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
for $t = 1$ **to** $T - 1$ **do**
  $v_t \leftarrow \mathrm{LP}_P(\nabla f(x_t))$
  $x_{t+1} \leftarrow (1 - \gamma_t)x_t + \gamma_t v_t$ with $\gamma_t := \frac{2}{t+2}$
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** $y \leftarrow \mathrm{argmax}_{x \in P}\, c(x)$ {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 $\mathbb{B}_r(x) := \{y \mid \|x - y\| \leq r\}$ 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: $\quad v_t \leftarrow \mathrm{LPsep}_P(\nabla f(x_t), x_t, \Phi_t, K)$
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 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 $g(x) := \max_{v \in P} \nabla f(x)(x - v)$ 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** $t = 1, \dots, T$ **do**

3: define $\tilde{\nabla} f(x_t) \in \mathbb{R}^m$ as follows:

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

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 $M_1 := \sqrt{\frac{S}{8 \operatorname{card}(x^*)}}$, $M_2 := KC/2$, $\kappa := \min\{\frac{M_1}{2M_2}, 1/\sqrt{\Phi_0}\}$, $\eta_t := \kappa\sqrt{\Phi_{t-1}}$ and $\Delta_t := \sqrt{\frac{2 \operatorname{card}(x^*)\Phi_{t-1}}{S}}$, 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 $B := \kappa \cdot \frac{M_1}{2K}$.

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: $\quad v_t \leftarrow \mathrm{LPsep}_P(\nabla f(x_t), x_t, \Phi_{t-1}, K)$
4: $\quad$ **if** $v_t =$ **false then**
5: $\quad\quad x_{t+1} \leftarrow x_t$
6: $\quad\quad \Phi_t \leftarrow \frac{\Phi_{t-1}}{2}$ {Update Φ}
7: $\quad$ **else**
8: $\quad\quad \gamma_t \leftarrow \operatorname{argmin}_{0 \leq \gamma \leq 1} f((1-\gamma)x_t + \gamma v_t)$
9: $\quad\quad x_{t+1} \leftarrow (1 - \gamma_t)x_t + \gamma_t v_t$ {Update iterate}
10: $\quad\quad \Phi_t \leftarrow \Phi_{t-1}$
11: $\quad$ **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*

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

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.

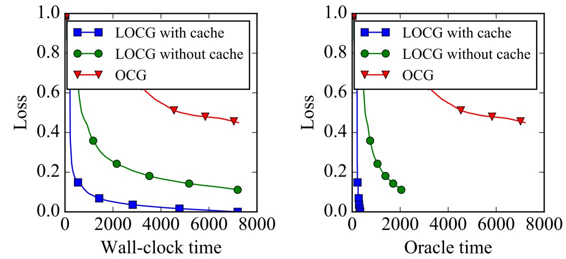


Figure 1. Performance gain due to caching and early termination for stochastic optimization over a maximum cut problem with linear losses. The red line is the OCG baseline, the green one is the lazy variant using only early termination, and the blue one uses caching and early termination. Left: loss 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.

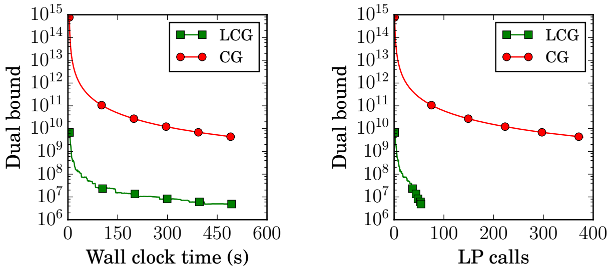


Figure 2. Performance on an instance of the video colocalization problem. We solve quadratic minimization over a flow polytope and report the achieved dual bound (or Wolfe-gap) over wall-clock time in seconds in logscale on the left and over the number of actual LP calls on the right. We used the parameter-free 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.

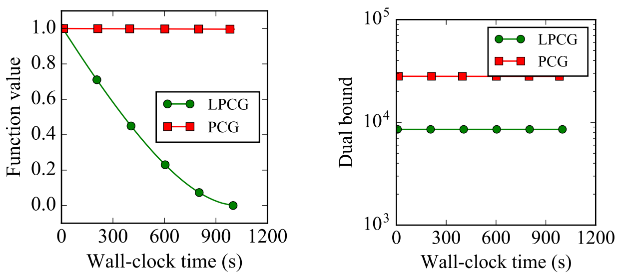


Figure 3. Performance on a large instance of the video colocalization problem using PCG and its lazy variant. We observe that lazy PCG is significantly better both in terms of function value and dual bound. Recall that the function value is normalized between $[0, 1]$.

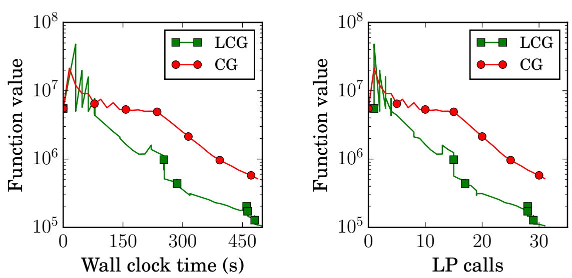


Figure 4. Performance on a matrix completion instance. More information about this problem can be found in the supplemental material (Section E). The performance is reported as the objective function value over wall-clock time in seconds on the left and over LP calls on the right. In both measures after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

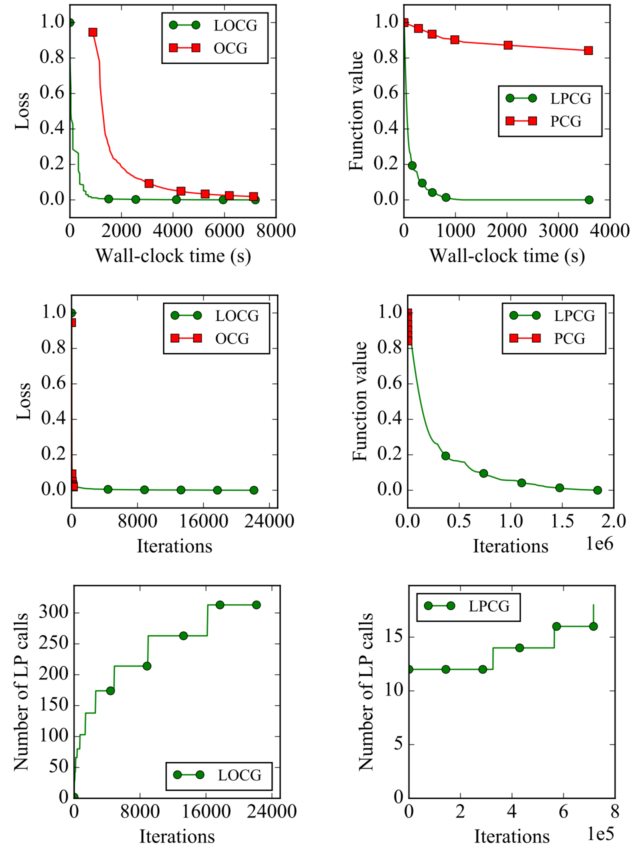


Figure 5. Performance of the two lazified variants LOCG (left column) and LPCG (right column). The feasible regions are a cut polytope on the left and the MIPLIB instance `air04` on the right. The objective functions are in both cases quadratic, on the left randomly chosen in every step. We show the 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.

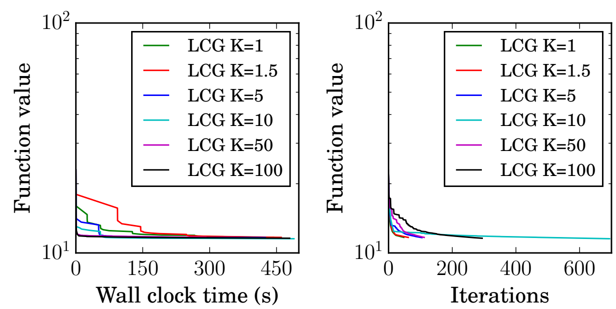


Figure 6. Impact of the oracle approximation parameter K depicted for the Lazy CG algorithm. We can see that increasing K leads to a deterioration of progress in iterations but improves performance in wall-clock time. The behavior is similar for other algorithms.

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, Lászlo. 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.

A. Implementation Improvements

Note that there are various obvious improvements to Algorithm 2 for actual implementations. These improvements do not affect the theoretical (worst-case) performance and for the sake of clarity of the exposition we did not include them in Algorithm 2.

(i) First of all, we can improve the update of Φ_t, using the actual gap closed, rather than the pessimistic update via the lower bound on gap closed, i.e., we can update $\Phi_t \leftarrow \Phi_t - (f(x_t) - f(x_{t+1}))$, whenever we calculated a new point x_{t+1}.

(ii) Moreover, we can better utilize information from negative oracle calls (i.e., when the oracle returns **false**): if the oracle utilizes linear optimization at its core, then a negative oracle call will certify $\nabla f(x_t)(x_t - v) \leq \Phi_t$ via maximizing $\nabla f(x_t)v$ with $v \in P$, i.e., the linear optimization oracle computes $g(x_t)$ and we can reset $\Phi_t \leftarrow g(x_t)$. If v^* realizes the Wolfe gap, which is obtained as a byproduct of the above linear maximization, we can further use v^* to make a step: rather than executing line 5, we can execute line 7 with $v_t = v^*$. By doing so we maximize the use of information obtained from a negative oracle call.

(iii) Finally, we can optimize the management of Φ_t. To obtain a better upper bound Φ_0, we can solve $v^* := \operatorname{argmax}_{v \in P} \nabla f(x_1)v$ at the expense of one extra LP call and set $\Phi_0 := \nabla f(x_1)(x_1 - v^*) = g(x_1)$. Alternatively, we can perform binary search over Φ_0 until the weak separation oracle produces an actual step. If $\bar{\Phi}$ is the value of the search for which we observe the first step, we can reset $\Phi_0 := 2\bar{\Phi}$ and we have $f(x_1) - f(x^*) \leq g(x_1) \leq 2\bar{\Phi}$.

Furthermore, we can change the strategy for managing Φ_t as follows: we keep the value of Φ_t fixed in line 2 and perform line search for γ. Whenever, we observe a negative oracle call, we set the current Φ_t to $\frac{1}{2}g(x_t)$ obtained from the negative call. As such, we ensure $\Phi_t < g(x_t) \leq 2\Phi_t$, which biases the algorithm towards (much cheaper) positive calls. Convergence is ensured by observing that an $\mathrm{LPsep}_P(\cdot, \cdot, \Phi/2, K)$ oracle is an $\mathrm{LPsep}_P(\cdot, \cdot, \Phi, 2K)$ oracle for which the theorem directly applies. With this strategy we maintain the same theoretical (worst-case) convergence up to a constant factor, however in case a faster convergence is possible, we adapt to that rate. The parameter-free version in Section 4 utilizes this technique.

B. Lazy Local Conditional Gradients

In this section we provide a lazy version (Algorithm 5) of the conditional gradient algorithm from (Garber & 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 the affine invariant of P from (Garber & Hazan, 2013). 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 Gradients (LLCG)

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

$x_1 \in P$ arbitrary and $\Phi_0 \geq f(x_1) - f(x^*)$
$\alpha \leftarrow \frac{S}{2K\beta n\mu^2}$
for $t = 1, \ldots, T$ **do**
  $\Phi_t \leftarrow \frac{\Phi_{t-1} + \frac{\beta}{2}\alpha^2 \min\{n\mu^2 r_t^2, D^2\}}{1 + \alpha/K}$
  $r_t \leftarrow \sqrt{\frac{2\Phi_{t-1}}{S}}$
  $p_t \leftarrow \mathrm{LLPsep}_P(\nabla f(x_t), x_t, r_t, \Phi_t, K)$
  if $p_t =$ **false then**
    $x_{t+1} \leftarrow x_t$
  else
    $x_{t+1} \leftarrow x_t + \alpha(p_t - x_t)$
  end if
end for

As an intermediary step, we first implement a *local weak separation oracle* in Algorithm 6, a *local* version of Oracle 1, analogously to the local linear optimization oracle in (Garber & Hazan, 2013). To this end, we recall a technical lemma from (Garber & Hazan, 2013).

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

Input: $c \in \mathbb{R}^n$ linear objective, $x \in P$ point, $r > 0$ radius, $\Phi > 0$ objective value
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: $i_1, \dots, i_M \quad cv_{i_1} \geq \dots \geq cv_{i_M}$.
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**

Lemma B.1. *(Garber & Hazan, 2013, Lemma 7) Let $P \subseteq \mathbb{R}^n$ be a polytope and $v_1, \dots, v_N$ 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*

$$y - x = -\sum_{i \in [N]} \gamma_i v_i + \left(\sum_{i \in [N]} \gamma_i\right) z$$
$$\sum_{i \in [N]} \gamma_i \leq \frac{\sqrt{n}\mu}{D} \|x - y\|.$$

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

Lemma B.2. *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 $c(x - y) > \Phi/K$,

(ii) or establishes $c(x - z) \leq \Phi$ 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 P. Moreover by construction of y we can write $y = \sum_{j=1}^{M} (\lambda_{i_j} - \gamma_j) v_{i_j} + \Delta v^*$ with $\Delta = \sum_{j=1}^{M} \gamma_j \leq \frac{\sqrt{n}\mu}{D} r$. Therefore

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

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 B.1 to decompose any $y \in P \cap \mathbb{B}_r(x)$ in the following way:

$$y = \sum_{i=1}^{N} (\lambda_i - \gamma_i) v_i + \left(\sum_{i=1}^{N} \gamma_i\right) z,$$

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

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

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 $u = \sum_{i=1}^{N} \eta_i v_i$ with $\sum_{i=1}^{N} \eta_i = \Delta$ and all $\eta_i \geq 0$. 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 B.3. *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 3.3. 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 the negative case, i.e., when $p_t = \textbf{false}$, then case (ii) of Lemma B.2 applies:

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

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

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

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

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

C. Lazy Online Conditional Gradients

In this section we lazify the online conditional gradient algorithm of (Hazan & Kale, 2012) over arbitrary polytopes $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$, resulting in Algorithm 7. We slightly improve constant factors by replacing (Hazan & 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 C.1. *Let $0 \leq b, s < 1$. Let $K > 1$ 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 7 where x_T^* is the minimizer of $\sum_{t=1}^{T} f_t$:*

Algorithm 7 Lazy Online Conditional Gradients (LOCG)

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

for $t = 1$ **to** $T - 1$ **do**
  $\nabla_t \leftarrow \nabla f_t(x_t)$
  if $t = 1$ **then**
    $h_1 \leftarrow \min\{\|\nabla_1\|^* D, 2\|\nabla_1\|^{*2}/S\}$
  else
    $K_t \leftarrow \sqrt{\frac{\|\nabla_t\|^{*2}}{2St^{1-s}}\left(\frac{\|\nabla_t\|^{*2}}{2St^{1-s}} + \Phi_{t-1}\right)}$
    $h_t \leftarrow \Phi_{t-1} + \min\left\{\|\nabla_t\|^* D, \frac{\|\nabla_t\|^{*2}}{St^{1-s}} + 2K_t\right\}$
  end if
  $\Phi_t \leftarrow \frac{h_t + \frac{Ct^{1-b}\gamma_t^2}{2(1-b)}}{1+\frac{\gamma_t}{K}}$
  $v_t \leftarrow \mathrm{LPsep}_P(\sum_{i=1}^{t} \nabla f_i(x_t), x_t, \Phi_t, K)$
  if $v_t =$ **false then**
    $x_{t+1} \leftarrow x_t$
  else
    $x_{t+1} \leftarrow (1 - \gamma_t)x_t + \gamma_t v_t$
    $\Phi_t \leftarrow h_t - \sum_{i=1}^{t} f_i(x_t) + \sum_{i=1}^{t} f_i(x_{t+1})$
  end if
end for

(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

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

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

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

the x_t satisfy

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

where

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

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

$$C_{F_T} := \frac{CT^{1-b}}{1-b} \geq C\sum_{t=1}^{T} t^{-b},$$

$$S_T := ST^{1-s} \leq S\sum_{t=1}^{T} t^{-s}.$$

We prove $F_t(x_t) - F_t(x_t^*) \leq h_t \leq \overline{h}_t$ by induction on t. The case $t = 1$ is clear. Let $\overline{\Phi}_t$ denote the value of Φ_t in Line 9, while we reserve Φ_t to denote its value as used in Line 7. We start by showing $F_t(x_{t+1}) - F_t(x_t^*) \leq \Phi_t \leq \overline{\Phi}_t$. We distinguish two cases depending on v_t from Line 10. If v_t is false, then $\Phi_t = \overline{\Phi}_t$ and the weak separation oracle asserts $\max_{y \in P} \nabla F_t(x_t)(x_t - y) \leq \Phi_t$, which combined with the convexity of F_t provides

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

Otherwise v_t is a vertex of P, then Line 15 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}:

$$\begin{aligned}
F_{t+1}(x_{t+1}) - F_{t+1}(x_{t+1}^*) &\leq (F_t(x_{t+1}) - F_t(x_t^*)) + (f_{t+1}(x_{t+1}) - f_{t+1}(x_{t+1}^*)) \\
&\leq \Phi_t + \|\nabla_{t+1}\|^* \cdot \|x_{t+1} - x_{t+1}^*\| \\
&\leq \Phi_t + \|\nabla_{t+1}\|^* \sqrt{\frac{2}{S_{t+1}}(F_{t+1}(x_{t+1}) - F_{t+1}(x_{t+1}^*))}.
\end{aligned} \tag{3}$$

Solving the quadratic inequality provides

$$F_{t+1}(x_{t+1}) - F_{t+1}(x_{t+1}^*) \leq \Phi_t + \frac{\|\nabla_{t+1}\|^{*2}}{S_{t+1}} + 2\sqrt{\frac{\|\nabla_{t+1}\|^{*2}}{2S_{t+1}} \left(\frac{\|\nabla_{t+1}\|^{*2}}{2S_{t+1}} + \Phi_t\right)}. \tag{4}$$

From Equation (3), 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 7, as claimed.

Now we estimate the right-hand side of Equation (4) 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, $\frac{L^2}{2S_{t+1}} + \Phi_t \le \overline{h}_t$ hence combining with Equation (4) we obtain

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

□

C.1. Stochastic and Adversarial Versions

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

For stochastic cost functions f_t, we obtain bounds from Theorem C.1 (i) similar to (Hazan & 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 $\|y_1 - y_2\|_2 \le \sqrt{\|y_1 - y_2\|_1 \|y_1 - y_2\|_\infty} \le \sqrt{k}$ for all $y_1, y_2 \in P$.

Corollary C.2. *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 7 applied to the f_t (with $b = 0$) yields with probability $1 - \delta$

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

(ii) Without any smoothness assumption, Algorithm 7 (applied to smoothenings of the f_t) provides with probability $1 - \delta$

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

Similar to (Hazan & Kale, 2012, Theorem 4.4), from Theorem C.1 (ii) we obtain the following regret bound for adversarial cost functions with an analogous proof.

Corollary C.3. *For any L-Lipschitz convex cost functions f_t, Algorithm 7 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*

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

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 & Hazan, 2013)).

D. 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. 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 & 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 3 Linear Augmentation Oracle AUG_P(c, x)

Input: $c \in \mathbb{R}^n$ linear objective, $x \in P$ vertex,
Output: $\bar{x} \in P$ vertex 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 & Lovász, 1993; Schulz et al., 1995; Schulz & 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 & 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 & 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 8), 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.

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.

Algorithm 8 Augmenting Weak Separation $\mathrm{LPsep}_P(c, x, \Phi, K)$

Input: $c \in \mathbb{R}^n$ linear objective, $x \in P$ point, $\Phi > 0$ objective value; $K > 1$ accuracy
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:     $x_i \leftarrow \mathrm{AUG}_P(c + \frac{\Phi - c(x - x_{i-1})}{k}(\mathbf{1} - 2x_{i-1}), x_{i-1})$
8:     **if** $x_i = x_{i-1}$ **then**
9:         **return false**
10:     **end if**
11: **end for**
12: **return** x_N

Proposition D.1. *Assume $\|y_1 - y_2\|_1 \leq k$ for all $y_1, y_2 \in P$. Then Algorithm 8 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 $(\mathbf{1} - 2x)v + \|x\|_1 = \|v - x\|_1$ 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.

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 $y \in P$

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

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.,

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

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. □

E. Experiments

We implemented and compared the parameter-free variant of LCG (Algorithm 4) to the standard Frank-Wolf algorithm (CG). Moreover, we implemented and compared Algorithm 3 (LPCG) to the Pairwise Conditional Gradient algorithm (PCG) variant of (Garber & Meshi, 2016) as well as implemented and compared Algorithm 7 (LOCG) to the Online Frank-Wolfe algorithm (OCG) of (Hazan & Kale, 2012). While we did implement the Local Conditional Gradient variant in (Garber & Hazan, 2013) as well, the very large constants in the original algorithms made it impractical to run.

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 E.2.1. 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.

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 for the lazy algorithms a binary search for Φ_0: 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 is 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 reduce 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 are 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 is 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 and the weak separation oracle. The latter was implemented via a callback function to stop the optimization as soon as a good enough feasible solution has been found. 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.

E.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 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:

$$\min \sum_{(i,j)\in\Omega} \|X_{i,j} - a_{i,j}\|^2 \qquad \text{s.t.} \quad \|X\|_* \leq R, \tag{5}$$

where $\|\cdot\|_*$ denotes the nuclear norm, i.e., $\|A\|_* = tr(\sqrt{A^t A})$. The set Ω, the matrix A with entries $a_{i,j}$, and R are given parameters. Similarly to (Lan & 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. The set of entries 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.

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. We construct the objective functions in the same way as for the video colocalization problem.

We will present in the following two sections the complete set of results for various problems grouped by the different versions of the considered algorithms. Every figure contains two columns, each containing one experiment. We use different measures to report performance: the first row reports loss or function value in wall-clock time (including time spent by the oracle), the second row contains loss or function value in the number of iterations. In some cases we include a row reporting the loss or function value over the number of linear optimization calls. In some other cases we report in another row the dual bound or Wolfe gap in wall-clock time. The last row always reports the cumulative number of calls to the linear optimization oracle for the lazy algorithm. The red line denotes the non-lazy algorithm and the greed 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%.

E.1.1. Online Results

Additionally to the quadratic objective functions 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. For online conditional gradient algorithms, in every figure the left column uses linear loss functions, the right one uses quadratic loss functions of the form as described above over the same polytope.

As an instance of the structured regression problem we used the flow-based formulation for Hamiltonian cycles in graphs, i.e., the traveling salesman problem (TSP) for graphs with 11 and 16 nodes (Figures 7 and 8). While relatively small, the oracle problem can be solved in reasonable time for these instances. Another instance of the structured regression problem uses the standard formulation of the cut polytope for graphs with 23 and 28 nodes as the feasible region (Figures 9 and 10). Another set of feasible regions corresponding to NP-hard problems for the structured regression problem we tested our algorithm on are 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 (Figure 11 and 12). 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 13). 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. We tested on the structured regression problems with the MIPLIB (Achterberg et al., 2006; Koch et al., 2011)) instances `eil33-2` (Figure 14) and `air04` (Figure 15) as feasible regions. 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 10 and 25 nodes (Figures 16 and 17).

We observed that 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.

E.1.2. Offline Results

We describe the considered instances in the offline case separately for the vanilla Frank-Wolfe method and the Pairwise Conditional Gradients 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/` (Figures 18, 19 and 20). 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 these examples we did not use line search. We give the used parameters for each example in the figures below (Figures 21, 22, 23 and 24). 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 (Figures 25), two on two different cut polytope instances (Figure 26) and finally two on two spanning tree instances of different size (Figure 27).

Similarly to the online case, we observe 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`, `eilB101`, `nw04`, `disctom`, `m100n500k4r1` (Figures 28, 29 and 30).

Again similarly to the online case and the vanilla Frank-Wolfe algorihtm, we observe a significant improvement in wall-clock time of LPCG compared to CG, due to the faster iteration of the lazy algorithm.

E.2. Performance improvements, parameter sensitivity, and tuning

E.2.1. Paramter-free vs. textbook variant

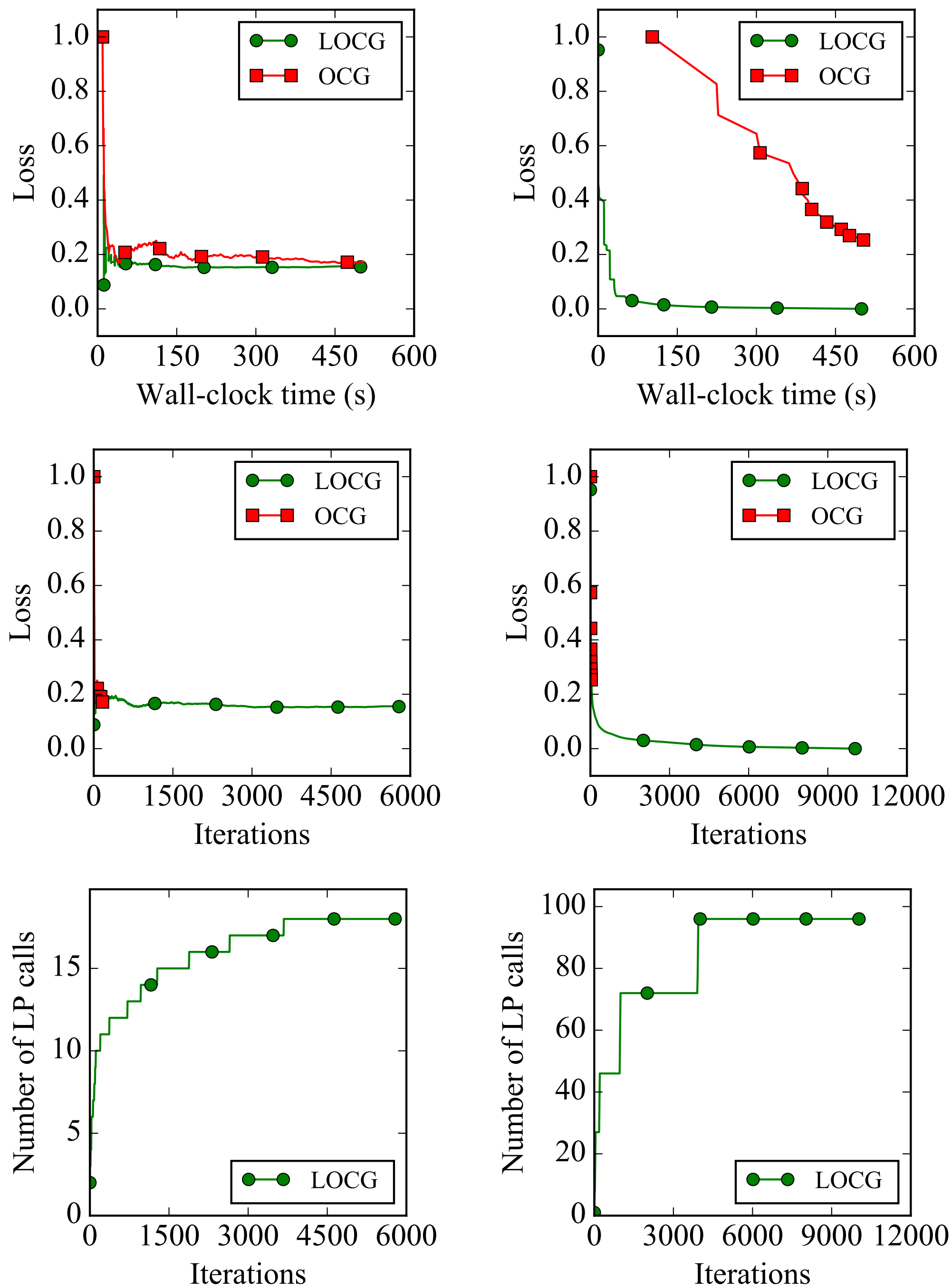
For illustrative purposes, we compare the textbook variant of the lazy conditional gradients (Algorithm 2) with its parameter-free counterpart (Algorithm 4) in Figure 31. The parameter-free variant outperforms the textbook variant due to the active management of Φ combined with line search.

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

F. Final Remarks

We would like to close with a few final remarks. 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) or atoms.



cache hit rate: 99.7% cache hit rate: 99.0%

Figure 7. LOCG vs. OCG with the TSP polytope for a graph with 11 nodes as the feasible region and with a 500 seconds time limit. OCG completed only a few iterations, resulting in a several times larger final loss for quadratic loss functions. Notice that with time LOCG needed fewer and fewer LP calls.

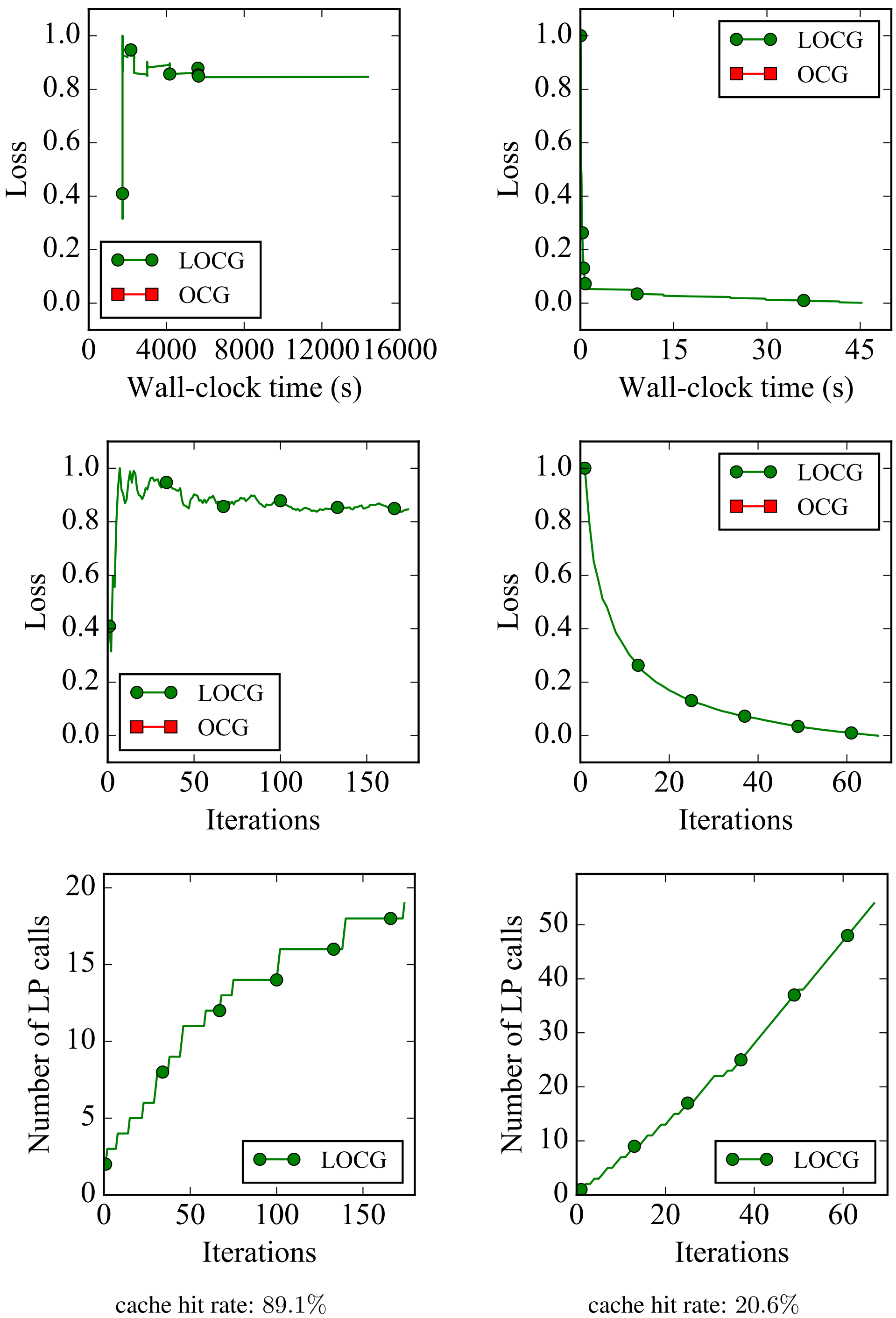
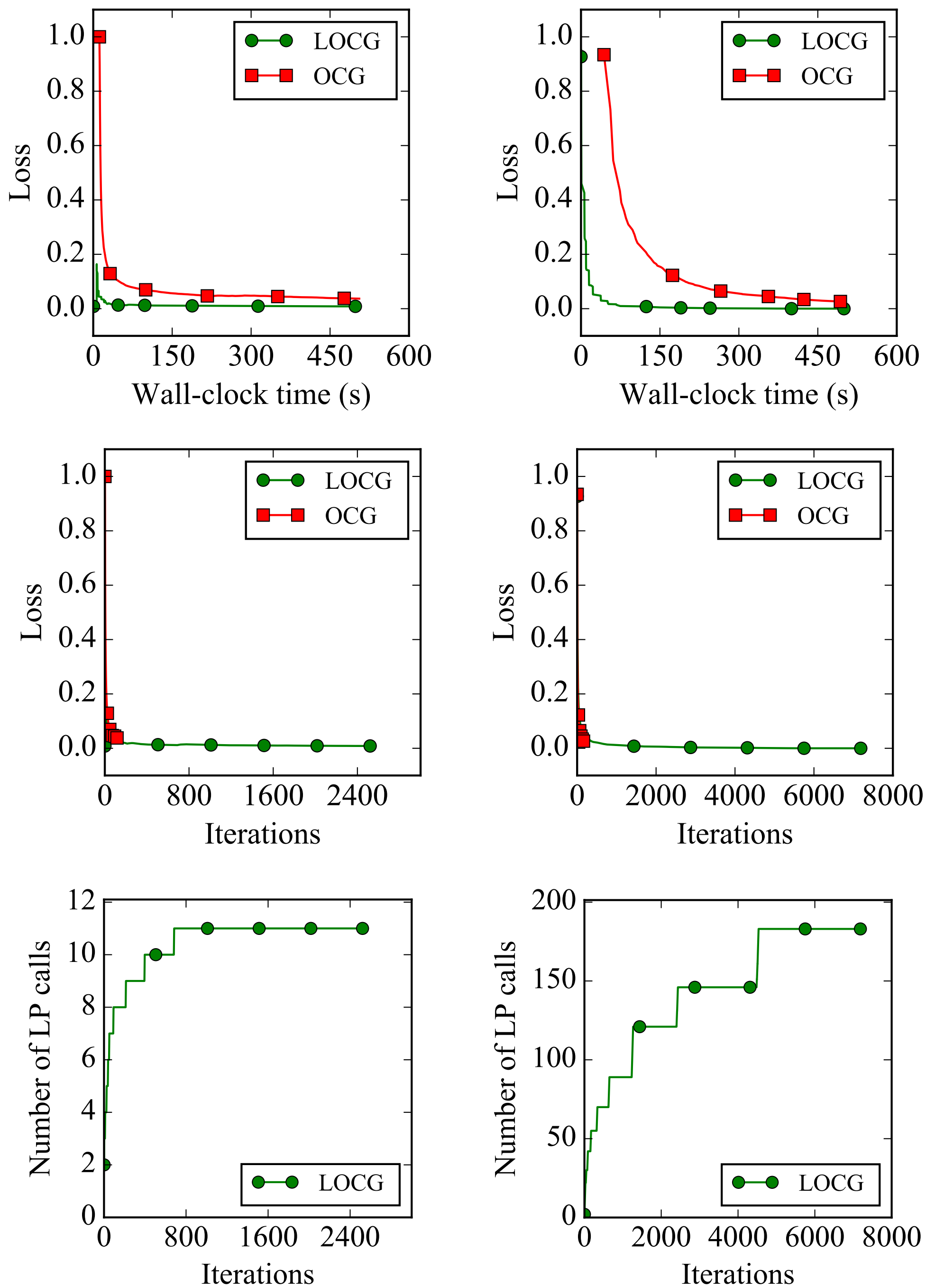


Figure 8. LOCG vs. OCG on the TSP polytope for a graph with 16 nodes with a time limit of 7200 seconds. OCG was not able to complete a single iteration and in the quadratic case even LOCG could not complete any more iteration after 50s. The quadratic losses on the right nicely demonstrate speed improvements (mostly) through early termination of the linear optimization as the cache rate is only 20.6%.



cache hit rate: 99.6%

cache hit rate: 97.5%

Figure 9. LOCG vs. OCG on the cut polytope for a graph with 23 nodes. Both LOCG and OCG converge to the optimum in a few iterations for linear losses, while LOCG is remarkably faster for quadratic losses. It demonstrates that the advantage of lazy algorithms strongly correlates with the difficulty of linear optimization. For linear losses, remarkably LOCG needed no oracle calls after one third of the time.

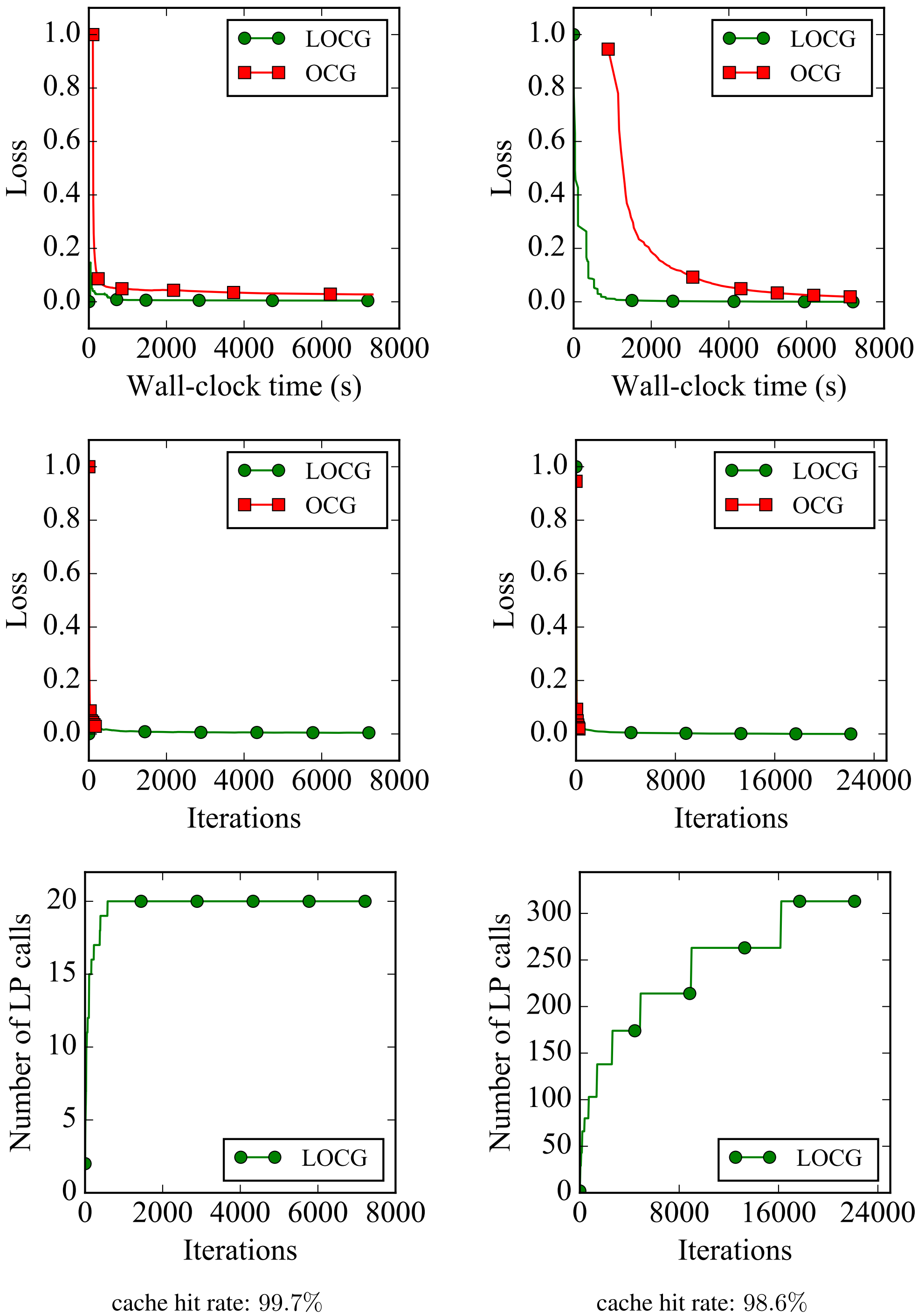
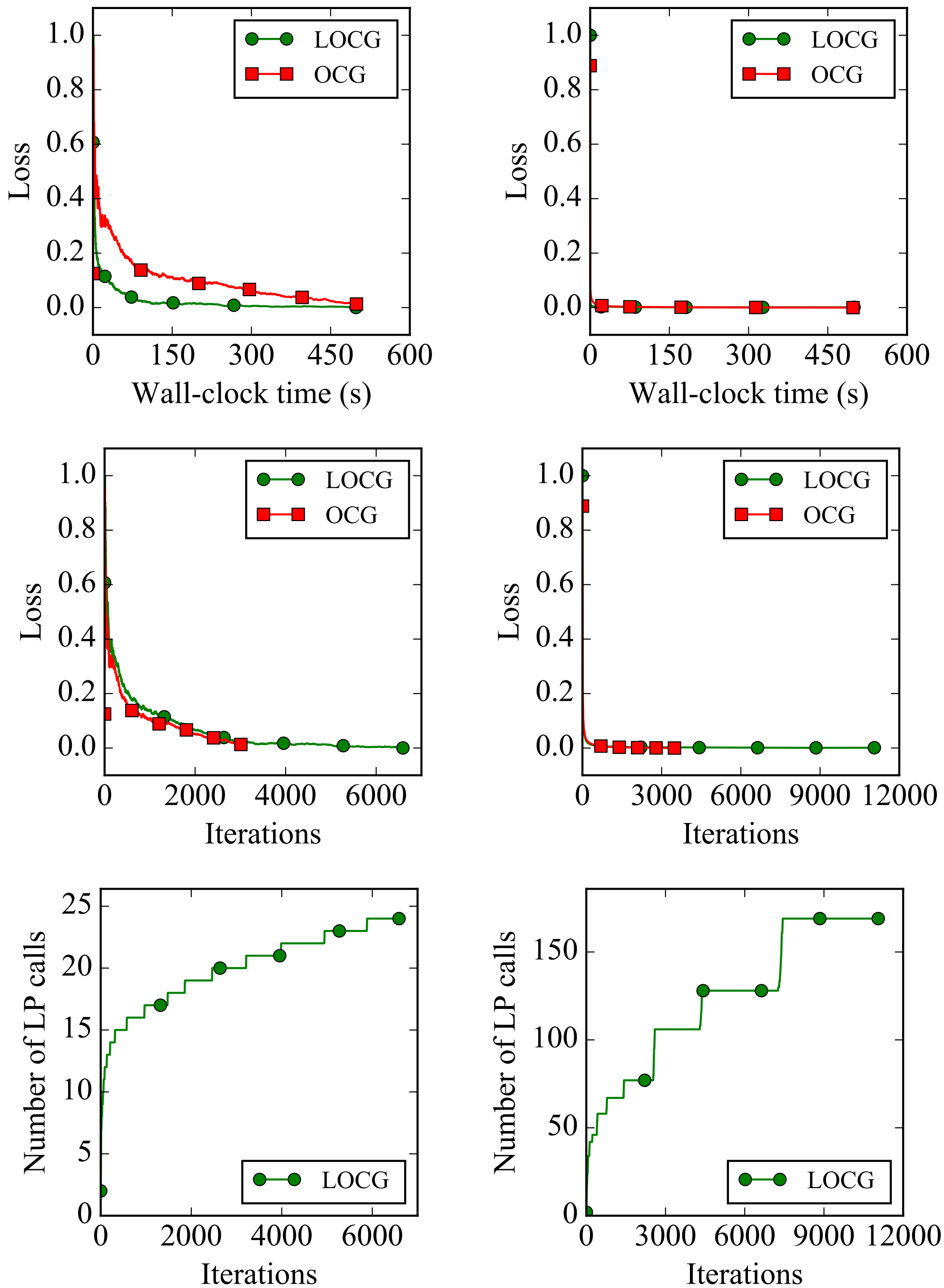
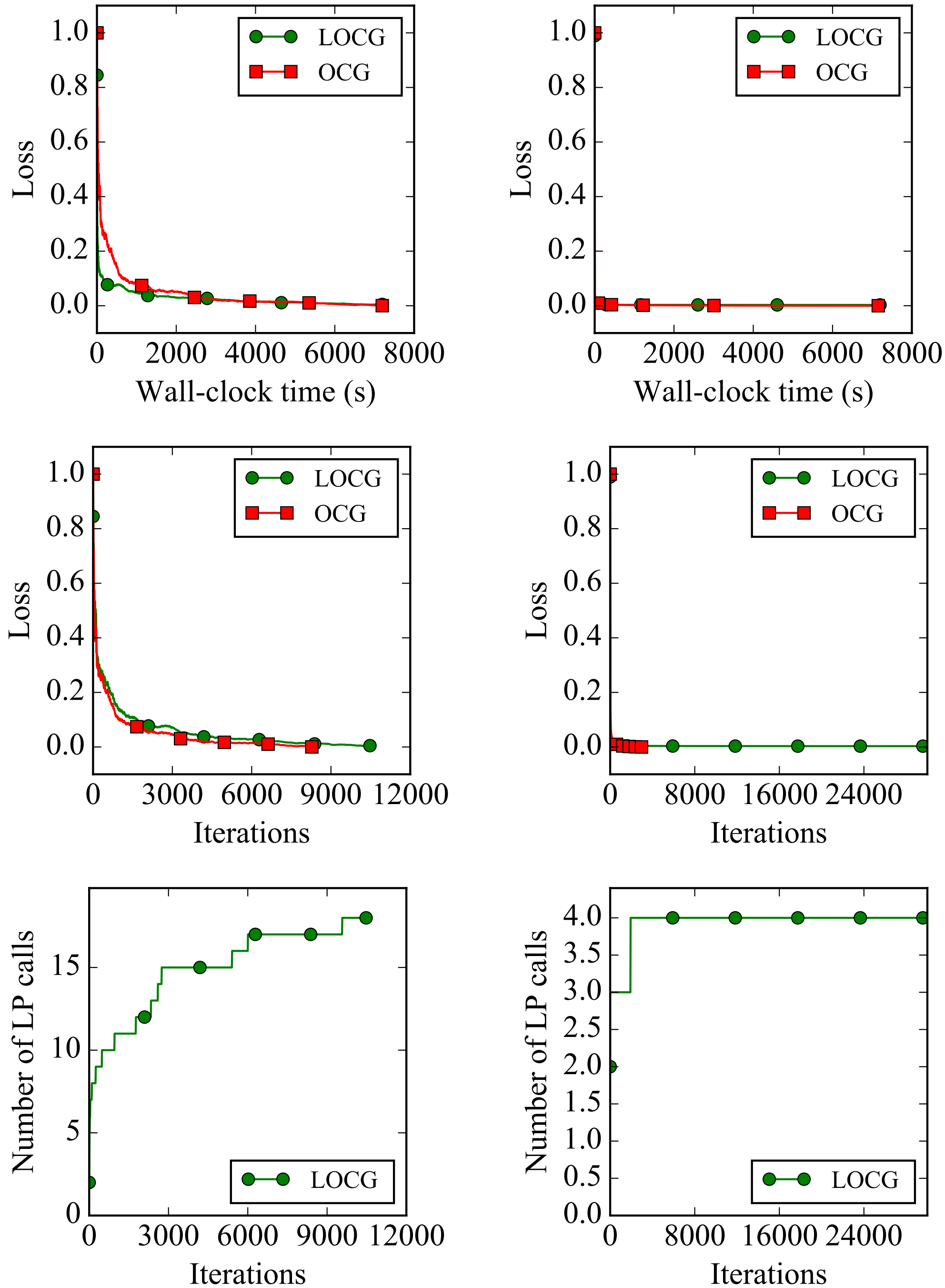


Figure 10. LOCG vs. OCG on the cut polytope for a 28-node graph. As for the smaller problem, this also illustrates the advantage of lazy algorithms when linear optimization is expensive. Again, LOCG needed no oracle calls after a small initial amount of time.



cache hit rate: 99.6%   cache hit rate: 98.5%

Figure 11. LOCG vs. OCG on a small QUBO instance. For quadratic losses, both algorithms converged very fast while LOCG still has a significant edge. For linear losses, LOCG is noticeably faster than OCG.



cache hit rate: 99.8%

cache hit rate: 99.99%

Figure 12. LOCG vs. OCG on a large QUBO instance. Both algorithms converge fast to the optimum. Interestingly, LOCG only performs 4 LP calls.

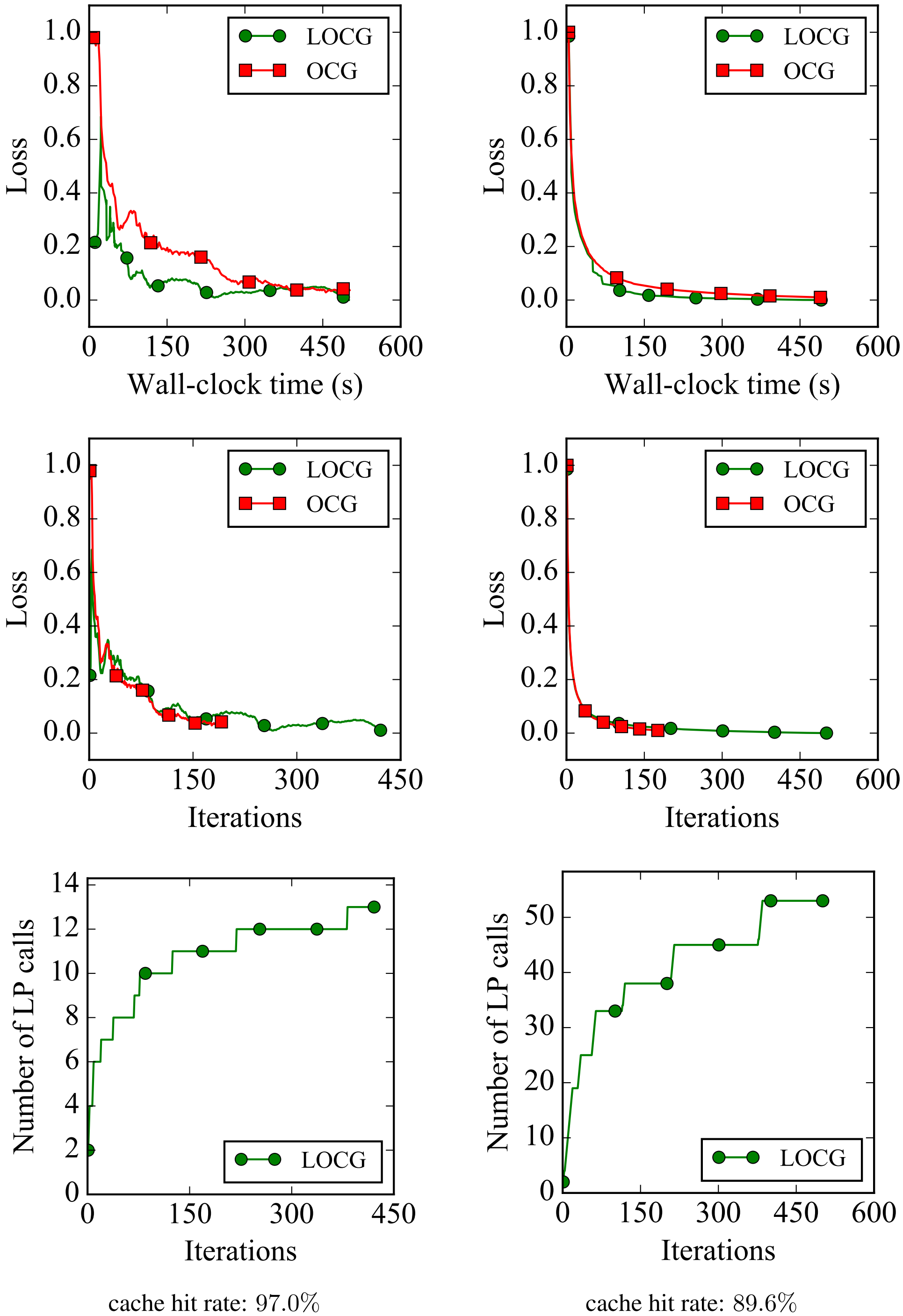


Figure 13. LOCG vs. OCG on a path polytope. Similar convergence rate in the number of iterations, but significant difference in terms of wall-clock time.

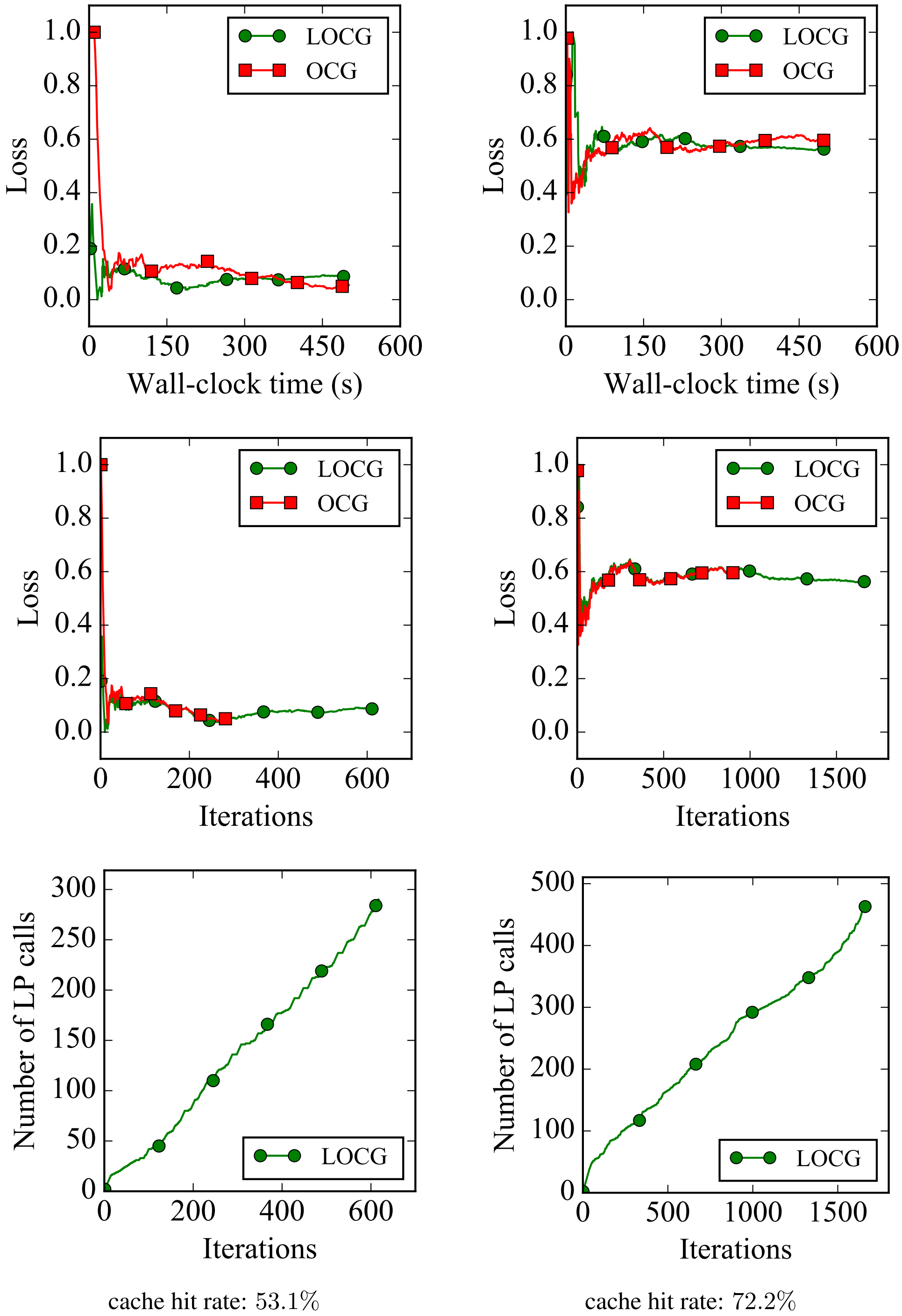


Figure 14. LOCG vs. OCG on the MIPLIB instance `eil33-2`. All algorithms performed comparably, due to fast convergence in this case.

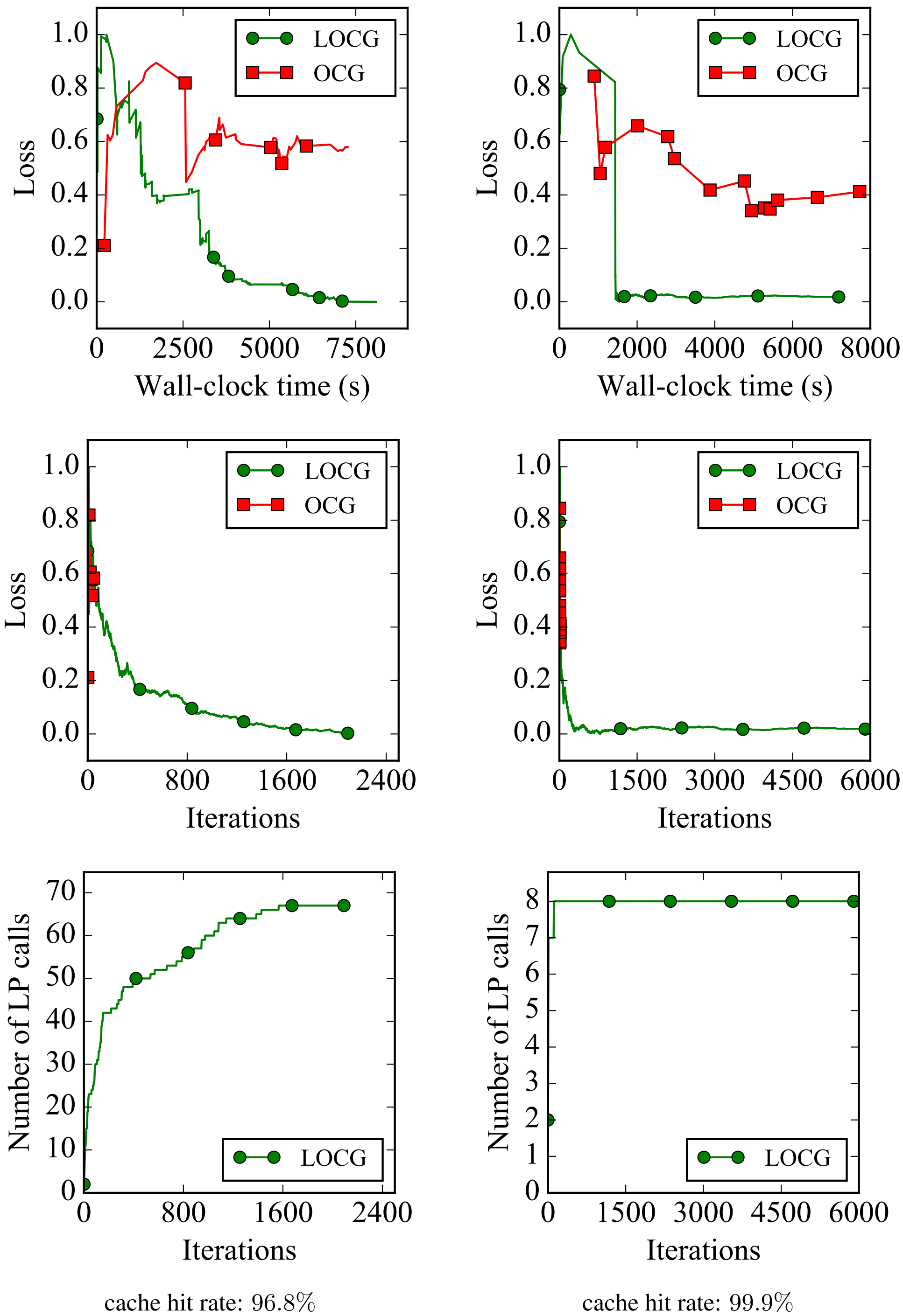
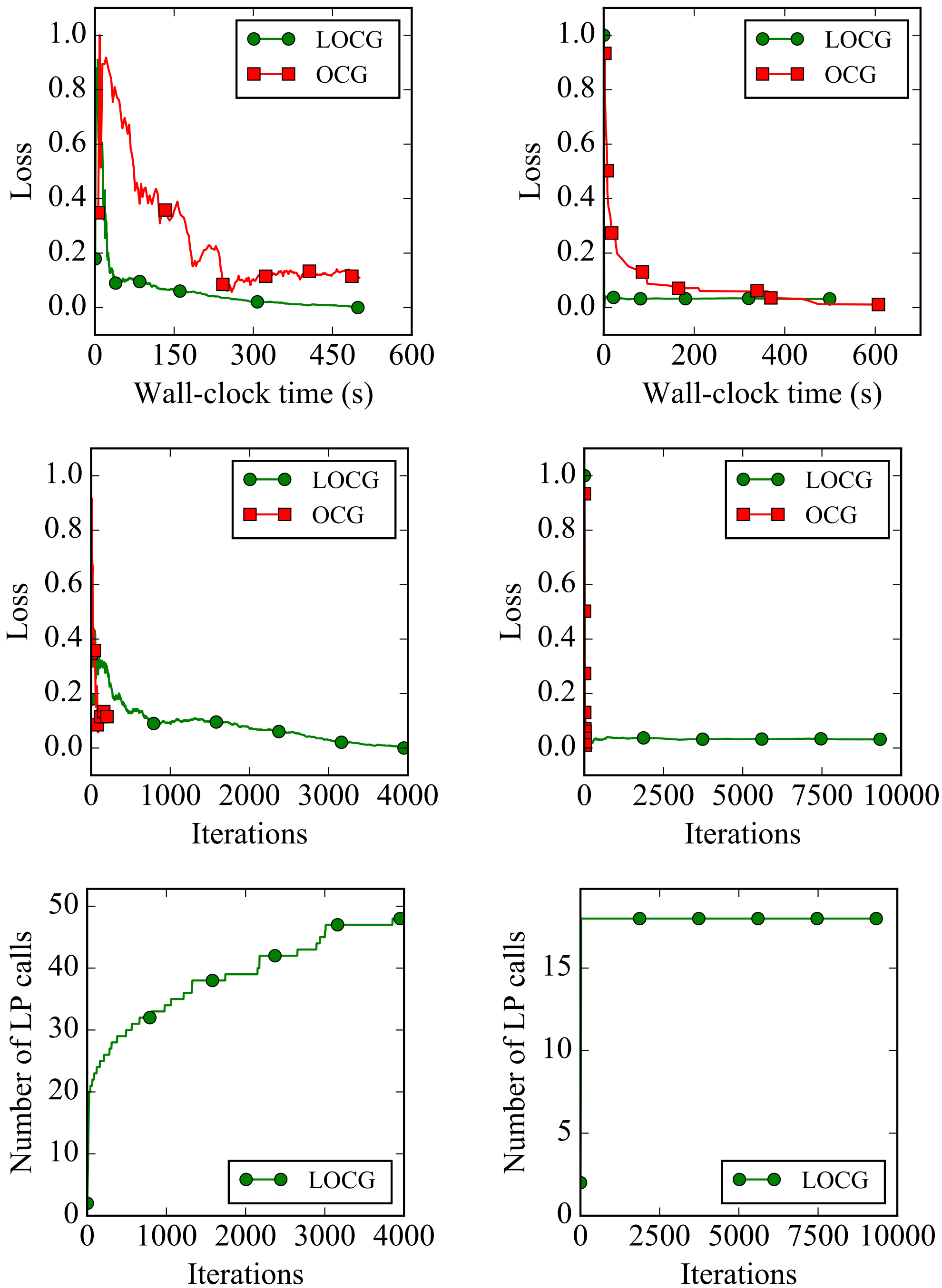


Figure 15. LOCG vs. OCG on the MIPLIB instance `air04`. LOCG clearly outperforms OCG as the provided time was not enough for OCG to complete the necessary number of iterations for entering reasonable convergence.



cache hit rate: 98.8%　　　　cache hit rate: 99.8%

Figure 16. LOCG vs. OCG on a spanning tree instance for a 10-node graph. LOCG makes significantly more iterations, few oracle calls, and converges faster in wall-clock time.

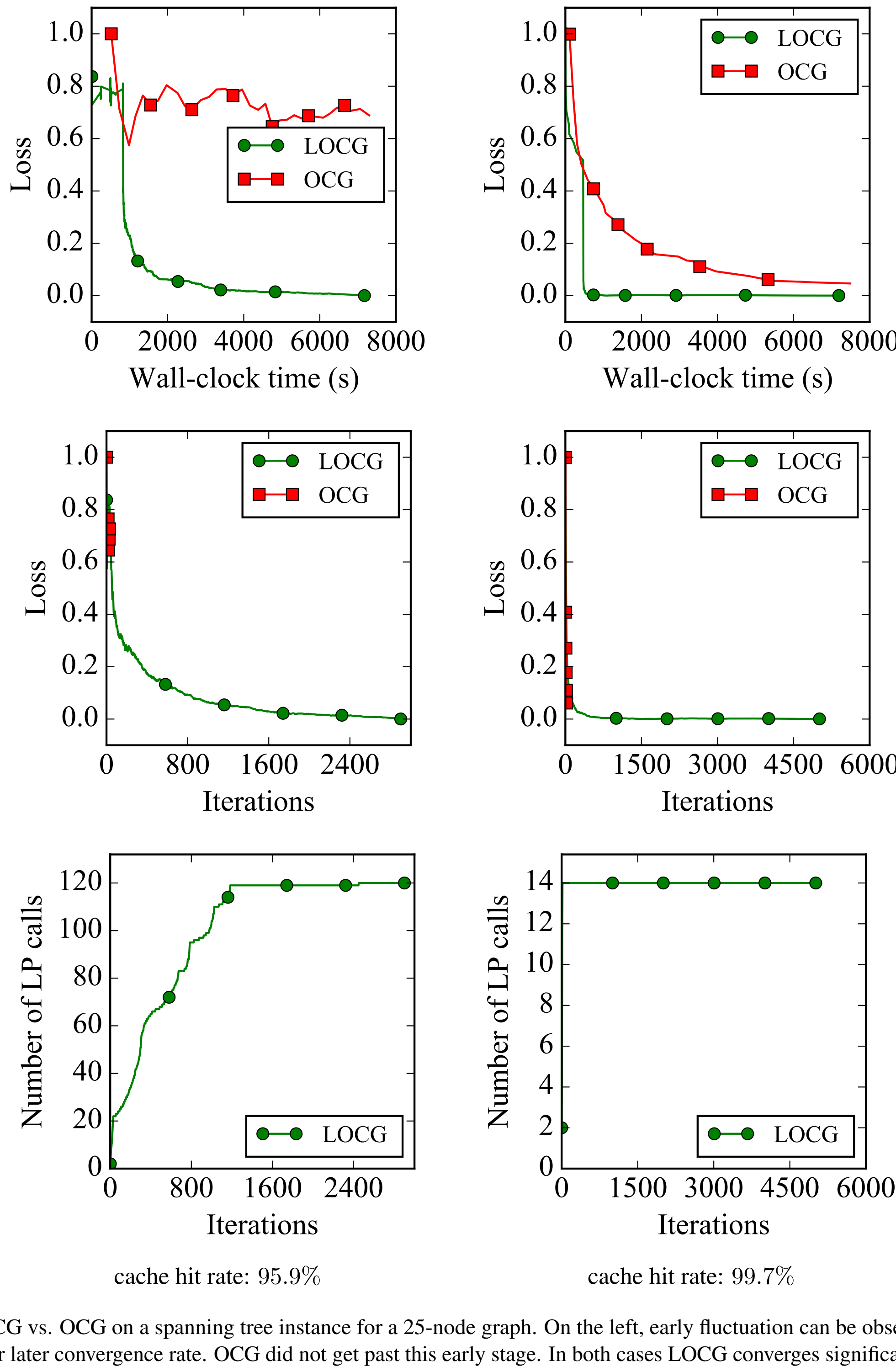
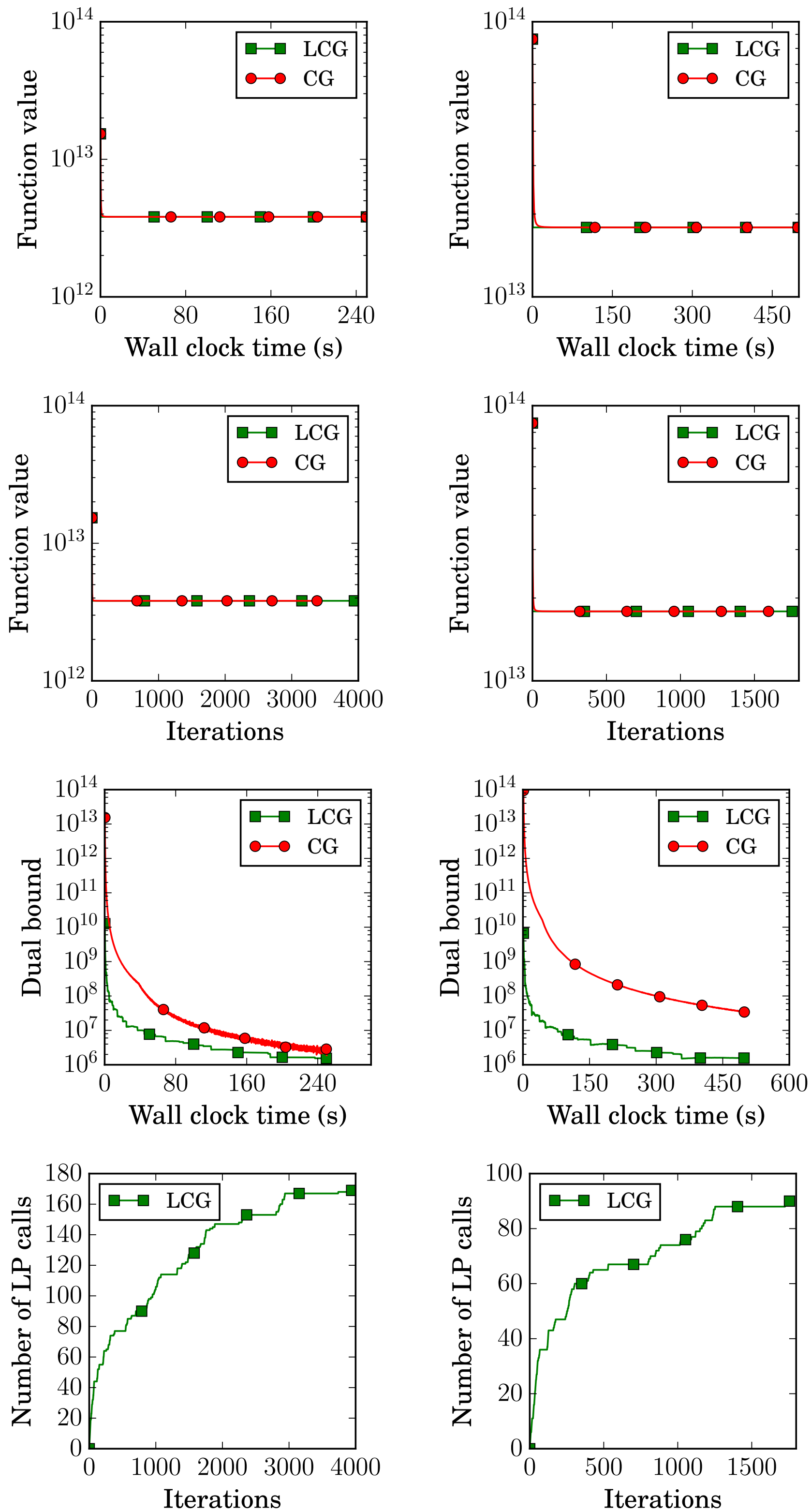


Figure 17. LOCG vs. OCG on a spanning tree instance for a 25-node graph. On the left, early fluctuation can be observed, bearing no consequence for later convergence rate. OCG did not get past this early stage. In both cases LOCG converges significantly faster.



cache hit rate: 95.72%   cache hit rate: 94.83%

Figure 18. LCG vs. CG on small netgen instances `netgen 08a` (left) and `netgen 10a` (right) with quadratic objective functions. In both cases both algorithms are able to reduce the function value very fast, however the dual bound or Wolfe gap is reduced much faster by LCG. Observe that the vertical axis is given with a logscale.

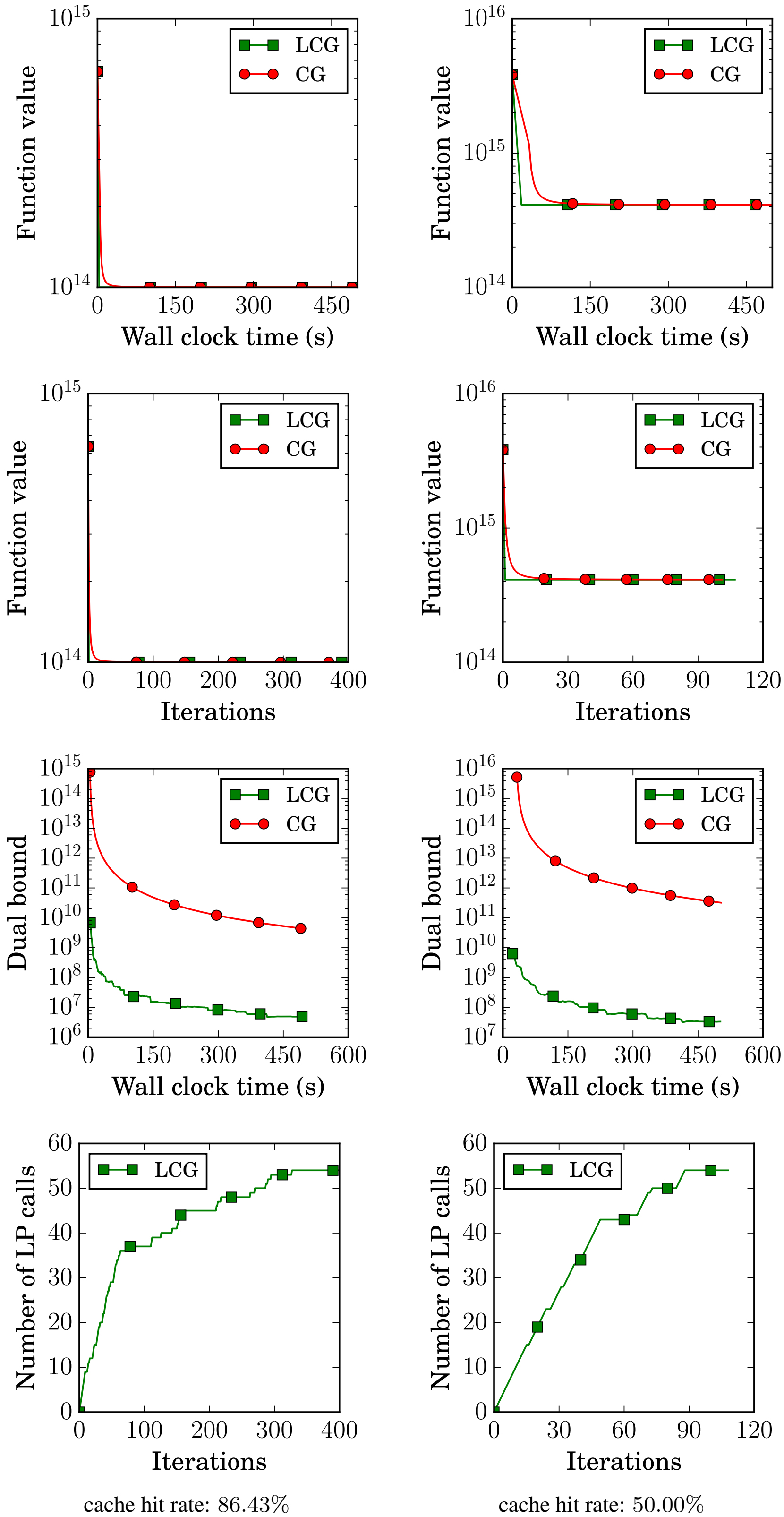
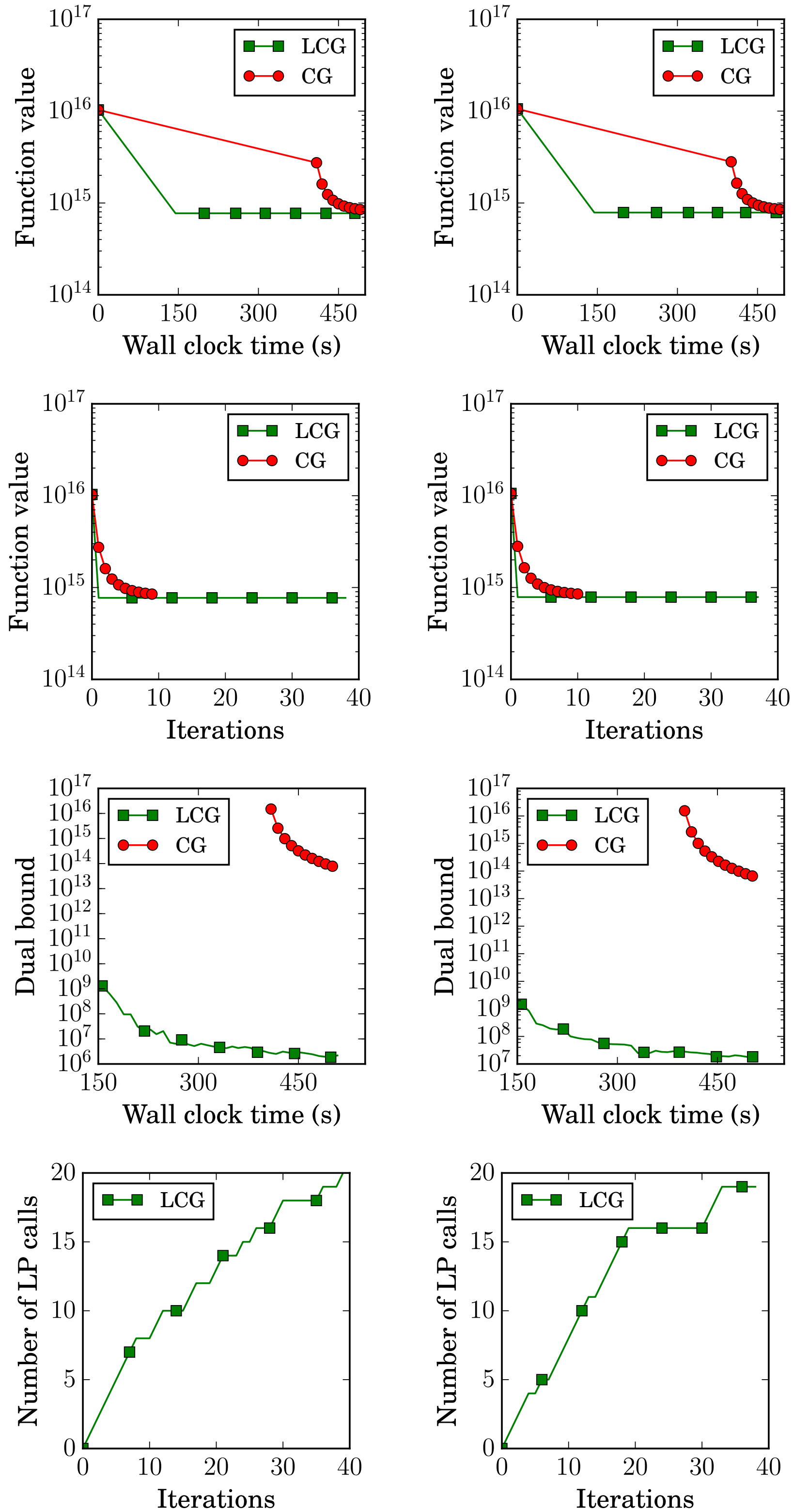


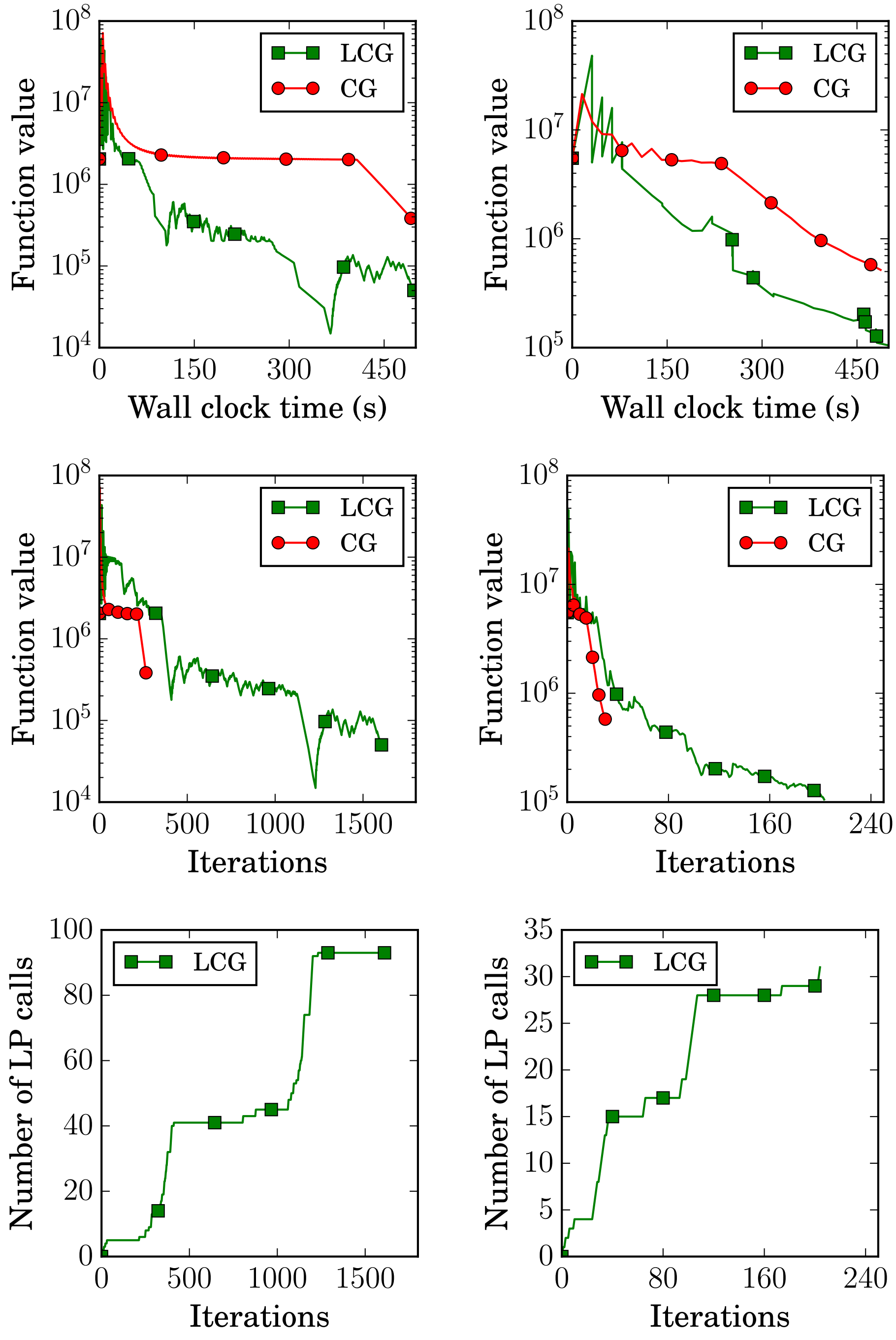
Figure 19. LCG vs. CG on medium sized netgen instances `netgen 12b` (left) and `netgen 14a` (right) with quadratic objective functions. The behavior of both versions on these instances is very similar to the small netgen instances (Figure 18), however both in the function value and the dual bound the difference between the lazy and the non-lazy version is more prominent. Again, we used a logscale for the vertical axis.



cache hit rate: 48.72%

cache hit rate: 50.00%

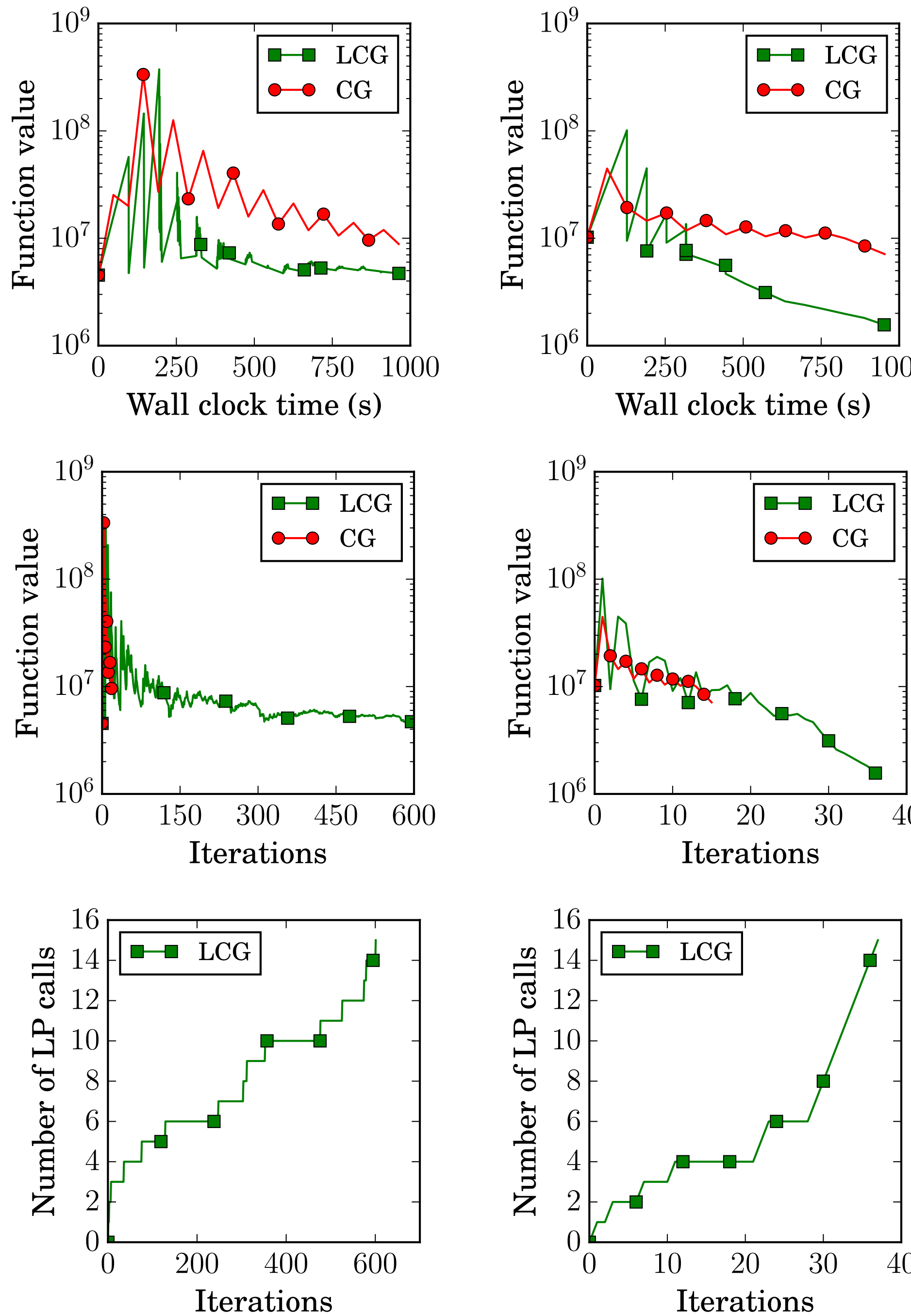
Figure 20. LCG vs. CG on large netgen instances `netgen 16a` (left) and `netgen 16b` (right) with quadratic objective functions. In both cases the difference in function value between the two versions of the algorithm is large. In the dual bound the performance of the lazy version is multiple orders 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.



cache hit rate: 94.24%

cache hit rate: 84.80%

Figure 21. LCG vs. CG on two matrix completion instances. We solve the problem as given in Equation (5) with the paramters $n = 3000$, $m = 1000$, $r = 10$ 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 interations, however significantly faster in wall clock time.



cache hit rate: 97.50%   cache hit rate: 59.46%

Figure 22. LCG vs. CG on two more matrix completion instances. The parameters for Equation (5) 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 interations, however in wall clock time the lazy version reaches lower function values faster.

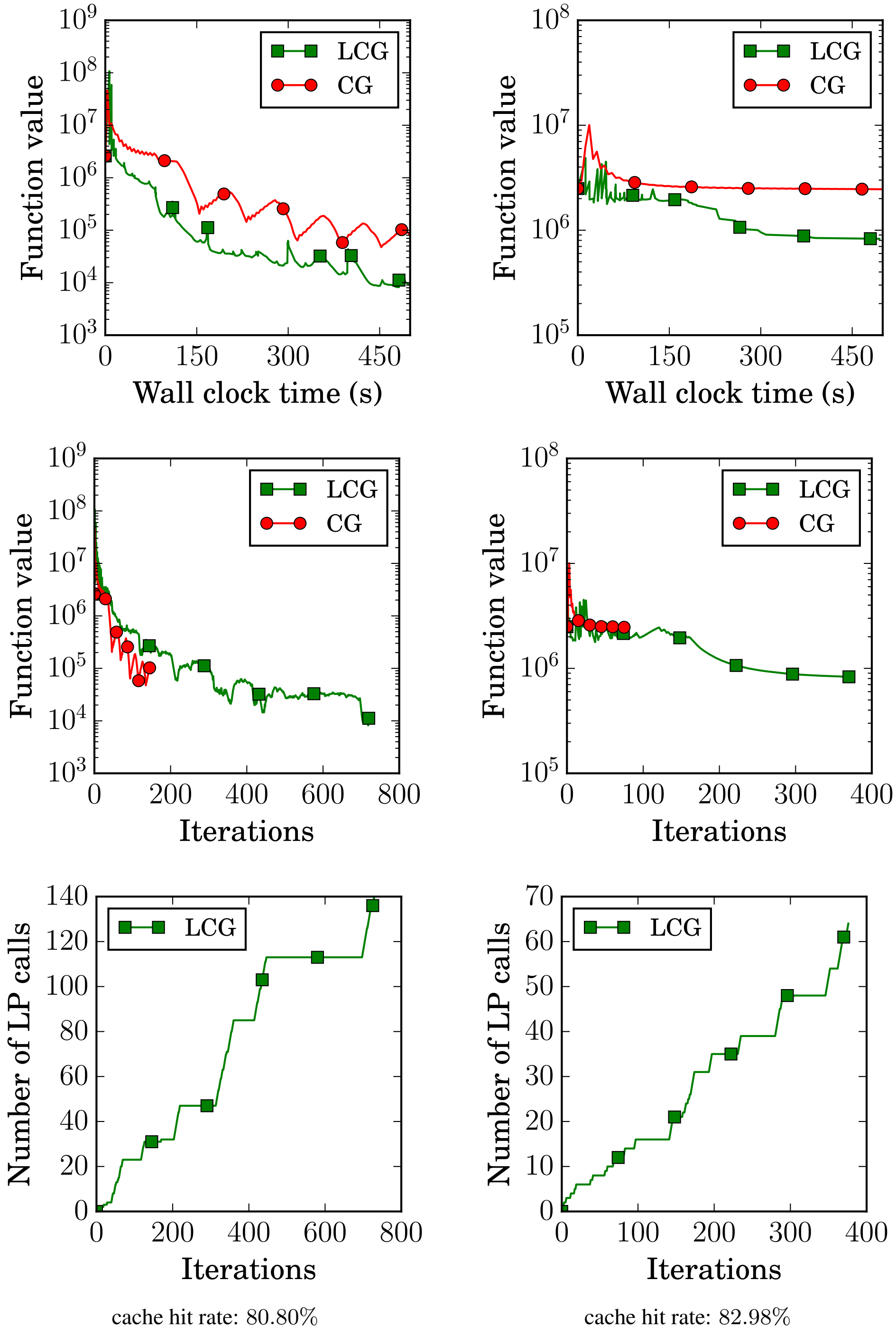
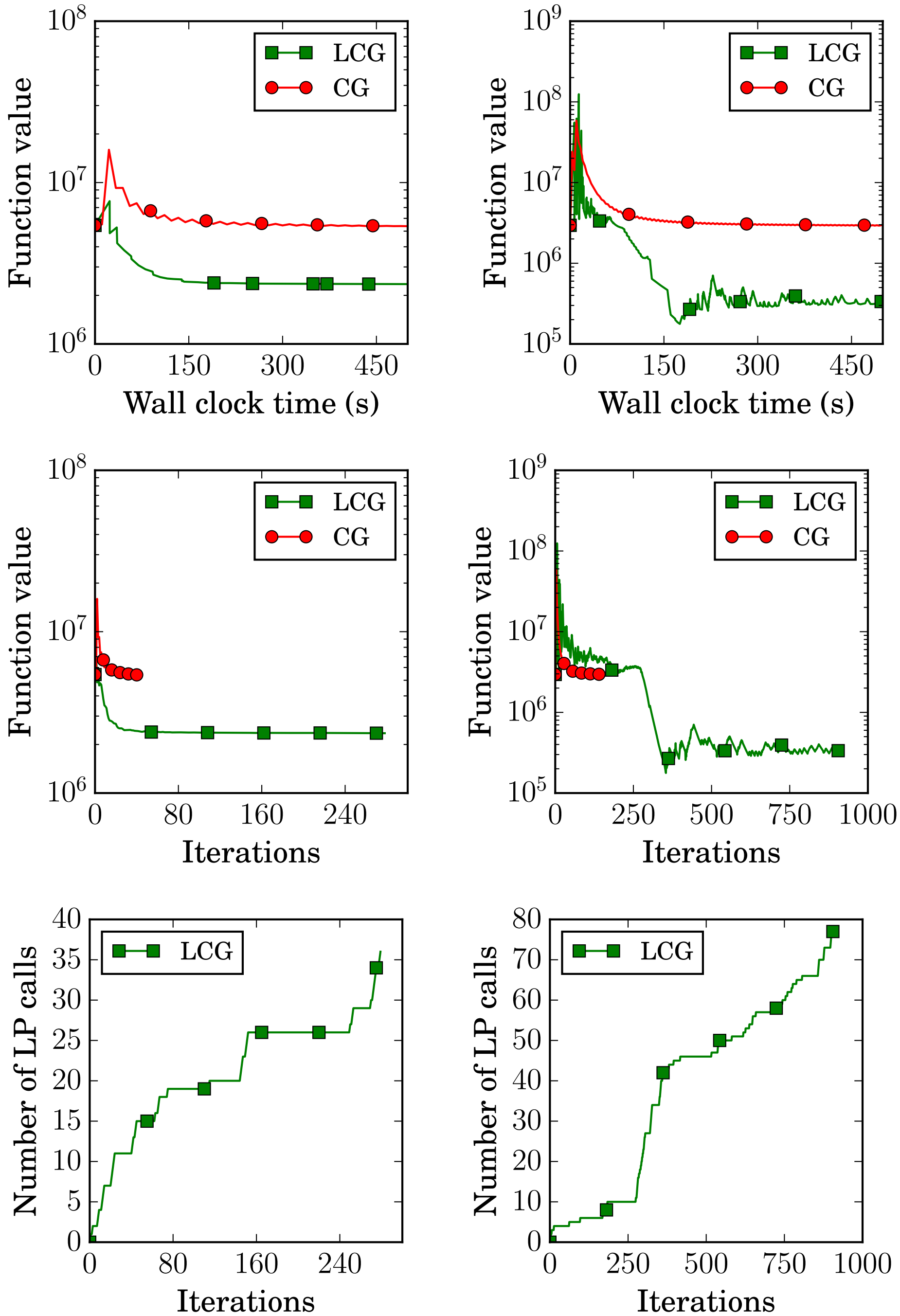


Figure 23. LCG vs. CG on our fifth and sixth instances of the matrix completion problem. The parameters are $n = 5000$, $m = 100$, $r = 10$ and $R = 15000$ for the left instance and $n = 3000$, $m = 2000$, $r = 10$ and $R = 10000$ for the right instance. The behavior is very similar to Figure 22. similar performance over iterations however advantages for the lazy version over wall clock time.



cache hit rate: 87.10%

cache hit rate: 91.55%

Figure 24. LCG vs. CG on the final two matrix completion instances. The parameters are $n = 10000$, $m = 1000$, $r = 10$ and $R = 1 - 000$ for the left instance and $n = 5000$, $m = 1000$, $r = 10$ and $R = 30000$ for the right instance. On the left in both measures, instances and wall clock time, the lazy version performs better than the non-lazy counterpart, due to a suboptimal direction at the beginning with a fairly large step size in the non-lazy version.

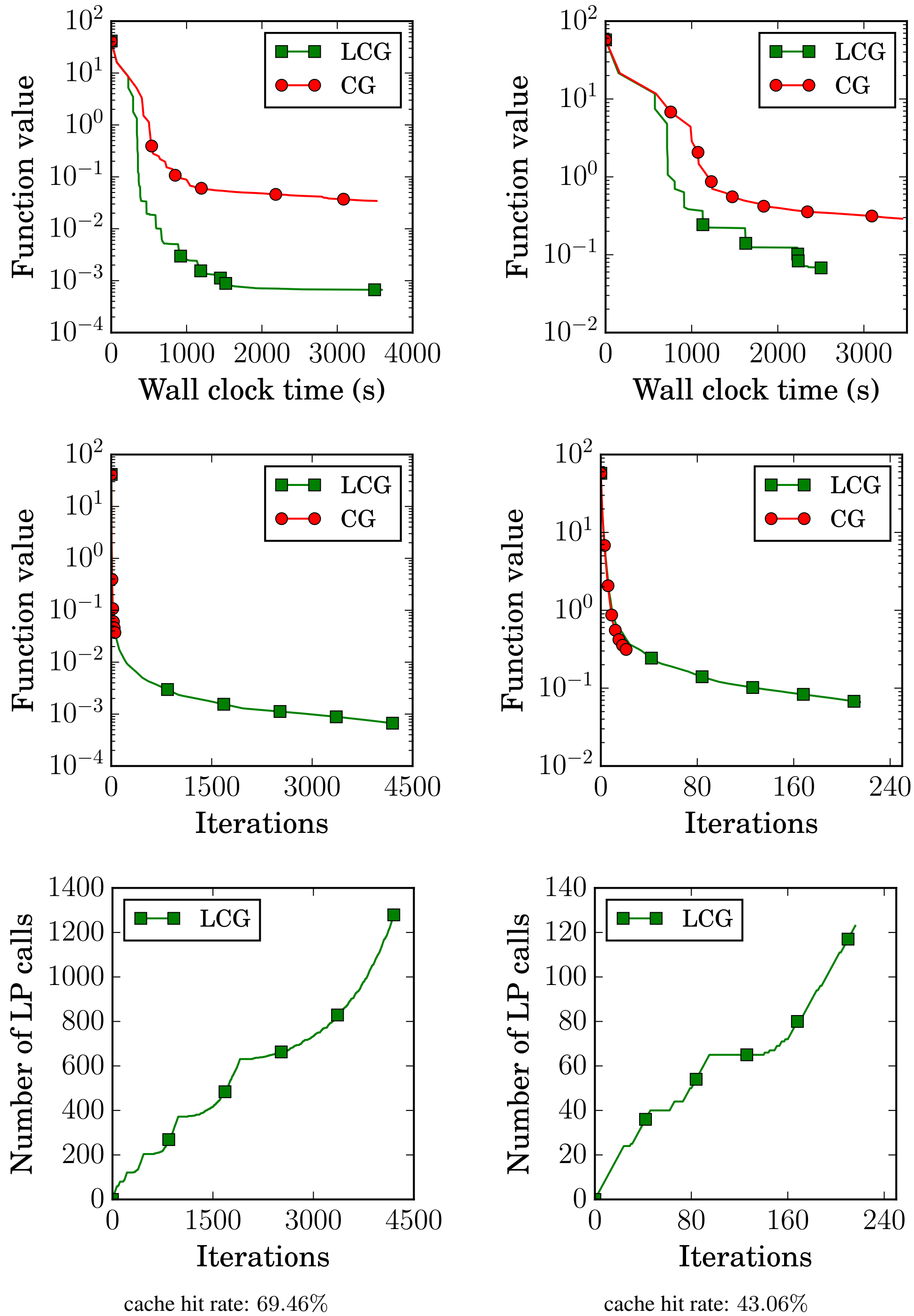
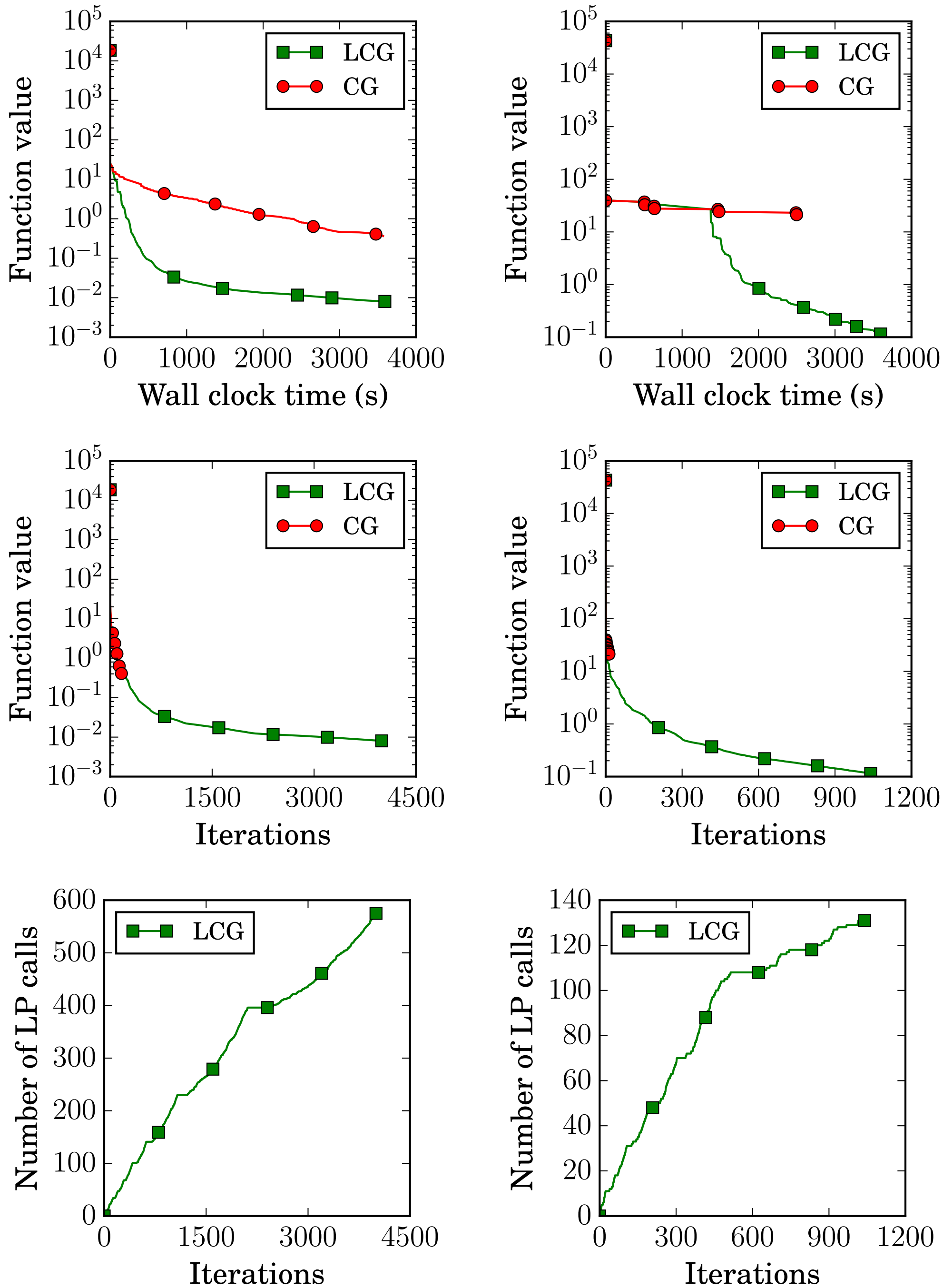


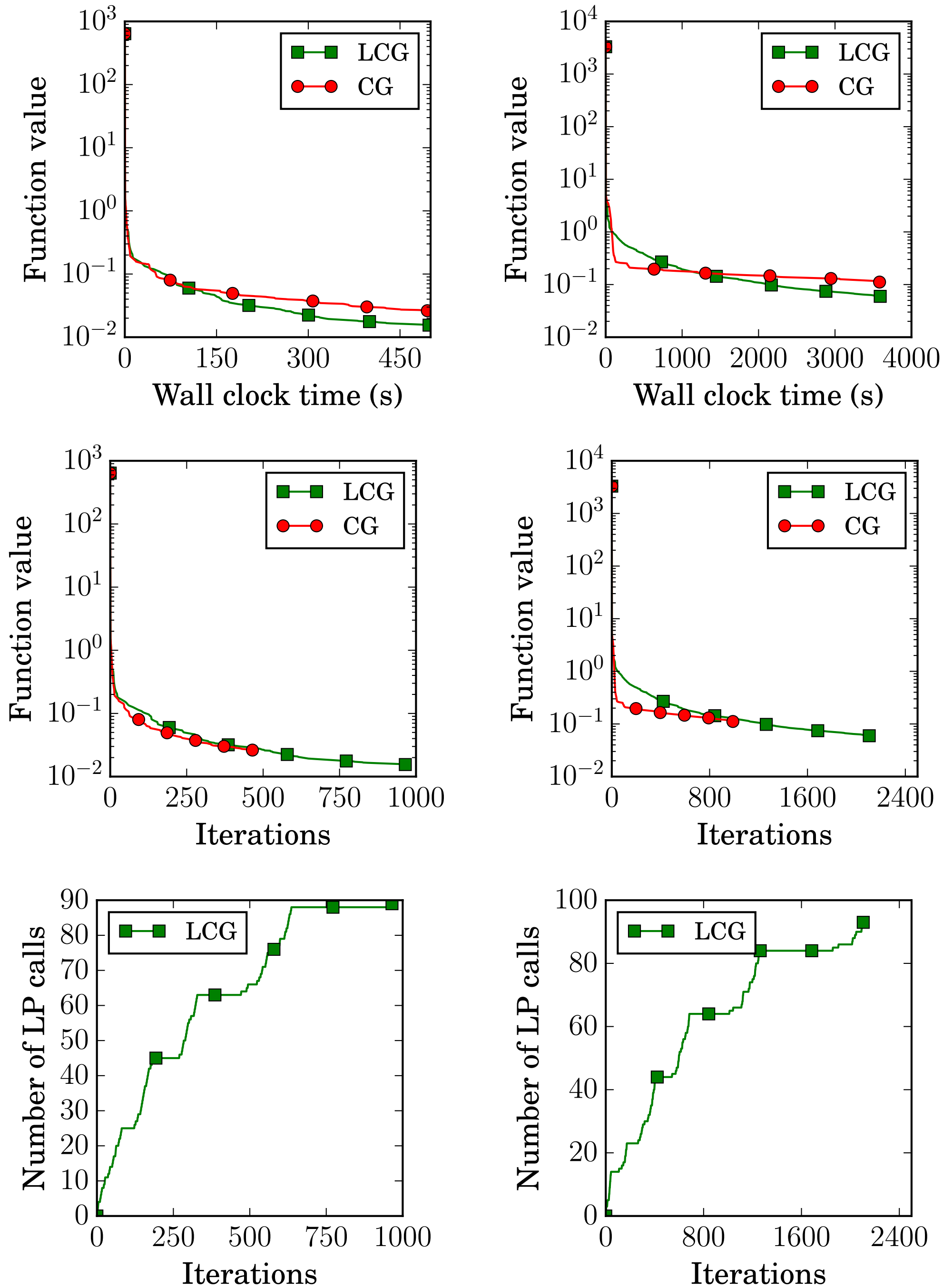
Figure 25. LCG vs. CG on structured regression problems with feasible regions being a TSP polytope over 11 nodes (left) and 12 nodes (right). In both cases LCG is significantly faster in wall-clock time.



cache hit rate: 85.61%

cache hit rate: 87.48%

Figure 26. LCG vs. CG on structured regression instances using cut polytopes over a graph on 23 nodes (left) and over 28 nodes (right) as feasible region. In both instances LCG performs significantly better than CG.



cache hit rate: 90.83%

cache hit rate: 95.59%

Figure 27. 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.

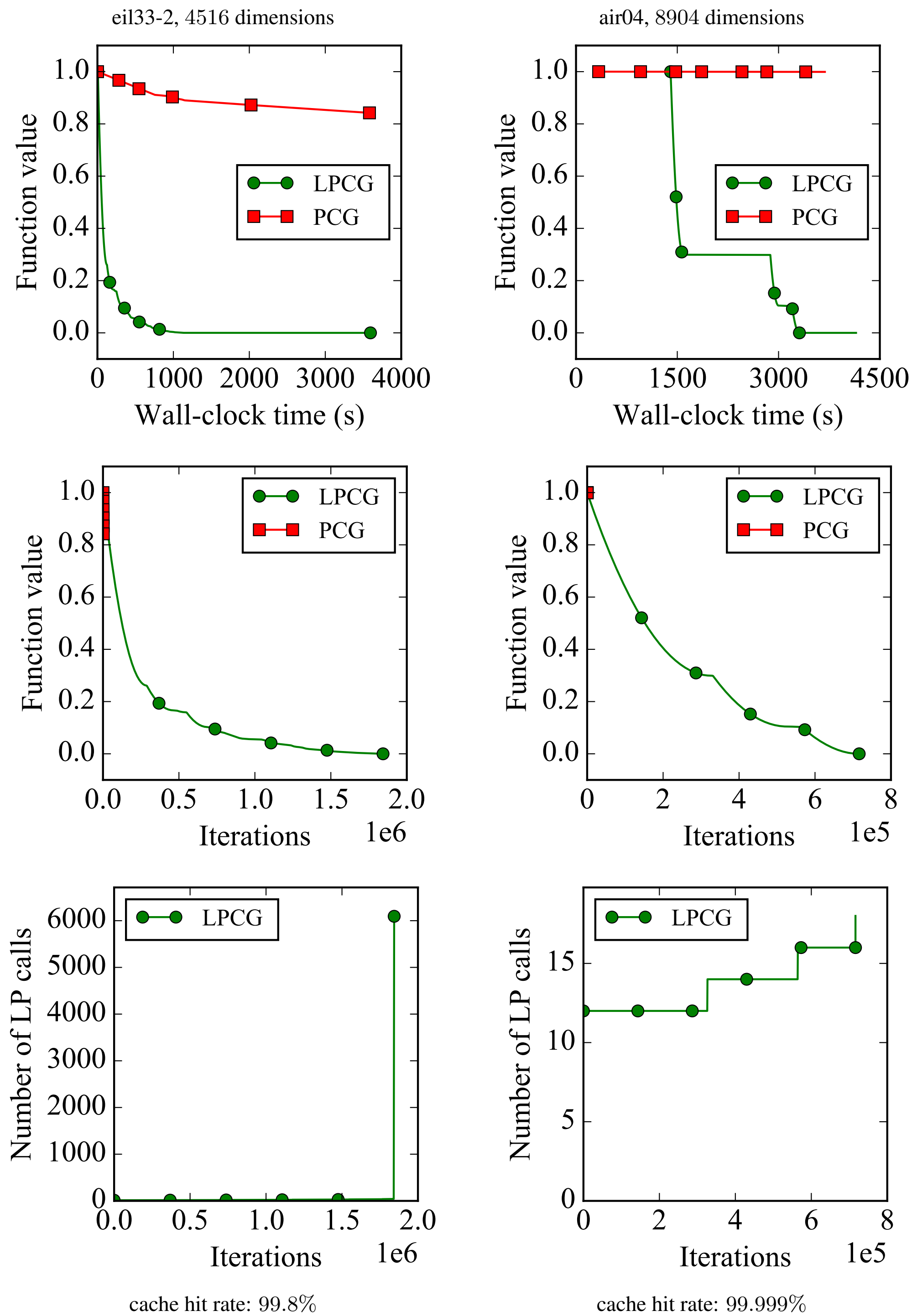


Figure 28. LPCG vs. PCG on two MIPLIB instances `eil33-2` and `air04`. LPCG converges very fast, making millions of iterations with a relatively few oracle calls, while PCG completed only comparably few iterations due to the time-consuming oracle calls. This clearly illustrates the advantage of lazy 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 tigthening of Φ_0, before converging significantly faster than PCG.

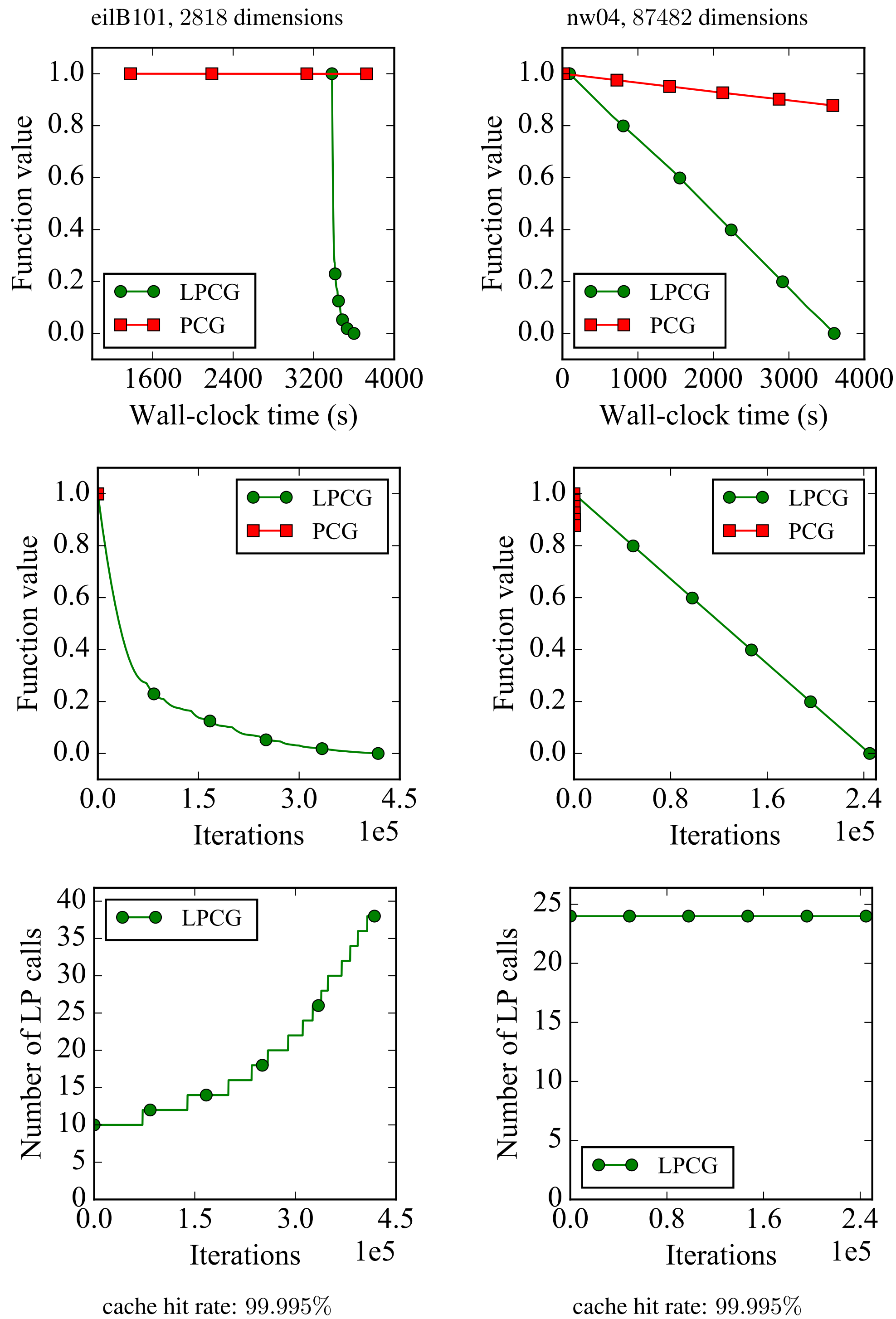
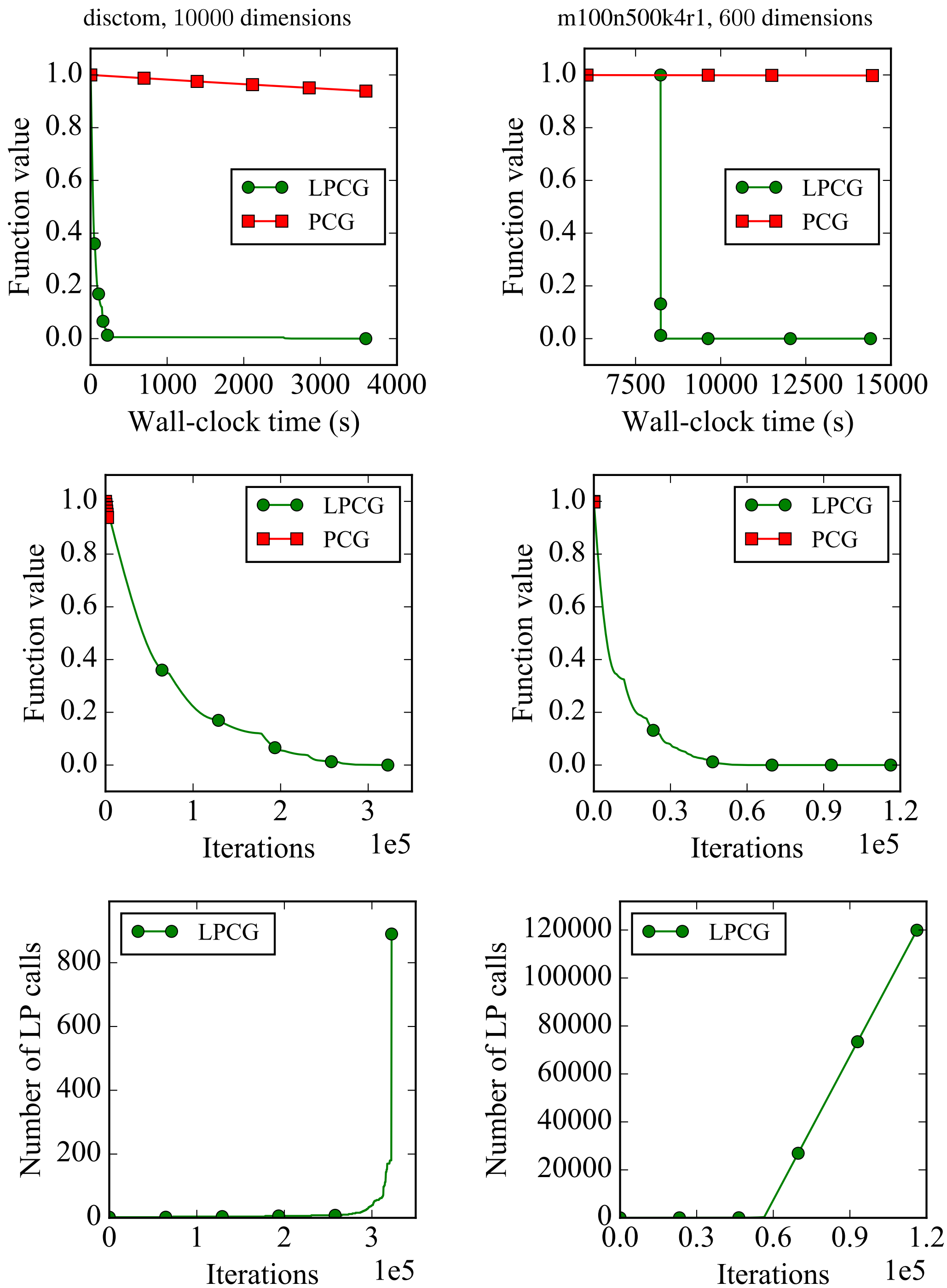


Figure 29. LPCG vs. PCG on MIPLIB instances `eilB101` and `nw04` with quadratic loss functions. For the `eilB101` instance, LPCG spent most of the time tightening Φ_0, after which it converged very fast, while PCG was unable to complete a single iteration even solving the problem only approximately. For the `nw04` instance LPCG needed no more oracle calls after an initial phase, while significantly outperforming PCG.



cache hit rate: 99.9%

cache hit rate: 48.4%

Figure 30. LPCG vs. PCG on MIPLIB instances `disctom` and `m100n500k4r1`. After very fast convergence, there is a huge increase in the number of oracle calls for the lazy algorithm LPCG due to reaching ε-optimality as explained before. On the right the initial bound tightening for Φ_0 took a considerable amount of time but then convergence is almost instantaneous.

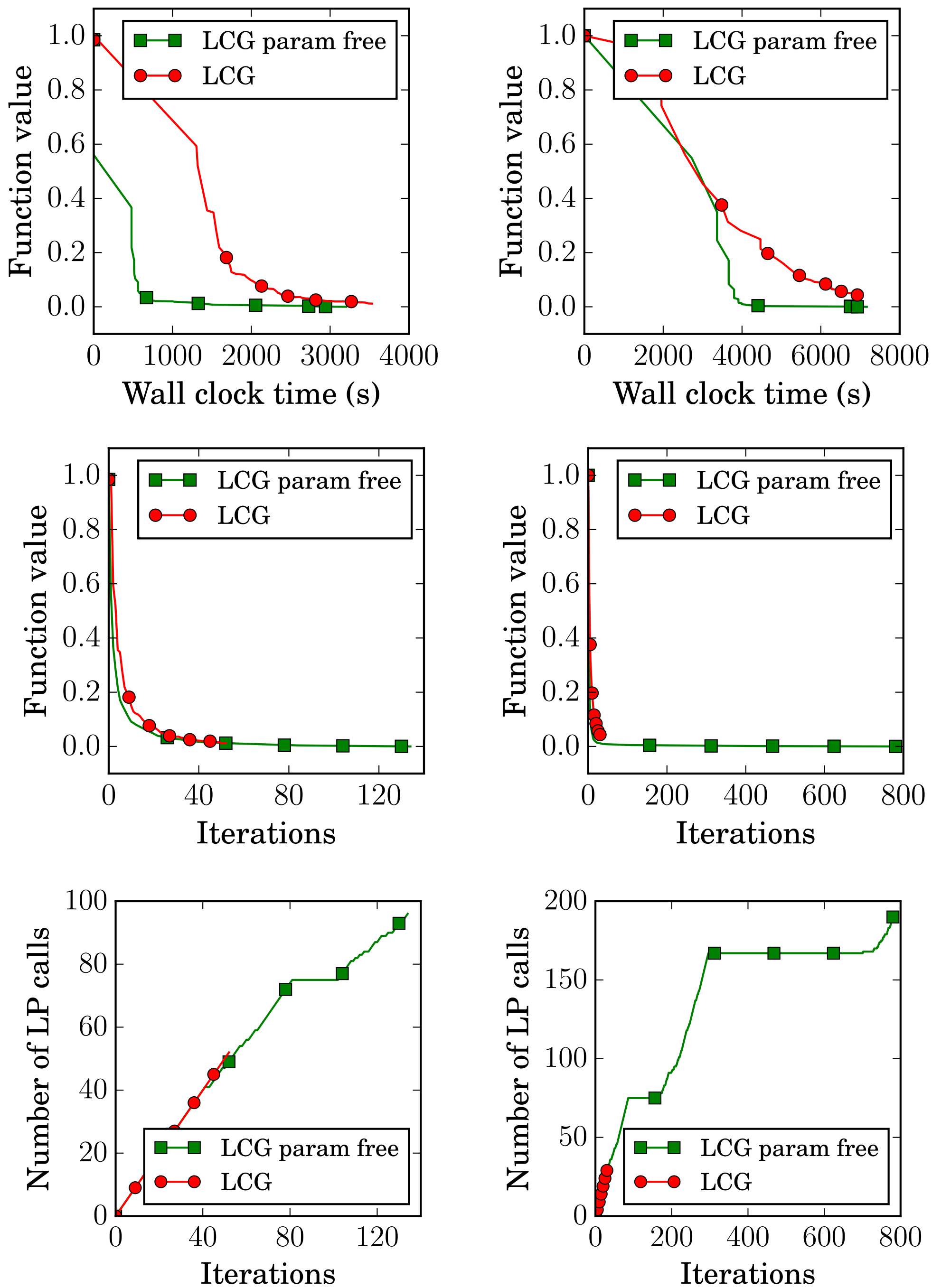


Figure 31. Comparison of the 'textbook' variant of the Lazy CG algorithm (Algorithm 2) vs. the Parameter-free Lazy CG (Algorithm 4) depicted for two sample instances to demonstrate behavior. The parameter-free variant usually has a slighlty improved behavior in terms of iterations and a significantly 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 4.3).