale or hard combinatorial problems, lazy algorithms performed several thousand iterations whereas the non-lazy versions completed only a handful of iterations due to the large time spent approximately solving the linear optimization problem. The observed cache hit rate was at least 90% in most cases, and often even above 99%.

Compared to the non-lazy variants, the lazy variants might use weaker descent directions (due to employing the weak-separation oracle instead of the LP oracle), and hence one expects that the lazy algorithms in general require more iterations despite being faster in wall-clock time. However, the lazy and non-lazy algorithms usually generate different sequences of iterates, and hence the lazy algorithms may converge faster even in the number of iterations by chance; this is even expected in a small number of cases by the law of large numbers. This happens for example in Figures 3 and 14.

8.1.1. Offline Results

We describe the considered instances in the offline case separately for the vanilla Frank–Wolfe method and the Pairwise Conditional Gradient method.

Vanilla Frank–Wolfe Method We tested the vanilla Frank–Wolfe algorithm on the six video colocalization instances with underlying path polytopes from `http://lime.cs.elte.hu/~kpeter/data/mcf/netgen/` (Figure 1). In these instances we additionally report the dual bound or Wolfe gap in wall clock time. We further tested the vanilla Frank–Wolfe algorithm on eight instances of the matrix completion problem generated as described above, for which we did not use line search; the parameter-free lazy variant is run with approximate minimization as described in Remark 11, the others use their respective standard step sizes. We provide the used parameters for each example in the figures below (Figures 2 and 3). The last tests for this version were performed on three instances of the structured regression problem, two with the feasible region containing flow-based formulations of Hamiltonian cycles in graphs (Figure 4), and further tests on two spanning tree instances of different size (Figure 5).

We observed a significant speedup of LCG compared to CG, due to the faster iteration of the lazy algorithm.

Pairwise Conditional Gradient Algorithm As we inherit structural restrictions of PCG on the feasible region, the problem repertoire is limited in this case. We tested the

Pairwise Conditional Gradient algorithm on the structured regression problem with feasible regions from the MIPLIB instances `eil33-2`, `air04` (Figure 6).

Again similarly to the vanilla Frank–Wolfe algorithm, we observed a significant improvement in wall-clock time of LPCG compared to CG, due to the faster iteration of the lazy algorithm.

8.1.2. Online Results

Additionally to the quadratic objective functions above we tested the online version on random linear functions $cx + b$ with $c \in [-1, +1]^n$ and $b \in [0, 1]$. For online algorithms, each experiment used a random sequence of 100 different random loss functions. In every figure the left column uses linear loss functions, while the right one uses quadratic loss functions over the same polytope. As customary, we did not use line search here but used the respective prescribed step sizes.

As an instance of the structured regression problem we used the standard formulation of the cut polytope for graphs with 28 nodes as the feasible region (Figure 7). We also tested our algorithm on the quadratic unconstrained boolean optimization (QUBO) instances defined on Chimera graphs Dash (2013), which are available at `http://researcher.watson.ibm.com/researcher/files/us-sanjeebd/chimera-data.zip`. The instances are relatively hard albeit their rather small size and in general the problem is NP-hard. (Figure 8).

One instance of the video colocalization problem uses a path polytope from `http://lime.cs.elte.hu/~kpeter/data/mcf/netgen/` that was generated with the `netgen` graph generator (Figure 9). Most of these instances are very large-scale minimum cost flow instances with several tens of thousands nodes in the underlying graphs, therefore solving still takes considerable time despite the problem being in P. Finally, for the spanning tree problem, we used the well-known extended formulation with $O(n^3)$ inequalities for an n-node graph. We considered graphs with 25 nodes (Figures 10).

We observed that similarly to the offline case while OCG and LOCG converge comparably in the number of iterations, the lazy LOCG performed significantly more iterations; for hard problems, where linear optimization is costly and convergence requires a large number of iterations, this led LOCG converging much faster in wall-clock time. In extreme cases OCG could not complete even a single iteration. This is due to LOCG only requiring *some* good enough solution, whereas OCG requires a stronger guarantee. This is reflected in faster oracle calls for LOCG.

Weak-Separation via Augmentation As discussed in Section 7 in some cases it can be very beneficial to realize the weak-separation oracle by means of augmentation instead of linear optimization. To verify this we implemented a weak-separation oracle via an augmentation oracle for quadratic unconstrained boolean optimization (QUBO) instances (see above for details). For those instances, the primal heuristics of Gurobi (or CPLEX) can find improving solutions very fast due to the structure of the instances. We obtain the augmentation oracle then by exiting the Gurobi call as soon as the first improving solution is found. For comparability, apart from using the augmentation oracle we used the same configuration as above. The results can be found in Figure 11, where we can observe a significant speedup of the lazified variant of OCG using augmentation over the base OCG algorithm. Also observe that this is in contrast to Figure 8, where the advantage of LOCG

(without augmentation) over OCG was not that pronounced. To provide further insight in this case we also report *oracle time*, which denotes actual time spent in the augmentation oracle, which is minimal in both cases as can be seen.

8.2. Performance improvements, parameter sensitivity, and tuning

8.2.1. Effect of caching

As mentioned before, lazy algorithms have two improvements: caching and early termination. Here we depict the effect of caching in Figure 12, comparing OCG (no caching, no early termination), LOCG (caching and early termination) and LOCG (only early termination) (see Algorithm 8). 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.

8.2.2. 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 13.

8.2.3. Parameter-free vs. textbook variant

For illustrative purposes, we compare the textbook variant of the lazy conditional gradient (Algorithm 3) with its parameter-free counterpart (Algorithm 7) in Figure 14. The parameter-free variant outperforms the textbook variant due to the active management of Φ combined with line search.

Similar parameter-free variants can be derived for the other algorithms; see discussion in Section 4.

9. Final Remarks

As discussed above in Section 6, if a given baseline algorithm works over general compact convex sets P, then so does the lazified version. In fact, as the lazified algorithm runs, it produces a polyhedral approximation of the set P with very few vertices (subject to optimality vs. sparsity tradeoffs; see (Jaggi, 2013, Appendix C)).

Moreover, the weak separation oracle does not need to return extreme points. All algorithms also work with maximal solutions that are not necessarily extremal (e.g., lying in a higher-dimensional face). However, in that case we lose the desirable property that the final solution is a sparse convex combination of extreme points (typically vertices in the polyhedral setup).

We would also like to briefly address potential downsides of our approach. In fact, we believe the right perspective is the following: when using the lazy oracle over the LP oracle, we obtain potentially *weaker* approximations $v_t - x_t$ of the true gradient $\nabla f(x_t)$ compared

to solving the actual LP, but the computation might be much faster. This is the tradeoff that one has to consider: working with weaker approximations (which implies potentially less progress per iteration) vs. potentially significantly faster computation of the approximations. If solving the LP is expensive, then lazification will be usually very beneficial, if the LP is very cheap as in the case of $P = [0, 1]^n$ or $P = \Delta_n$ being the probability simplex, then lazification might be slower.

A related remark in this context is that once the lazified algorithm has obtained vertices $x_1, \ldots, x_m$ of P, so that the minimizer x^* of f satisfies $x^* \in \operatorname{conv}\{x_1, \ldots, x_m\}$, then from that point onwards no actual calls to the true LP oracle have to be performed anymore for primal progress and the algorithm will only use cache calls; the only remaining true LP calls are at most a logarithmic number for dual progress updates of the Φ_t.

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

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

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

G. Braun, S. Pokutta, and D. Zink. Lazifying Conditional Gradient Algorithms. *Proceedings of ICML*, 2017.

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

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

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

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

Robert M. Freund and Paul Grigas. 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`.

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

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

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

Swati Gupta, Michel Goemans, and Patrick Jaillet. 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/`.

Elad Hazan. 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/`.

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

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

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

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

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

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

Simon Lacoste-Julien, Martin Jaggi, Mark Schmidt, and Patrick Pletscher. Block-coordinate Frank–Wolfe optimization for structural SVMs. In *ICML 2013 International Conference on Machine Learning*, pages 53–61, 2013.

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

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

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

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

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

Sebastian Pokutta. Smooth convex optimization via geometric scaling. *preprint*, 2017.

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

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

Neel Shah, Vladimir Kolmogorov, and Christoph H Lampert. 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*, pages 2737–2745, 2015.

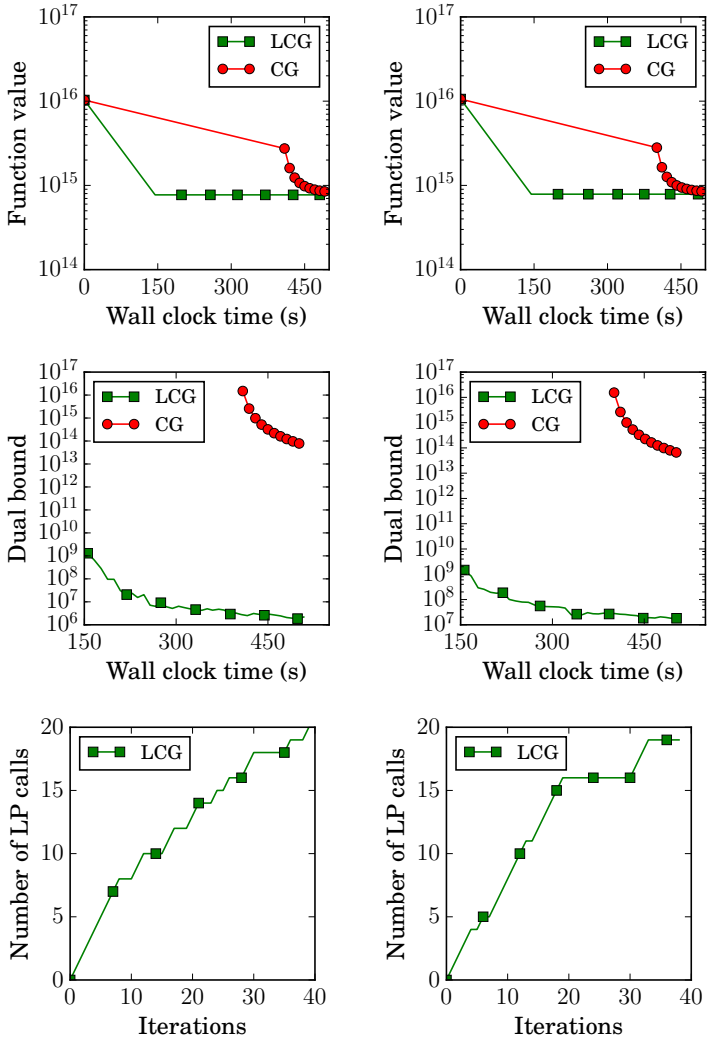

Figure 1: LCG vs. CG on large netgen instances `netgen 16a` (left) and `netgen 16b` (right) with quadratic objective functions. In both cases the difference in function value between the two versions of the algorithm is large. In the dual bound the performance of the lazy version is multiple orders of magnitude better than the performance of the non-lazy counterpart. The cache hit rates for these two instances are lower due to the high dimension of the polytope.

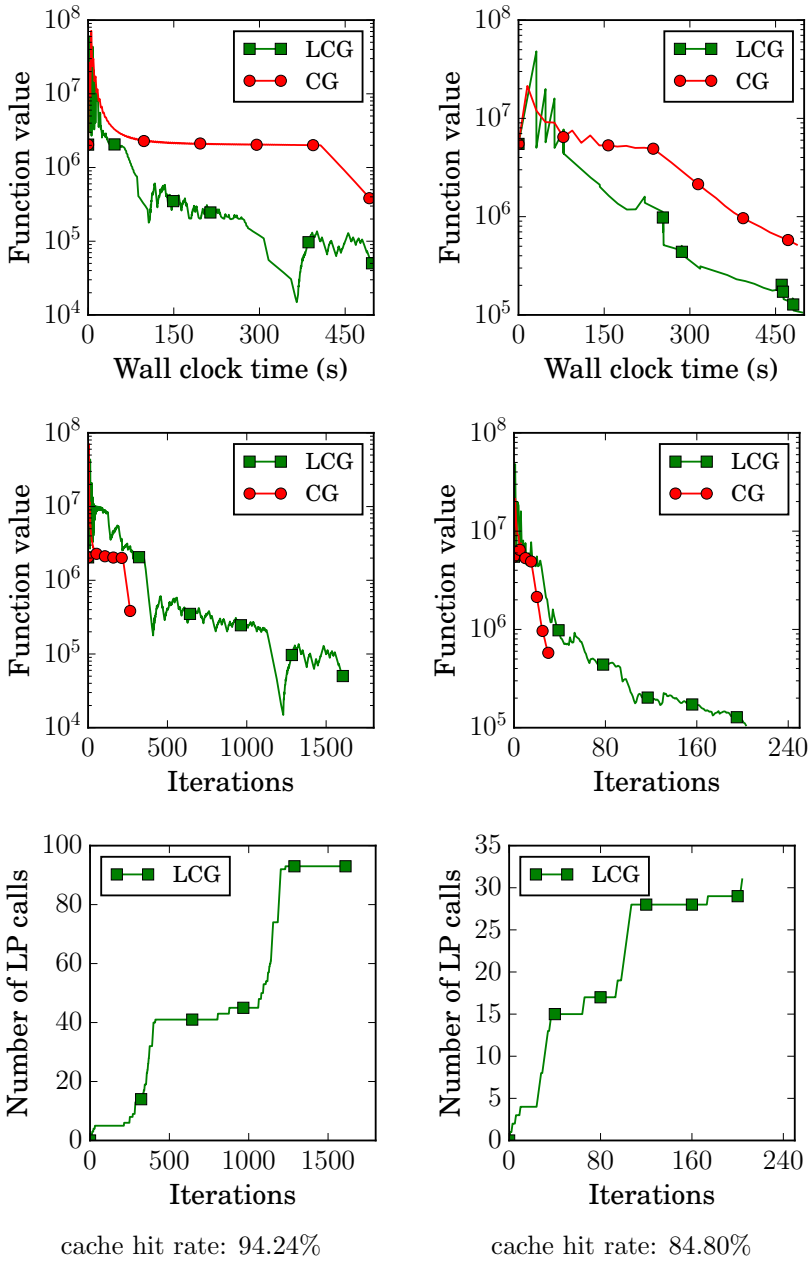

Figure 2: LCG vs. CG on two matrix completion instances. We solve the problem as given in Equation (6) with the parameters $n = 3000$, $m = 1000$, $r = 10$ and $R = 30000$ for the left instance and $n = 10000$, $m = 100$, $r = 10$ and $R = 10000$ for the right instance. In both cases the lazy version is slower in iterations, however significantly faster in wall clock time. Note that we used the short-step rule for step sizes for both algorithms as line search for matrix completion is very expensive.

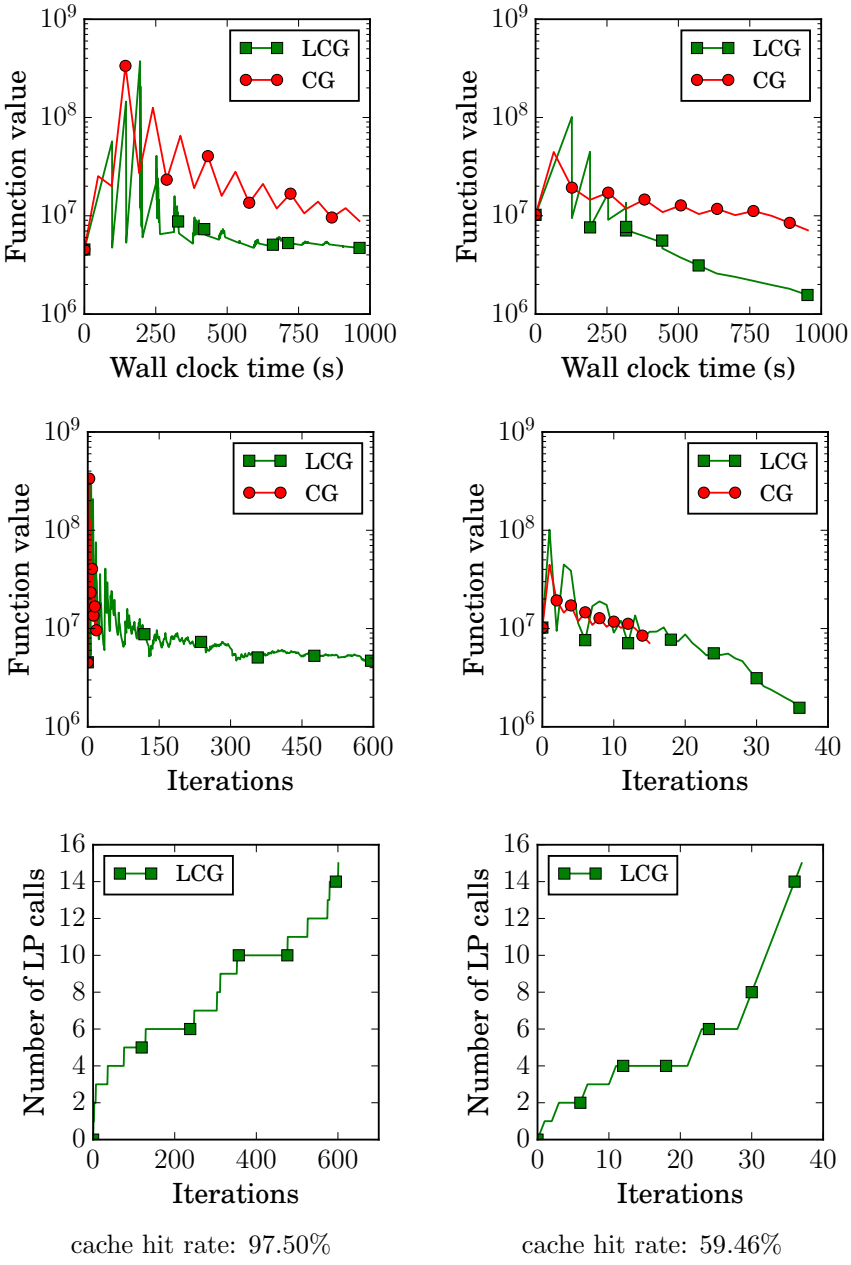

Figure 3: LCG vs. CG on two matrix completion instances. The parameters for Equation (6) are given by $n = 5000$, $m = 4000$, $r = 10$ and $R = 50000$ for the left instance and $n = 100$, $m = 20000$, $r = 10$ and $R = 15000$ for the right instance. In both of these cases the performance of the lazy and the non-lazy version are comparable in iterations, however in wall clock time the lazy version reaches lower function values faster. Note that we used the short-step rule for step sizes for both algorithms as line search for matrix completion is very expensive.

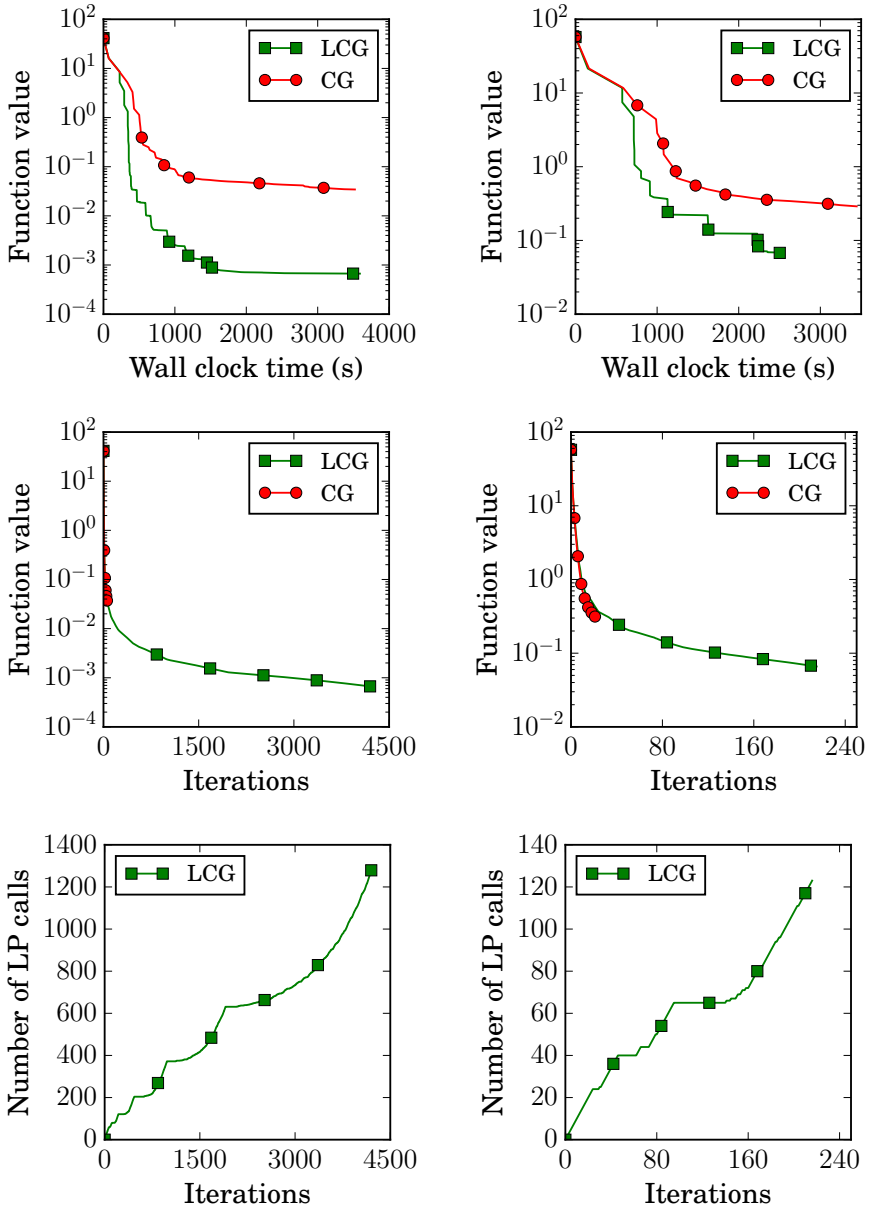

Figure 4: LCG vs. CG on structured regression problems with feasible regions being a TSP polytope over 11 nodes (left) and 12 nodes (right). In both cases LCG is significantly faster in wall-clock time.

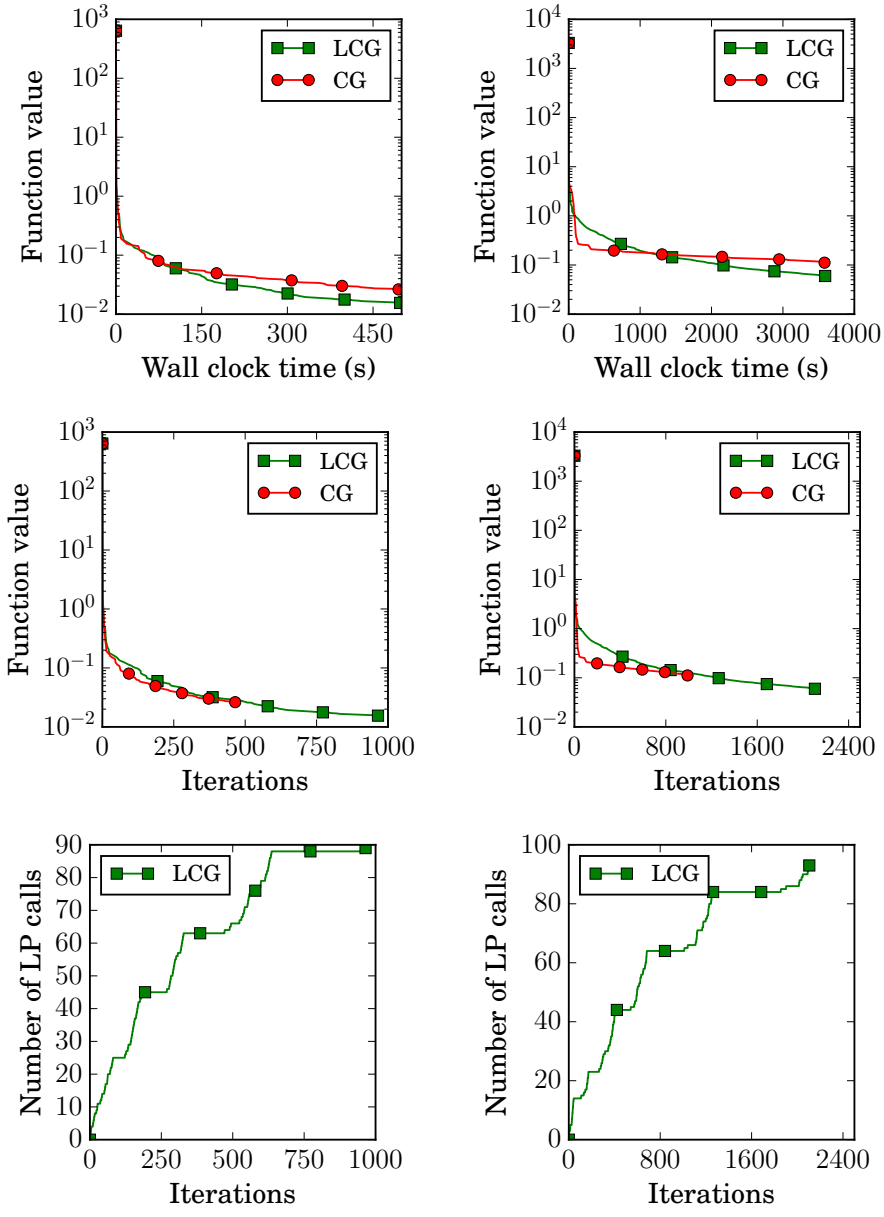

Figure 5: LCG vs. CG on structured regression instances with extended formulation of the spanning tree problem on a 10 node graph on the left and a 15 node graph on the right.

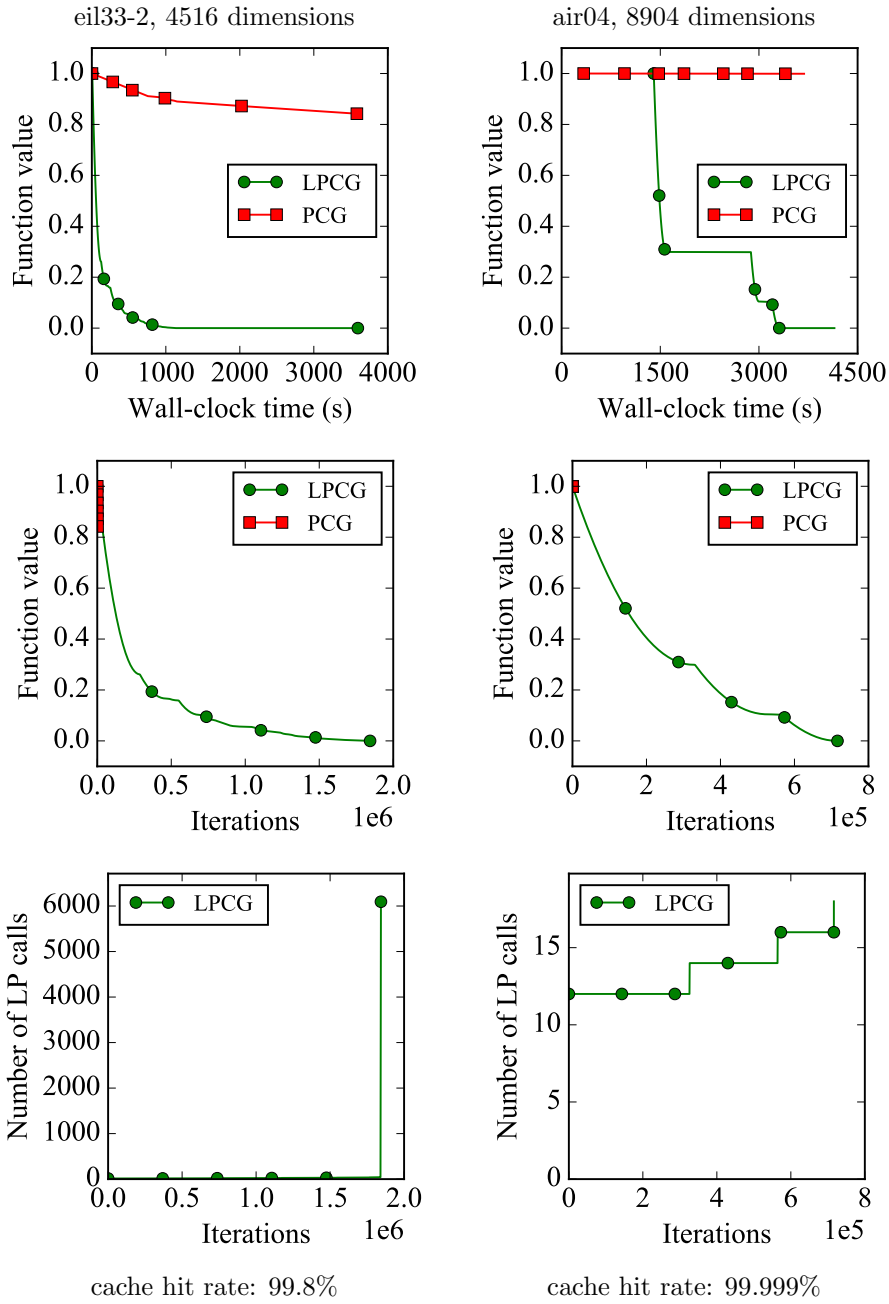

Figure 6: LPCG vs. PCG on two MIPLIB instances `eil33-2` and `air04`. LPCG converges very fast, making millions of iterations with a relatively few oracle calls, while PCG completed only comparably few iterations due to the time-consuming oracle calls. This clearly illustrates the advantage of lazy methods when the cost of linear optimization is non-negligible. On the left, when reaching ε-optimality, LPCG performs many (negative) oracle calls to (re-)prove optimality; at that point one might opt for stopping the algorithm. On the right LPCG needed a rather long time for the initial bound tightening of Φ_0, before converging significantly faster than PCG.

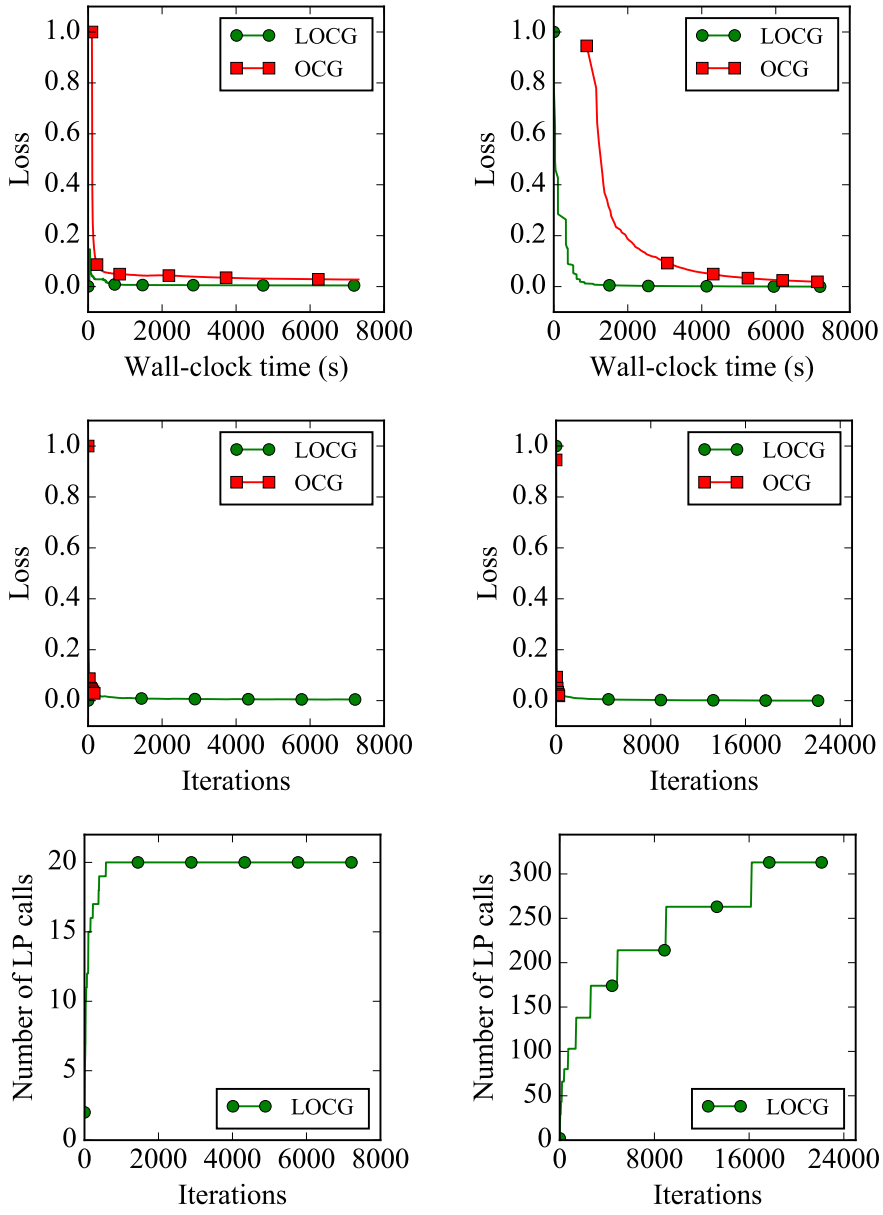

Figure 7: LOCG vs. OCG over cut polytope for a 28-node graph. As for the smaller problem, this also illustrates the advantage of lazy algorithms when linear optimization is expensive. Again, LOCG needed no oracle calls after a small initial amount of time.

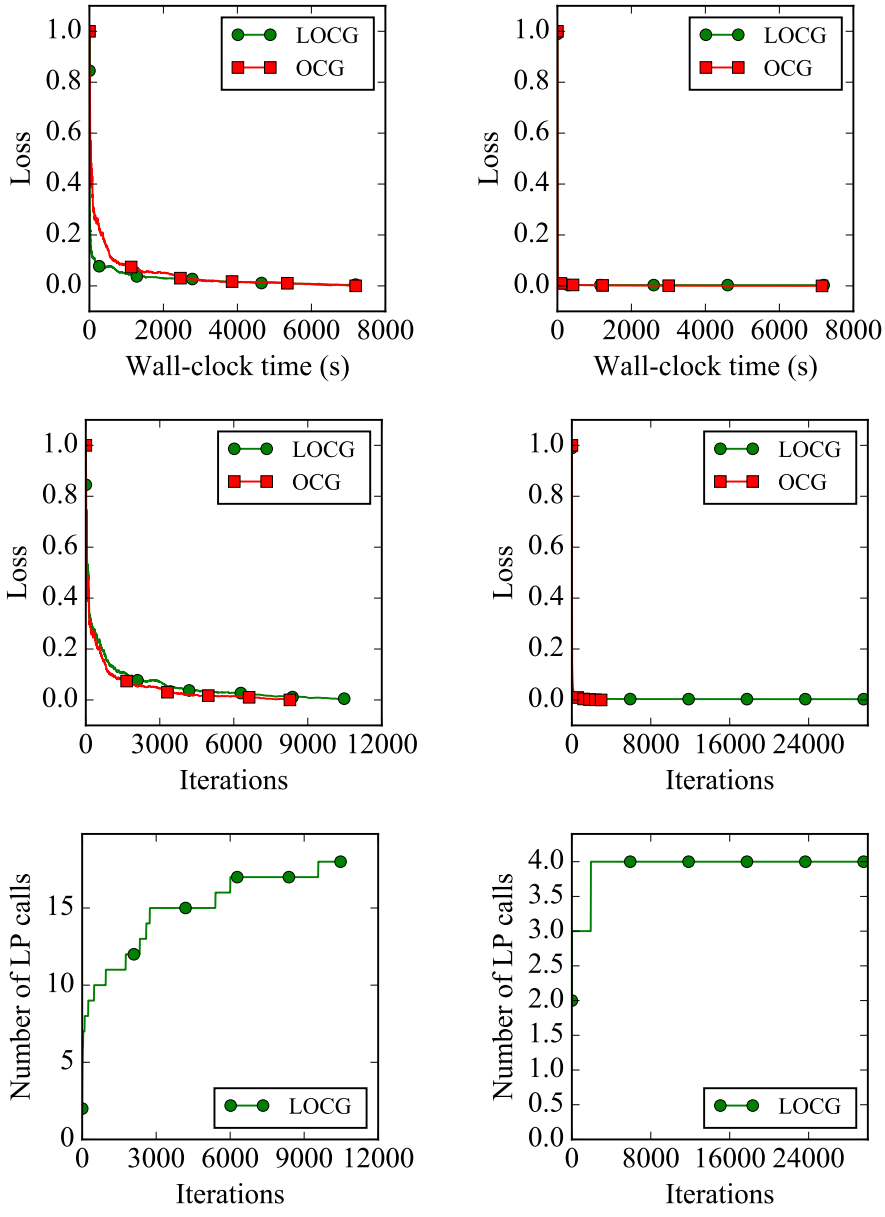

Figure 8: LOCG vs. OCG on a large QUBO instance. Both algorithms converge fast to the optimum. Interestingly, LOCG only performs 4 LP calls.

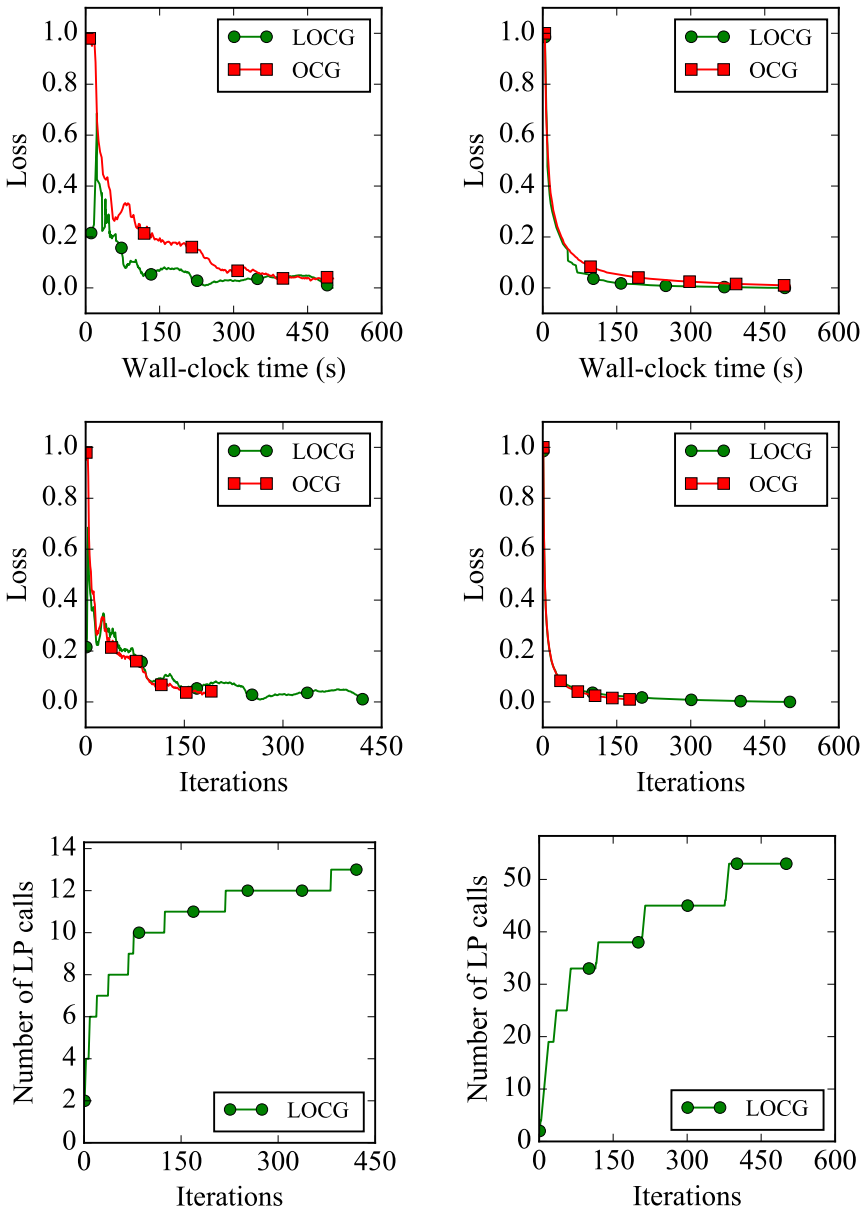

Figure 9: LOCG vs. OCG over a path polytope. Similar convergence rate in the number of iterations, but significant difference in terms of wall-clock time.

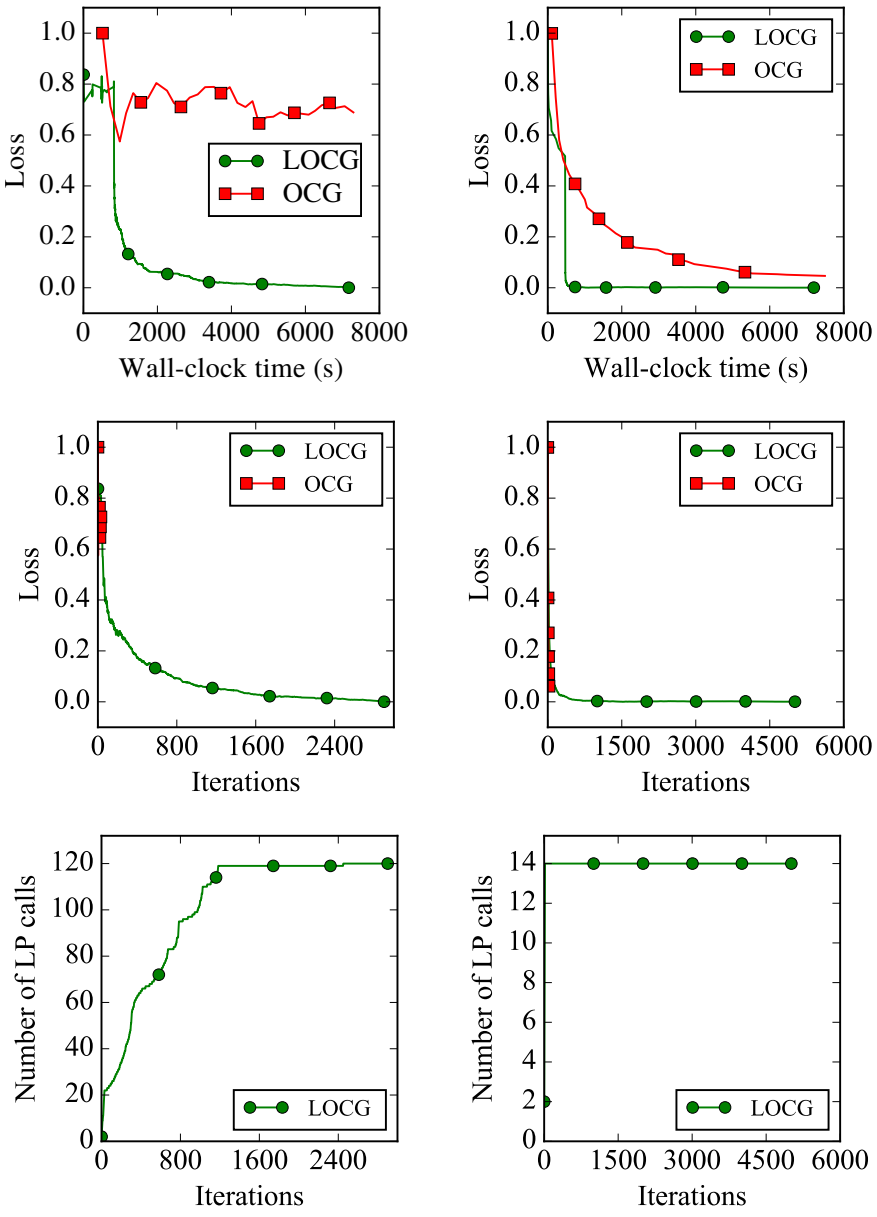

Figure 10: LOCG vs. OCG over a spanning tree instance for a 25-node graph. On the left, early fluctuation can be observed, bearing no consequence for later convergence rate. OCG did not get past this early stage. In both cases LOCG converges significantly faster.

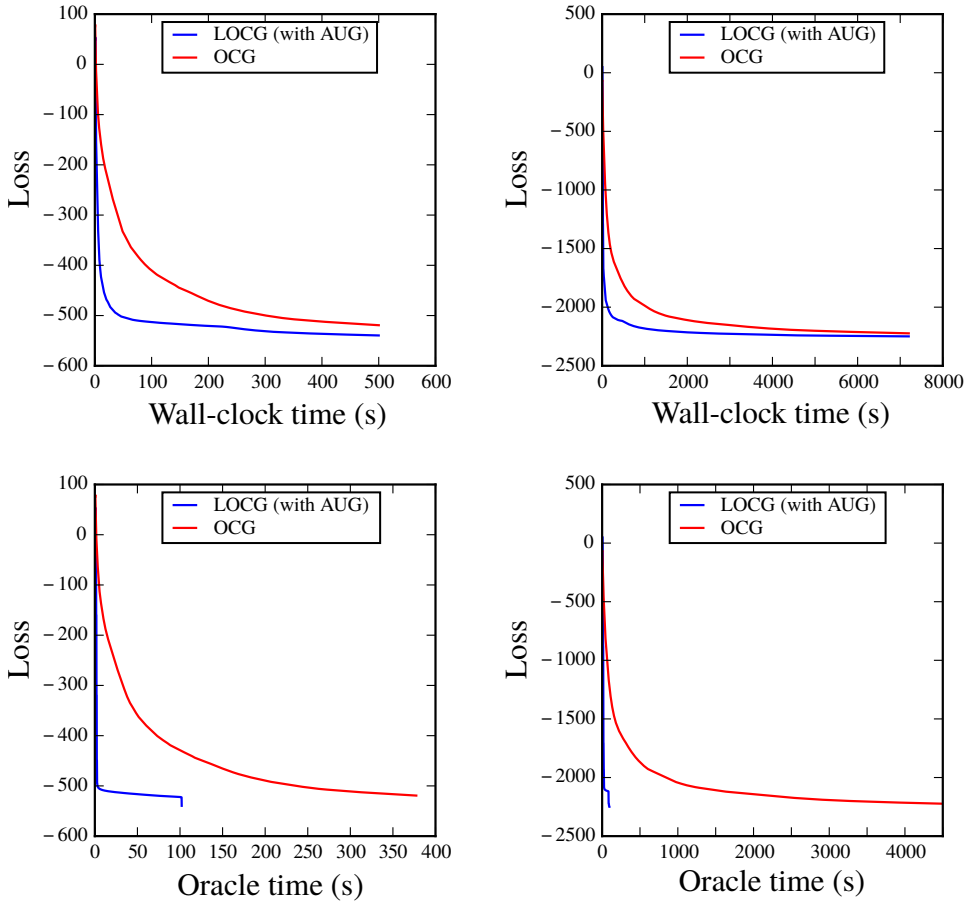

Figure 11: Lazy OCG (with augmentation) vs. OCG on two QUBO instances. In both cases the lazy variant together with augmentation significantly outperforms OCG.

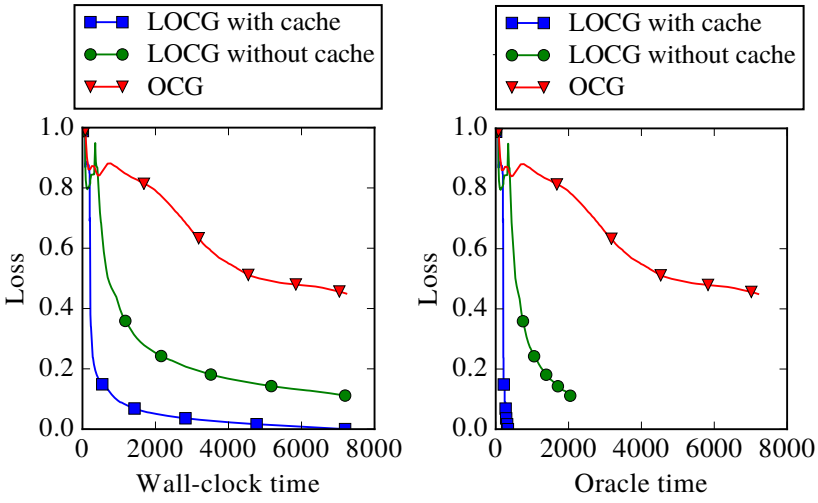

Figure 12: Performance gain due to caching and early termination for online optimization over a maximum cut problem with linear losses. The red line is the OCG baseline, the green one is the lazy variant using only early termination, and the blue one uses caching and early termination. Left: loss vs. 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.

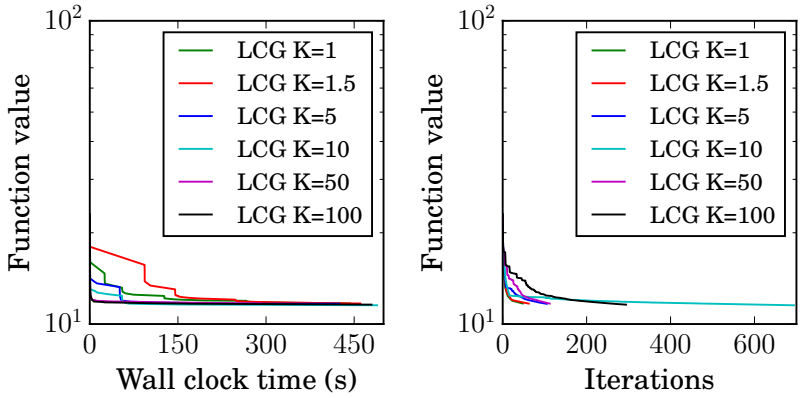

Figure 13: Impact of the oracle approximation parameter K depicted for the Lazy CG algorithm. We can see that increasing K leads to a deterioration of progress in iterations but improves performance in wall-clock time. The behavior is similar for other algorithms.

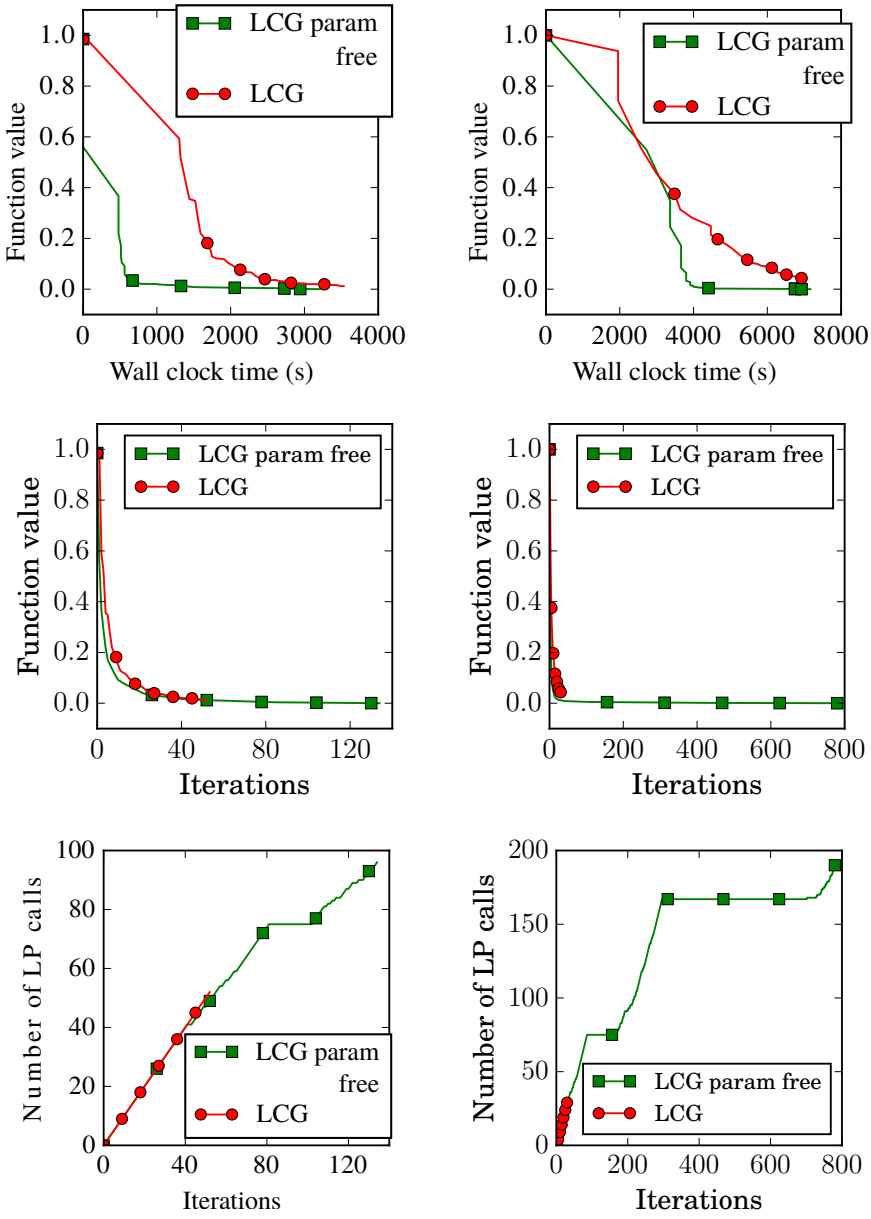

Figure 14: Comparison of the 'textbook' variant of the Lazy CG algorithm (Algorithm 3) vs. the Parameter-free Lazy CG (Algorithm 7) depicted for two sample instances to demonstrate behavior. The parameter-free variant usually has a slightly improved behavior in terms of iterations and a significantly improved behavior in terms of wall-clock performance. In particular, the parameter-free variant can execute significantly more oracle calls, due to the Φ-halving strategy and the associated bounded number of negative calls (see Theorem 9).