Minding the Gaps for Block Frank-Wolfe Optimization of Structured SVMs

Anton Osokin^{*,1} **Jean-Baptiste Alayrac**^{*,1} FIRST.LASTNAME@INRIA.FR
Isabella Lukasewitz^{1} **Puneet K. Dokania**^{2} **Simon Lacoste-Julien**^{1}
^{1} INRIA – École Normale Supérieure, Paris, France ^{2} INRIA – CentraleSupélec, Châtenay-Malabry, France
* Both authors contributed equally.

Abstract

In this paper, we propose several improvements on the block-coordinate Frank-Wolfe (BCFW) algorithm from Lacoste-Julien et al. (2013) recently used to optimize the structured support vector machine (SSVM) objective in the context of structured prediction, though it has wider applications. The key intuition behind our improvements is that the estimates of block gaps maintained by BCFW reveal the block suboptimality that can be used as an *adaptive* criterion. First, we sample objects at each iteration of BCFW in an adaptive non-uniform way via gap-based sampling. Second, we incorporate pairwise and away-step variants of Frank-Wolfe into the block-coordinate setting. Third, we cache oracle calls with a cache-hit criterion based on the block gaps. Fourth, we provide the first method to compute an approximate regularization path for SSVM. Finally, we provide an exhaustive empirical evaluation of all our methods on four structured prediction datasets.

1. Introduction

One of the most popular learning objectives for structured prediction is the structured support vector machine (Taskar et al., 2003; Tsochantaridis et al., 2005), which generalizes the classical binary SVM to problems with structured outputs. In this paper, we consider the ℓ_2-regularized ℓ_1-slack structured SVM, to which we will simply refer as SSVM. The SSVM method consists in the minimization of the regularized structured hinge-loss on the labeled training set. The optimization problem of SSVM is of significant complexity and, thus, hard to scale up. In the literature, multiple optimization methods have been applied to tackle this problem, including cutting-plane methods (Tsochantaridis et al., 2005; Joachims et al., 2009) and stochastic subgradient methods (Ratliff et al., 2007), among others.

Recently, Lacoste-Julien et al. (2013) proposed the block-coordinate Frank-Wolfe method (BCFW), which is currently one of the state-of-the-art algorithms for SSVM.[1] In contrast to the classical (batch) Frank-Wolfe algorithm (Frank & Wolfe, 1956), BCFW is a randomized block-coordinate method that works on *block-separable* convex compact domains. In the case of SSVM, BCFW operates in the dual domain and iteratively applies Frank-Wolfe steps on the blocks of dual variables corresponding to different objects of the training set. Distinctive features of BCFW for SSVM include optimal step size selection leading to the absence of the step-size parameter, convergence guarantees for the primal objective, and ability to compute the duality gap as a stopping criterion.

Notably, the duality gap obtained by BCFW can be written as a sum of block gaps, where each block of dual variables corresponds to one training example. In this paper, we exploit this property and improve the BCFW algorithm in multiple ways. First, we substitute the standard uniform sampling of objects at each iteration with an adaptive non-uniform sampling. Our procedure consists in sampling objects with probabilities proportional to the values of their block gaps, giving one of the first fully *adaptive* sampling approaches in the optimization literature that we are aware of. This choice of sampling probabilities is motivated by the intuition that objects with higher block gaps potentially can provide more improvement to the objective. We analyze the effects of the gap-based sampling on convergence and discuss the practical trade-offs.

Second, we apply pairwise (Mitchell et al., 1974) and away (Wolfe, 1970) steps of Frank-Wolfe to the block-coordinate setting. This modification is motivated by the fact that batch algorithms based on these steps have linear convergence rates (Lacoste-Julien & Jaggi, 2015) whereas convergence of standard Frank-Wolfe is sublinear.

Third, we cache oracle calls and propose a gap-based criterion for calling the oracle (cache miss vs. cache hit). Caching the oracle calls was shown do deliver significant speed-ups when the oracle is expensive, e.g., in the case of

Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48. Copyright 2016 by the author(s).

[1]Independently, Branson et al. (2013) proposed their SVM-IS algorithm which is equivalent to BCFW in some scenarios.

cutting-plane methods (Joachims et al., 2009).

Finally, we propose an algorithm to approximate the regularization path of SSVM, i.e., solve the problem for all possible values of the regularization parameter. Our method exploits block gaps to construct the breakpoints of the path and leads to an ε-approximate path.

Contributions. Overall, we make the following contributions: (i) adaptive non-uniform sampling of the training objects; (ii) pairwise and away steps in the block-coordinate setting; (iii) gap-based criterion for caching the oracle calls; (iv) regularization path for SSVM. The first three contributions are general to BCFW and thus could be applied to other block-separable optimization problems where BCFW could or have been used such as video co-localization (Joulin et al., 2014), multiple sequence alignment (Alayrac et al., 2016, App. B) or structured submodular optimization (Jegelka et al., 2013), among others.

This paper is organized as follows. In Section 2, we describe the setup and review the BCFW algorithm. In Section 3, we describe our contributions: adaptive sampling (Section 3.1), pairwise and away steps (Section 3.2), caching (Section 3.3). In Section 4, we explain our algorithm to compute the regularization path. We discuss the related work in the relevant sections of the paper. Section 5 contains the experimental study of the methods. The code and datasets are available at our project webpage.[2]

2. Background

2.1. Structured Support Vector Machine (SSVM)

In structured prediction, we are given an input $\boldsymbol{x} \in \mathcal{X}$, and the goal is to predict a structured object $\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})$ (such as a sequence of tags). In the standard setup for structured SVM (SSVM) (Taskar et al., 2003; Tsochantaridis et al., 2005), we assume that prediction is performed with a linear model $h_{\boldsymbol{w}}(\boldsymbol{x}) = \operatorname{argmax}_{\boldsymbol{y}\in\mathcal{Y}(\boldsymbol{x})}\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y})\rangle$ parameterized by the weight vector $\boldsymbol{w}$ where the structured feature map $\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y}) \in \mathbb{R}^d$ encodes the relevant information for input/output pairs. We reuse below the notation and setup from Lacoste-Julien et al. (2013). Given a labeled training set $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$, the parameters $\boldsymbol{w}$ are estimated by solving a convex non-smooth optimization problem

$$\min_{\boldsymbol{w}} \quad \tfrac{\lambda}{2}\|\boldsymbol{w}\|^2 + \tfrac{1}{n}\sum_{i=1}^{n} \tilde{H}_i(\boldsymbol{w}) \tag{1}$$

where λ is the regularization parameter and $\tilde{H}_i(\boldsymbol{w})$ is the structured hinge loss defined using the *loss-augmented decoding* subproblem (or *maximization oracle*):

$$\text{'max oracle'} \quad \tilde{H}_i(\boldsymbol{w}) := \max_{\boldsymbol{y}\in\mathcal{Y}_i} \underbrace{L_i(\boldsymbol{y}) - \langle \boldsymbol{w}, \boldsymbol{\psi}_i(\boldsymbol{y})\rangle}_{=:\,H_i(\boldsymbol{y};\boldsymbol{w})}. \tag{2}$$

Here $\boldsymbol{\psi}_i(\boldsymbol{y}) := \boldsymbol{\phi}(\boldsymbol{x}_i,\boldsymbol{y}_i) - \boldsymbol{\phi}(\boldsymbol{x}_i,\boldsymbol{y})$, $\mathcal{Y}_i := \mathcal{Y}(\boldsymbol{x}_i)$, and $L_i(\boldsymbol{y}) := L(\boldsymbol{y}_i,\boldsymbol{y})$ denotes the task-dependent structured error of predicting an output $\boldsymbol{y}$ instead of the observed output $\boldsymbol{y}_i$ (e.g., a Hamming distance between two sequences).

[2] http://www.di.ens.fr/sierra/research/gapBCFW/

Algorithm 1 Block-Coordinate Frank-Wolfe (BCFW) algorithm for structured SVM

```
1:  Let $\boldsymbol{w}^{(0)} := \boldsymbol{w}_i^{(0)} := \mathbf{0}$;  $\ell^{(0)} := \ell_i^{(0)} := 0$
2:  for $k := 0,\dots,\infty$ dos
3:      Pick $i$ at random in $\{1,\dots,n\}$
4:      Solve $\boldsymbol{y}_i^* := \operatorname{argmax}_{\boldsymbol{y}\in\mathcal{Y}_i} H_i(\boldsymbol{y};\boldsymbol{w}^{(k)})$
5:      Let $\boldsymbol{w_s} := \frac{1}{\lambda n}\boldsymbol{\psi}_i(\boldsymbol{y}_i^*)$ and $\ell_{\boldsymbol{s}} := \frac{1}{n}L_i(\boldsymbol{y}_i^*)$
6:      Let $g_i^{(k)} := \lambda(\boldsymbol{w}_i^{(k)} - \boldsymbol{w_s})^\mathsf{T}\boldsymbol{w}^{(k)} - \ell_i^{(k)} + \ell_{\boldsymbol{s}}$
7:      Let $\gamma := \frac{g_i^{(k)}}{\lambda\|\boldsymbol{w}_i^{(k)} - \boldsymbol{w_s}\|^2}$ and clip to $[0,1]$
8:      Update $\boldsymbol{w}_i^{(k+1)} := (1-\gamma)\boldsymbol{w}_i^{(k)} + \gamma\,\boldsymbol{w_s}$
9:          and $\ell_i^{(k+1)} := (1-\gamma)\ell_i^{(k)} + \gamma\,\ell_{\boldsymbol{s}}$
10:     Update $\boldsymbol{w}^{(k+1)} := \boldsymbol{w}^{(k)} + \boldsymbol{w}_i^{(k+1)} - \boldsymbol{w}_i^{(k)}$
11:         and $\ell^{(k+1)} := \ell^{(k)} + \ell_i^{(k+1)} - \ell_i^{(k)}$
12: end for
```

Dual formulation. The negative of a Fenchel dual for objective (1) can be written as

$$\min_{\substack{\boldsymbol{\alpha}\in\mathbb{R}^m\\ \boldsymbol{\alpha}\succcurlyeq 0}} \quad f(\boldsymbol{\alpha}) := \tfrac{\lambda}{2}\big\|A\boldsymbol{\alpha}\big\|^2 - \boldsymbol{b}^\mathsf{T}\boldsymbol{\alpha} \qquad \text{s.t.} \quad \textstyle\sum_{\boldsymbol{y}\in\mathcal{Y}_i}\alpha_i(\boldsymbol{y}) = 1 \quad \forall i\in[n] \tag{3}$$

where $\alpha_i(\boldsymbol{y})$, $i \in [n] := \{1,\dots,n\}$, $\boldsymbol{y} \in \mathcal{Y}_i$ are the dual variables. The matrix $A \in \mathbb{R}^{d\times m}$ consists of the $m := \sum_i m_i = \sum_i |\mathcal{Y}_i|$ columns $A := \big\{\frac{1}{\lambda n}\boldsymbol{\psi}_i(\boldsymbol{y}) \in \mathbb{R}^d \;\big|\; i\in[n], \boldsymbol{y}\in\mathcal{Y}_i\big\}$, and the vector $\boldsymbol{b} \in \mathbb{R}^m$ is given by $\boldsymbol{b} := \big(\frac{1}{n}L_i(\boldsymbol{y})\big)_{i\in[n],\boldsymbol{y}\in\mathcal{Y}_i}$.

In SSVM (as for the standard SVM), the Karush-Kuhn-Tucker (KKT) optimality conditions can give the primal variables $\boldsymbol{w}(\boldsymbol{\alpha}) = A\boldsymbol{\alpha} = \sum_{i,\,\boldsymbol{y}\in\mathcal{Y}_i}\alpha_i(\boldsymbol{y})\frac{\boldsymbol{\psi}_i(\boldsymbol{y})}{\lambda n}$ corresponding to the dual variables $\boldsymbol{\alpha}$ (see, e.g., (Lacoste-Julien et al., 2013, App. E)). The gradient of f then takes a simple form $\nabla f(\boldsymbol{\alpha}) = \lambda A^\mathsf{T}A\boldsymbol{\alpha} - \boldsymbol{b} = \lambda A^\mathsf{T}\boldsymbol{w} - \boldsymbol{b}$; its $(i,\boldsymbol{y})$-th component equals $-\frac{1}{n}H_i(\boldsymbol{y};\boldsymbol{w})$.

2.2. Block Coordinate Frank-Wolfe method (BCFW)

We give in Alg. 1 the BCFW algorithm from Lacoste-Julien et al. (2013) applied to problem (3). It exploits the block-separability of the domain $\mathcal{M} := \Delta_{|\mathcal{Y}_1|} \times \ldots \times \Delta_{|\mathcal{Y}_n|}$ for problem (3) and sequentially applies the Frank-Wolfe steps to the blocks of the dual variables $\boldsymbol{\alpha}_{(i)} \in \mathcal{M}^{(i)} := \Delta_{|\mathcal{Y}_i|}$.

While BCFW works on the dual (3) of SSVM, it only maintains explicitly the primal variables via the relationship $\boldsymbol{w}(\boldsymbol{\alpha})$. Most importantly, the Frank-Wolfe linear oracle on block i at iterate $\boldsymbol{\alpha}^{(k)}$ is equivalent to the max oracle (2) at the corresponding weight vector $\boldsymbol{w}^{(k)} := A\boldsymbol{\alpha}^{(k)}$ (Lacoste-Julien et al., 2013, App. B.1) (see line 4 of Alg. 1):

$$\max_{\boldsymbol{s}_{(i)}\in\mathcal{M}^{(i)}} \big\langle \boldsymbol{s}_{(i)}, -\nabla_{(i)} f(\boldsymbol{\alpha}^{(k)})\big\rangle = \tfrac{1}{n}\max_{\boldsymbol{y}\in\mathcal{Y}_i} H_i(\boldsymbol{y};\boldsymbol{w}^{(k)}). \tag{4}$$

Here, the operator $\nabla_{(i)}$ denotes the partial gradient corresponding to the block i, i.e., $\nabla f = (\nabla_{(i)} f)_{i=1}^n$. Note that each argmax of the r.h.s. of (4), $\boldsymbol{y}^*_{(i)}$, corresponds to a corner $\boldsymbol{s}^*_{(i)}$ of the polytope $\mathcal{M}^{(i)}$ maximizing the l.h.s. of (4).

As the objective (3) is quadratic, the optimal step size that yields the maximal improvement in the chosen direction $\boldsymbol{s}^*_{(i)} - \boldsymbol{\alpha}^{(k)}_{(i)}$ can be found analytically (Line 7 of Alg. 1).

2.3. Duality gap

At each iteration, the batch Frank-Wolfe algorithm (Frank & Wolfe, 1956), (Lacoste-Julien et al., 2013, Section 3) computes the following quantity, known as the *linearization duality gap* or *Frank-Wolfe gap*:

$$g(\boldsymbol{\alpha}) := \max_{\boldsymbol{s} \in \mathcal{M}} \langle \boldsymbol{\alpha} - \boldsymbol{s}, \nabla f(\boldsymbol{\alpha}) \rangle = \langle \boldsymbol{\alpha} - \boldsymbol{s}^*, \nabla f(\boldsymbol{\alpha}) \rangle. \quad (5)$$

It turns out that this Frank-Wolfe gap exactly equals the Lagrange duality gap between the dual objective (3) at a point $\boldsymbol{\alpha}$ and the primal objective (1) at the point $\boldsymbol{w}(\boldsymbol{\alpha}) = A\boldsymbol{\alpha}$ (Lacoste-Julien et al., 2013, App. B.2).

Because of the separability of $\mathcal{M}$, the Frank-Wolfe gap (5) can be represented here as a sum of block gaps $g_i(\boldsymbol{\alpha})$, $g(\boldsymbol{\alpha}) = \sum_{i=1}^n g_i(\boldsymbol{\alpha})$, where

$$g_i(\boldsymbol{\alpha}) := \max_{\boldsymbol{s}_{(i)} \in \mathcal{M}^{(i)}} \left\langle \boldsymbol{\alpha}_{(i)} - \boldsymbol{s}_{(i)}, \nabla_{(i)} f(\boldsymbol{\alpha}) \right\rangle. \quad (6)$$

Block gaps can be easily computed using the quantities maintained by Alg. 1 (see line 6).

Finally, we can rewrite the block gap in the form

$$g_i(\boldsymbol{\alpha}) = \tfrac{1}{n}\left(\max_{\boldsymbol{y} \in \mathcal{Y}_i} H_i(y; \boldsymbol{w}) - \sum_{\boldsymbol{y} \in \mathcal{Y}_i} \alpha_i(\boldsymbol{y}) H_i(\boldsymbol{y}; \boldsymbol{w}) \right) \quad (7)$$

providing understandable intuition of when the block gap equals zero. This is the case when all the support vectors, i.e., labelings corresponding to $\alpha_i(\boldsymbol{y}) > 0$, are tied solutions of the max oracle (4).

2.4. Convergence of BCFW

Lacoste-Julien et al. (2013) prove the convergence of the BCFW algorithm at a rate $\mathcal{O}(\frac{1}{k})$.

Theorem 1 (Lacoste-Julien et al. (2013), Theorem 2)**.** *For each $k \geq 0$, the iterate*[3] *$\boldsymbol{\alpha}^{(k)}$ of Alg. 1 satisfies $\mathbf{E}\big[f(\boldsymbol{\alpha}^{(k)})\big] - f(\boldsymbol{\alpha}^*) \leq \frac{2n}{k+2n}\big(C_f^{\otimes} + h_0\big)$, where $\boldsymbol{\alpha}^* \in \mathcal{M}$ is a solution of the problem (3), $h_0 := f(\boldsymbol{\alpha}^{(0)}) - f(\boldsymbol{\alpha}^*)$ is the suboptimality at the starting point of the algorithm, $C_f^{\otimes} := \sum_{i=1}^n C_f^{(i)}$ is the sum of the curvature constants*[4] *of f with respect to the domains $\mathcal{M}^{(i)}$ of individual blocks. The expectation is taken over the random choice of the block i at iterations $1, \ldots, k$ of the algorithm.*

[3]Note that Alg. 1 does not maintain iterates $\boldsymbol{\alpha}^{(k)}$ explicitly. They are stored in the form of $\boldsymbol{w}^{(k)} = A\boldsymbol{\alpha}^{(k)}$.

[4]For the definition of curvature constant, see Definition 2 in App. B or (Lacoste-Julien & Jaggi, 2015, App. A)

The proof of Theorem 1 crucially depends on a standard *descent lemma* applied to a block, stating that at each iteration of BCFW, for any picked block i and any scalar $\gamma \in [0, 1]$, the following inequality holds:

$$f(\boldsymbol{\alpha}^{(k+1)}) \leq f(\boldsymbol{\alpha}^{(k)}) - \gamma g_i(\boldsymbol{\alpha}^{(k)}) + \tfrac{\gamma^2}{2} C_f^{(i)}. \quad (8)$$

We rederive inequality (8) as Lemma 3 in App. B. Note that $\boldsymbol{\alpha}^{(k+1)} \in \mathcal{M}$ is defined by a line search, which is why the bound (8) holds for any scalar $\gamma \in [0, 1]$.

Taking the expectation of (8) w.r.t. the random choice of block i (sampled uniformly on $[n]$), we get the inequality

$$\mathbf{E}\big[f(\boldsymbol{\alpha}^{(k+1)}) \,|\, \boldsymbol{\alpha}^{(k)}\big] \leq f(\boldsymbol{\alpha}^{(k)}) - \tfrac{\gamma}{n} g(\boldsymbol{\alpha}^{(k)}) + \tfrac{\gamma^2}{2n} C_f^{\otimes} \quad (9)$$

which can be used to get the convergence theorem.

3. Block gaps in BCFW

In this section, we propose three ways to improve the BCFW algorithm: adaptive sampling (Sec. 3.1), pairwise and away steps (Sec. 3.2) and caching (Sec. 3.3).

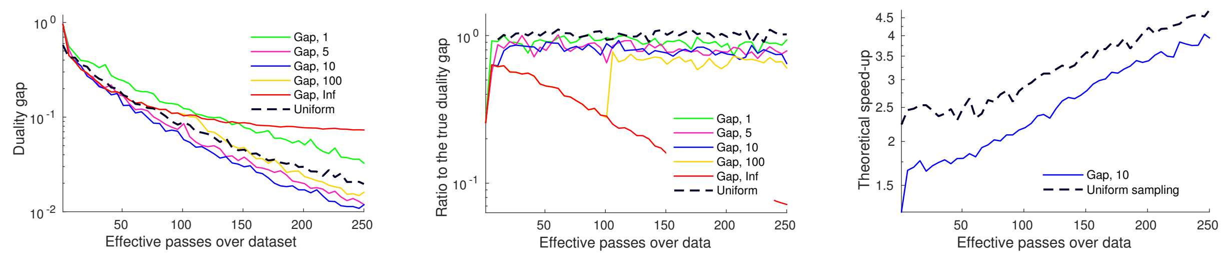
3.1. Adaptive non-uniform sampling

Motivation. When optimizing finite sums such as (1), it is often the case that processing some summands does not lead to significant progress of the algorithm. At each iteration, the BCFW algorithm selects a training object and performs the block-coordinate step w.r.t. the corresponding dual variables. If these variables are already close to being optimal, then BCFW does not make significant progress at this iteration. Usually, it is hard to identify whether processing the summand would lead to an improvement without actually doing computations on it. The BCFW algorithm obtains at each iteration the block gap (6) quantifying the suboptimality on the block. In what follows, we use the block gaps to randomly choose a block (an object of the training set) at each iteration in such a way that the blocks with larger suboptimality are sampled more often (the sampling probability of a block is proportional to the value of the current gap estimate).

Convergence. Assume that at iteration k of Alg. 1, we have the probability $p_i^{(k)}$ of sampling block i. By minimizing the descent lemma bound (8) w.r.t. γ for each i independently under the assumption that $g_i(\boldsymbol{\alpha}^{(k)}) \leq C_f^{(i)}$, and then taking the conditional expectation w.r.t. i, we get

$$\mathbf{E}\big[f(\boldsymbol{\alpha}^{(k+1)}) \,|\, \boldsymbol{\alpha}^{(k)}\big] \leq f(\boldsymbol{\alpha}^{(k)}) - \tfrac{1}{2} \sum_{i=1}^n p_i^{(k)} \tfrac{g_i^2(\boldsymbol{\alpha}^{(k)})}{C_f^{(i)}}. \quad (10)$$

Intuitively, by adapting the probabilities $p_i^{(k)}$, we can obtain a better bound on the expected improvement of f. In the ideal scenario, one would choose deterministically the block i with the maximal value of $g_i^2(\boldsymbol{\alpha}^{(k)})/C_f^{(i)}$.



(a) Convergence plots (b) Quality of gap estimates (c) Theoretical improvement

Figure 1. Plot (a) shows exploitation/staleness trade-off for the gap sampling approach. We report the duality gap against the number of effective passes over the data for uniform sampling and for gap sampling with the different frequencies of batch passes updating the gap estimates (every pass over data, every 5, 10, 100 passes, no batch updates). Plot (b) shows the quality of heuristic gap estimates obtained by the same methods. We report the ratio of the heuristic gap estimate to the true gap value. Plot (c) shows the factor of improvement of exact gap sampling predicted by Theorem 2 for real gaps appearing during a run of BCFW with either uniform or gap sampling.

In practice, the curvature $C_f^{(i)}$ is unknown, and having access to all $g_i(\boldsymbol{\alpha}^{(k)})$'s at each step is prohibitively expensive. However, the values of the block gaps obtained at the previous iterations can serve as estimates of the block gaps at the current iteration. We use them in the following *non-uniform* gap sampling scheme: $p_i^{(k)} \propto g_i(\boldsymbol{\alpha}^{(k_i)})$. where k_i records the last iteration at which the gap i was computed. Alg. 2 in App. D summarizes the method.

We also motivate this choice by Theorem 2 below which shows that BCFW with (exact) gap sampling converges with a better constant in the rate than BCFW with uniform sampling when the gaps are non-uniform enough (and is always better when the curvatures $C_f^{(i)}$'s are uniform). See the proof and discussion in App. E.

Theorem 2. *Consider the same notation as in Theorem 1. Assume that at each iterate $\boldsymbol{\alpha}^{(k)}$, BCFW with gap sampling (Alg. 2) has access to the exact values of the block gaps. Then, at each iteration, it holds that* $\mathbf{E}\big[f(\boldsymbol{\alpha}^{(k)})\big] - f(\boldsymbol{\alpha}^*) \leq \frac{2n}{k+2n}\big(C_f^{\otimes}\chi^{\otimes} + h_0\big)$ *where the constant $\chi^{\otimes}$ is an upper bound on* $\mathbf{E}\Big[\frac{\chi(C_f^{(:)})}{\chi(\boldsymbol{g}_{:}(\boldsymbol{\alpha}^{(k)}))^3}\Big]$. *The* non-uniformity measure *$\chi(\boldsymbol{x})$ of a vector $\boldsymbol{x} \in \mathbb{R}_+^n$ is defined as* $\chi(\boldsymbol{x}) := \sqrt{1 + n^2 \operatorname{Var}[\boldsymbol{p}]}$ *where $\boldsymbol{p} := \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_1}$ is the probability vector obtained by normalizing $\boldsymbol{x}$.*

Adaptive procedure. Note that this procedure is *adaptive*, meaning that the criterion for choosing an object to optimize changes during the optimization process. Our adaptive approach differs from more standard techniques that sample proportional to the Lipschitz constants, as e.g., in Nesterov (2012). In App. C, we illustrate the advantage of this property by constructing an example where the convergence of gap sampling can be shown *tightly* to be n times faster than when using Lipschitz sampling.

Exploitation versus staleness trade-off. In practice, having access to the exact block gaps is intractable because it requires a full pass over the dataset after every block update. However, we have access to the estimates of the block gaps computed from past oracle calls on each block. Notice that such estimates are outdated, i.e., might be quite far from the current values of the block gaps. We call this effect "staleness". One way to compensate staleness is to refresh the block gaps by doing a full gap computation (a pass over the dataset) after several block-coordinate passes. These gap computations were often already done during the optimization process, e.g., to monitor convergence.

We demonstrate the exploitation/staleness trade-off in our exploratory experiment reported in Figure 1. On the OCR dataset (Taskar et al., 2003), we run the gap sampling algorithm with a gap computation pass after 1, 5, 10 and 100 block-coordinate passes (Gap 1, Gap 5, Gap 10, Gap 100) and without any gap computation passes (Gap Inf). As a baseline, we use BCFW with uniform sampling (Uniform). Figure 1a reports the duality gap after each number of effective passes over the data.[5] Figure 1b shows the ratio of the exact value of the duality gap to the heuristic gap estimate defined as the sum of the current gap estimates. We observe that when the gap computation is never run, the gap becomes significantly underestimated and the algorithm does not converge. On another extreme, when performing the gap computation after each pass of BCFW, the algorithm wastes too many computations and converges slowly. Between the two extremes, the method is not very sensitive to the parameter (we have tried 5, 10, 20, 50) allowing us to always use the value of 10.

Comparing adaptive methods to BCFW with uniform sampling, we observe a faster convergence. Figure 1c reports the improvement of gap sampling at each iteration w.r.t. uniform sampling that is predicted by Theorem 2. Specifically, we report the quantity $\chi(\boldsymbol{g}_{:}(\boldsymbol{\alpha}^{(k)}))^3 / \chi(C_f^{(:)})$ with the block gaps estimated at the runs of BCFW with both uniform and gap sampling schemes. To estimate the curvature constants $C_f^{(i)}$, we use the upper bounds proposed by Lacoste-Julien et al. (2013, App. A): $\frac{4R_i^2}{\lambda n^2}$ where $R_i := \max_{\boldsymbol{y}\in\mathcal{Y}_i} \|\boldsymbol{\psi}_i(\boldsymbol{y})\|_2$. We approximate R_i by picking the largest value $\|\boldsymbol{\psi}_i(\boldsymbol{y})\|_2$ corresponding to a labeling $\boldsymbol{y}$ observed within the run of BCFW.

[5] An effective pass consists in n calls to the max oracle.

Related work. Non-uniform sampling schemes have been used over the last few years to improve the convergence rates of well known randomized algorithms (Nesterov, 2012; Needell et al., 2014; Zhao & Zhang, 2015). Most of these approaches use the Lipschitz constants of the gradients to sample more often functions for which gradient changes quickly. This approach has two main drawbacks. First, Lipschitz constants are often unknown and heuristics are needed to estimate them. Second, such schemes are not adaptive to the current progress of the algorithm. To the best of our knowledge, the only other approach that uses an *adaptive* sampling scheme to guide the optimization with convergence guarantees is the one from Csiba et al. (2015), in the context of the stochastic dual coordinate ascent (SDCA) algorithm. A cyclic version of BCFW has been analyzed by Beck et al. (2015) while Wang et al. (2014) analyzed its mini-batch form.

3.2. Pairwise and away steps

Motivation. In the batch setting, the convergence rate of the Frank-Wolfe algorithm is known to be sublinear when the solution is on the boundary (Wolfe, 1970), as is the case for SSVM. Several modifications have been proposed in the literature to address this issue. All these methods replace (or complement) the FW step with a step of another type: pairwise step (Mitchell et al., 1974), away step (Wolfe, 1970), fully-corrective step (Holloway, 1974) (see Lacoste-Julien & Jaggi (2015) for a recent review and the proof that all these methods have a linear rate on the objective (3) despite not being strongly convex). A common feature of these methods is the ability to remove elements of the active set (support vectors in the case of SSVM) in order to reach the boundary, unlike FW which oscillates while never completely reaching the boundary. As we expect the solution of SSVM to be sparse, these variants seem natural in our setting. In the rest of this section, we present the pairwise steps in the block-coordinate setting (the away-step version is described in Alg. 4 of App. 4).

Pairwise steps. A (block) pairwise step consists in *removing* mass from the *away corner* on block i and transferring it to the *FW corner* obtained by the max oracle (4). The away corner is the element of the active set $\mathcal{S}_i := \{\boldsymbol{y} \in \mathcal{Y}_i \mid \alpha_i(\boldsymbol{y}) > 0\} \subseteq \mathcal{Y}_i$ worst aligned with the current descent direction, which can be found by solving $\boldsymbol{y}_i^{\boldsymbol{a}} := \operatorname{argmin}_{\boldsymbol{y} \in \mathcal{S}_i} H_i(\boldsymbol{y}; \boldsymbol{w})$. This does not require solving a combinatorial optimization problem because the size of the active set is typically small, e.g., bounded by the number of iterations performed on the block i. Analogously to the case of BCFW, the optimal step size γ for the pairwise step can be computed explicitly by clipping $\frac{\lambda(\boldsymbol{w_a}-\boldsymbol{w_s})^\mathsf{T}\boldsymbol{w}^{(k)}+\ell_{\boldsymbol{s}}-\ell_{\boldsymbol{a}}}{\lambda\|\boldsymbol{w_a}-\boldsymbol{w_s}\|^2}$ to the segment $[0, \alpha_i^{(k)}(\boldsymbol{y}_i^{\boldsymbol{a}})]$ where the upper bound $\alpha_i^{(k)}(\boldsymbol{y}_i^{\boldsymbol{a}})$ corresponds to the mass of the away corner before the step and the quantities $\boldsymbol{w_s} := \frac{1}{\lambda n}\boldsymbol{\psi}_i(\boldsymbol{y}_i^*)$, $\ell_{\boldsymbol{s}} := \frac{1}{n}L_i(\boldsymbol{y}_i^*)$ and $\boldsymbol{w_a} := \frac{1}{\lambda n}\boldsymbol{\psi}_i(\boldsymbol{y}_i^a)$, $\ell_{\boldsymbol{a}} := \frac{1}{n}L_i(\boldsymbol{y}_i^a)$ represent the FW and away corners. Alg. 3 in App. D summarizes the block-coordinate pairwise Frank-Wolfe (BCPFW) algorithm.

In contrast to BCFW, the steps of BCPFW cannot be expressed in terms of the primal variables $\boldsymbol{w}$ only, thus it is required to explicitly store the dual variables $\boldsymbol{\alpha}_i$. Storing the dual variables is feasible, because they are extremely sparse, but still can lead to computational overheads caused by the maintenance of the data structure.

The standard convergence analysis for pairwise and away-step FW cannot be easily extended to BCFW. We show the geometric decrease of the objective in Theorem 4 of App. G only when *no* block would have a drop step (a.k.a. 'bad step'); a condition that cannot be easily analyzed due to the randomization of the algorithm. We believe that novel proof techniques are required here, even though we did observe empirically a linear convergence rate when λ is big enough.

Related work. Ñanculef et al. (2014, Alg. 4) used the pairwise FW algorithm on the dual of binary SVM (in batch mode, however). It is related to classical working set algorithms, such as the SMO algorithm used to train SVMs (Platt, 1999), also already applied on SSVMs in Taskar (2004, Ch. 6). Franc (2014) recently proposed a version of pairwise FW for the block-coordinate setting. Their SDA-WSS2 algorithm uses a different criterion for choosing the away corner than BCPFW: instead of minimizing H_i over the active set $\mathcal{S}_i$, they compute the improvement for all possible away corners and pick the best one. Their FASOLE algorithm also contains a version of gap sampling in the form of variable shrinking: if a block gap becomes small enough, the block is not visited again, until all the counters are reset.

3.3. Caching

Motivation. At each step, the BCFW and BCPFW algorithms call the max oracle to find the Frank-Wolfe corner. In cases where the max oracle is expensive, this step becomes a computational bottleneck. A natural idea to overcome this problem consists in using a "cheaper oracle" most of the time hoping that the resulting corner would be good enough. Caching the results of the max oracle implements this idea by reusing the previous calls of the max oracle to store potentially promising corners.

Caching. The main principle of caching consists in maintaining a working set $\mathcal{C}_i \subset \mathcal{Y}_i$ of labelings/corners for each block i, where $|\mathcal{C}_i| \ll |\mathcal{Y}_i|$. A *cache oracle* obtains the *cache corner* defined as a corner from the working set best aligned with the descent direction, i.e., $\boldsymbol{y}_i^c := \operatorname{argmax}_{\boldsymbol{y} \in \mathcal{C}_i} H_i(\boldsymbol{y}; \boldsymbol{w})$. If the obtained cache corner passes a *cache hit criterion*, i.e., there is a *cache hit*, we do a Frank-Wolfe (or pairwise) step based on the cache corner. A step defined this way is equivalent to the corresponding step on the convex hull of the working set, which is a subset of the block domain $\mathcal{Y}_i$. If a cache hit criterion is not satisfied, i.e., there is a *cache miss*, we call the (possibly expensive) max oracle to obtain a Frank-Wolfe corner over the full domain $\mathcal{Y}_i$. Alg. 5 in App. D summarizes the BCFW method with caching.

Note that, in the case of BCPFW, the working set $\mathcal{C}_i$ is closely related to the active set $\mathcal{S}_i$. On the implementation side, we maintain both sets in the same data structure and keep $\mathcal{S}_i \subseteq \mathcal{C}_i$.

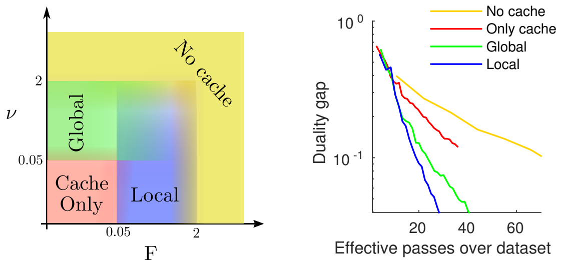
Cache hit criterion. An important part of a caching scheme is the criterion deciding whether the cache look up is sufficient or the max oracle needs to be called. Intuitively, we want to use the cache whenever it allows optimization to make large enough progress. We use as measure of potential progress the inner product between the candidate direction and the negative gradient (which would give the block gap g_i (6) if the FW corner is used). For a cache step, it gives $\hat{g}_i^{(k)} := \lambda(\boldsymbol{w}_i^{(k)} - \boldsymbol{w}_{\boldsymbol{c}})^\mathsf{T}\boldsymbol{w}^{(k)} - \ell_i{}^{(k)} + \ell_{\boldsymbol{c}}$, which is defined by quantities $\boldsymbol{w}_{\boldsymbol{c}} = \frac{\boldsymbol{\psi}_i(\boldsymbol{y}_i^c)}{\lambda n}$, $\ell_{\boldsymbol{c}} = \frac{1}{n}L_i(\boldsymbol{y}_i^c)$ similar to the ones defining the block gap. The quantity $\hat{g}_i^{(k)}$ is then compared to a *cache hit threshold* defined as $\max(Fg_i^{(k_i)}, \frac{\nu}{n}g^{(k_0)})$ where k_i identifies the iteration when the max oracle was last called for the block i, k_0 is the index of the iteration when the full batch gap was computed, $F > 0$ and $\nu > 0$ are cache parameters.

The following theorem gives a safety convergence result for BCFW with caching (see App. F for the proof).

Theorem 3. *Consider the same notation as in Theorem 1. Let* $\tilde{\nu} := \frac{1}{n}\nu \leq 1$. *The iterate* $\boldsymbol{\alpha}^{(k)}$ *of Alg. 5 satisfies* $\mathbf{E}\big[f(\boldsymbol{\alpha}^{(k)})\big] - f(\boldsymbol{\alpha}^*) \leq \frac{2n}{\tilde{\nu}k+2n}\big(\frac{1}{\tilde{\nu}}C_f^{\otimes} + h_0\big)$ *for* $k \geq 0$.

Note that the convergence rate of Theorem 3 differs from the original rate of BCFW (Theorem 1) by the constant $\tilde{\nu}$. If $\tilde{\nu}$ equals one the rate is the same, but the criterion effectively prohibits cache hits. If $\tilde{\nu} < 1$ then the convergence is slower, meaning that the method with cache needs more iterations to converge, but the oracles calls might be cheaper because of the cache hits.

Effect of F and ν. The parameter ν controls the global component and acts as a safety parameter to ensure convergence (Theorem 3). The parameter F controls, instead, the local (block-dependent) component of the criterion. Figure 2 illustrates the effect of the parameters on OCR dataset (Taskar et al., 2003) and motivates their choice. At one extreme, if either F or ν are too large the cache is almost never hit. At another extreme, if both values are small the cache is hit almost always, thus the method almost stops calling the oracle and does not converge. Between the two extremes, one of the components usually dominates. We observe empirically that the regime with the local component dominating leads to faster convergence. Our experiments show that the method is not very sensitive to the choice of the parameters, so, in what follows, we use values $F = 0.25$ and $\nu = 0.01$.



(a) Regimes of the cache (b) Convergence plots

Figure 2. Plot (a) illustrates different regimes induced by the cache parameters F and ν. Plot (b) shows the evolution of the duality gap within BCFW with gap sampling and with cache parameters in different regimes.

Related work. In the context of SSVM, the idea of caching was successfully applied to the cutting plane methods by Joachims et al. (2009), and, recently, to BCFW by Shah et al. (2015). In contrast to Shah et al. (2015), our method chooses whether to call the oracle or to use the cache in an adaptive way by looking at the gap estimates of the current blocks. In the extreme case, when just one block is hard and requires computation and all the rest are easy, our method would be able to call an oracle on the hard block and to use the cache everywhere else. This will result to n times less oracle calls, compared to their strategy.

4. Regularization path

According to the definition of Efron et al. (2004), a regularization path is a set of minimizers of a regularized objective in the form of (1) for all possible values of the regularization parameter λ. Similarly to LASSO and binary SVM, the general result of Rosset & Zhu (2007, Proposition 1) is applicable to the case of SSVM and implies that the exact regularization path is piecewise linear in $1/\lambda$. However, recovering the exact path is, up to our knowledge, intractable in the case of SSVM. In this paper, we construct an ε-approximate regularization path, meaning that, for each feasible λ, we have a corresponding primal variables $\boldsymbol{w}$ which is ε-approximate, i.e., the suboptimality $f_\lambda(\boldsymbol{w}) - f_\lambda^*$ does not exceed ε. We use a piecewise *constant* approximation except for the first piece which is linear. The approximation is represented by a set of breakpoints $\{\lambda_j\}_{j=0}^{J+1}$, $\lambda_0 = +\infty$, $\lambda_{J+1} = 0$, $\lambda_{j+1} \leq \lambda_j$, and a set of parameter vectors $\{\boldsymbol{w}^j\}_{j=1}^{J}$ with the following properties: for each $\lambda \in [\lambda_{j+1}, \lambda_j]$, $j \geq 1$, the vector $\boldsymbol{w}^j$ is ε-approximate; for $\lambda \geq \lambda_1$, the vector $\frac{\lambda_1}{\lambda}\boldsymbol{w}^1$ is ε-approximate.

Our algorithm consists of two steps: (1) at the initialization step, we find the maximal finite breakpoint $\lambda^\infty := \lambda_1$ and the vector $\boldsymbol{w}^\infty := \boldsymbol{w}^1$; (2) at the induction step, we compute a value λ_{j+1} and a vector $\boldsymbol{w}^{j+1}$ given quantities λ_j and $\boldsymbol{w}^j$. At both steps of our algorithm, we explicitly maintain dual variables $\boldsymbol{\alpha}$ that correspond to $\boldsymbol{w}$. Alg. 7 in App. D presents the complete procedure.

Initialization of the regularization path. First, note that, for $\lambda = \infty$, the KKT conditions for (1) and (3) imply that $\boldsymbol{w} = \mathbf{0}$ is a solution of the problem (1). In what follows, we provide a finite value for λ^∞ and explicitly construct $\boldsymbol{\alpha}^\infty$ and $\boldsymbol{w}^\infty$ such that $\frac{\lambda^\infty}{\lambda}\boldsymbol{w}^\infty$ is ε-approximate for $\lambda \geq \lambda^\infty$.

Let $\tilde{\boldsymbol{y}}_i = \operatorname{argmax}_{\boldsymbol{y}\in\mathcal{Y}_i} H_i(\boldsymbol{y};\mathbf{0}) = \operatorname{argmax}_{\boldsymbol{y}\in\mathcal{Y}_i} L_i(\boldsymbol{y})$ be the output of the max oracle for $\boldsymbol{w} = \mathbf{0}$. First, we construct a dual point $\boldsymbol{\alpha}^\infty \in \mathcal{M}$ by setting $\alpha_i^\infty(\tilde{\boldsymbol{y}}_i) = 1$. For any value of λ^∞, the corresponding weight vector can be easily computed: $\boldsymbol{w}^\infty = \frac{1}{\lambda^\infty n}\sum_{i=1}^n \boldsymbol{\psi}_i(\tilde{\boldsymbol{y}}_i)$. Identity (7) provides the duality gap:

$$g(\boldsymbol{\alpha}^\infty, \lambda^\infty, \boldsymbol{w}^\infty) = \frac{1}{n}\sum_{i=1}^n \Big(\max_{\boldsymbol{y}\in\mathcal{Y}_i} \big(L_i(\boldsymbol{y}) - \langle \boldsymbol{w}^\infty, \boldsymbol{\psi}(\boldsymbol{y})\rangle\big) - L_i(\tilde{\boldsymbol{y}}_i) + \langle(\boldsymbol{w}^\infty, \boldsymbol{\psi}(\tilde{\boldsymbol{y}}_i)\rangle\Big).$$

The inequality $\max_x(f(x) + g(x)) \leq \max_x f(x) + \max_x g(x)$ and the equality $\max_{\boldsymbol{y}\in\mathcal{Y}_i} L_i(\boldsymbol{y}) = L_i(\tilde{\boldsymbol{y}}_i)$ bound the gap:

$$g(\boldsymbol{\alpha}^\infty, \lambda^\infty, \boldsymbol{w}^\infty) \leq \frac{1}{n}\sum_i \Big(\max_{\boldsymbol{y}\in\mathcal{Y}_i}(-\langle \boldsymbol{w}^\infty, \boldsymbol{\psi}(\boldsymbol{y})\rangle) + \langle \boldsymbol{w}^\infty, \boldsymbol{\psi}(\tilde{\boldsymbol{y}}_i)\rangle\Big) = \frac{1}{n\lambda^\infty}\sum_{i=1}^n \theta_i + \frac{1}{\lambda^\infty}\left\|\tilde{\boldsymbol{\psi}}\right\|^2$$

where the quantities $\theta_i = \max_{\boldsymbol{y}\in\mathcal{Y}_i}\big(-\langle\tilde{\boldsymbol{\psi}}, \boldsymbol{\psi}(\boldsymbol{y})\rangle\big)$ and $\tilde{\boldsymbol{\psi}} := \frac{1}{n}\sum_i \boldsymbol{\psi}_i(\tilde{\boldsymbol{y}}_i)$ are easily computable. To ensure that $g(\boldsymbol{\alpha}^\infty, \lambda, \frac{\lambda^\infty}{\lambda}\boldsymbol{w}^\infty) \leq \varepsilon$ for $\lambda \geq \lambda^\infty$, we can now set

$$\lambda^\infty := \frac{1}{\varepsilon}\Big(\|\tilde{\boldsymbol{\psi}}\|^2 + \frac{1}{n}\sum_{i=1}^n \theta_i\Big).$$

Induction step. We utilize the intuition that the expression (7) provides control on the Frank-Wolfe gap for different values of λ if the primal variables $\boldsymbol{w}$ and, consequently, the results of the max oracles stay unchanged. Proposition 1 formalizes this intuition.

Proposition 1. *Assume that $L_i(\boldsymbol{y}_i) = 0$, $i = 1,\dots,n$, i.e., the loss on the ground truth equals zero. Let $\rho := \frac{\lambda^{new}}{\lambda^{old}} < 1$. Then, setting $\alpha_i(\boldsymbol{y}) := \rho\alpha_i^{old}(\boldsymbol{y})$, $\boldsymbol{y} \neq \boldsymbol{y}_i$, and $\alpha_i(\boldsymbol{y}_i) := 1 - \sum_{\boldsymbol{y}\neq\boldsymbol{y}_i}\alpha_i(\boldsymbol{y})$, we then have $\boldsymbol{w}^{new} = \boldsymbol{w}^{old}$ and*

$$g(\boldsymbol{\alpha},\ \lambda^{new}) = g(\boldsymbol{\alpha}^{old}, \lambda^{old}) + (1-\rho)\Delta(\boldsymbol{\alpha}^{old}, \lambda^{old}) \qquad (11)$$

where

$$\Delta(\boldsymbol{\alpha}^{old}, \lambda^{old}) := \frac{1}{n}\sum_{i=1}^n \sum_{\boldsymbol{y}\in\mathcal{Y}_i} \alpha_i^{old}(\boldsymbol{y}) H_i(\boldsymbol{y}; \boldsymbol{w}^{old}).$$

Proof. Consider the problem (3) for both λ^{new} and λ^{old}. Since $\boldsymbol{\psi}_i(\boldsymbol{y}_i) = \mathbf{0}$ and $A^{\text{new}} = \frac{1}{\rho}A^{\text{old}}$, we have that $\boldsymbol{w}^{\text{old}} = A^{\text{old}}\boldsymbol{\alpha}^{\text{old}} = A^{\text{new}}\boldsymbol{\alpha} = \boldsymbol{w}^{\text{new}}$. The assumption $L_i(\boldsymbol{y}_i) = 0$ implies equalities $H_i(\boldsymbol{y}_i; \boldsymbol{w}^{\text{old}}) = 0$. Under these conditions, the equation (11) directly follows from the computation of $g(\boldsymbol{\alpha},\ \lambda^{\text{new}}) - g(\boldsymbol{\alpha}^{\text{old}}, \lambda^{\text{old}})$ and the equality (7). □

Assume that for the regularization parameter λ^{old} the primal-dual pair $\boldsymbol{\alpha}^{\text{old}}$, $\boldsymbol{w}^{\text{old}}$ is $\kappa\varepsilon$-approximate, $0 < \kappa < 1$, i.e., $g(\boldsymbol{\alpha}^{\text{old}}, \lambda^{\text{old}}) \leq \kappa\varepsilon$. Proposition 1 ensures that $g(\boldsymbol{\alpha},\ \lambda^{\text{new}}) \leq \varepsilon$ whenever

$$\rho = 1 - \frac{\varepsilon - g(\boldsymbol{\alpha}^{\text{old}}, \lambda^{\text{old}})}{\Delta(\boldsymbol{\alpha}^{\text{old}}, \lambda^{\text{old}})} \leq 1 - \frac{\varepsilon(1-\kappa)}{\Delta(\boldsymbol{\alpha}^{\text{old}}, \lambda^{\text{old}})}. \qquad (12)$$

Having $\kappa < 1$ ensures that $\rho < 1$, i.e., we get a new break point $\lambda^{\text{new}} < \lambda^{\text{old}}$. If the equation (12) results in $\rho \leq 0$ then we reach the end of the regularization path, i.e., $\boldsymbol{w}^{\text{old}}$ is ε-approximate for all $0 \leq \lambda < \lambda^{\text{old}}$.

To be able to iterate the induction step, we apply one of the algorithms for the minimization of the SSVM objective for λ^{new} to obtain $\kappa\varepsilon$-approximate pair $\boldsymbol{\alpha}^{\text{new}}$, $\boldsymbol{w}^{\text{new}}$. Initializing from $\boldsymbol{\alpha}$, $\boldsymbol{w}^{\text{old}}$ provides fast convergence in practice.

Related work. Due to space constraints, see App. A.

5. Experiments

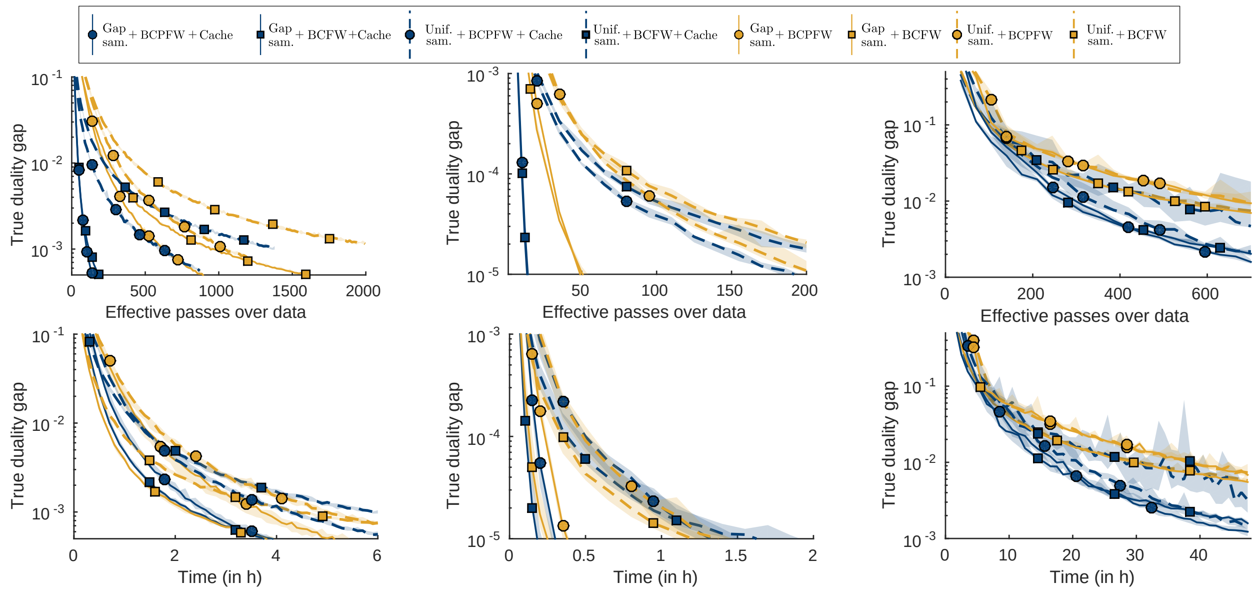
The experimental evaluation consists of two parts: Section 5.1 compares the different algorithms presented in Section 3; Section 5.2 evaluates our approach on the regularization path estimation.

Datasets. We evaluate our methods on four datasets for different structured prediction tasks: OCR (Taskar et al., 2003) for handwritten character recognition, CoNLL (Tjong Kim Sang & Buchholz, 2000) for text chunking, HorseSeg (Kolesnikov et al., 2014) for binary image segmentation and LSP (Johnson & Everingham, 2010) for pose estimation. The models for OCR and CoNLL were provided by Lacoste-Julien et al. (2013). We build our model based on the one by Kolesnikov et al. (2014) for HorseSeg, and the one by Chen & Yuille (2014) for LSP. For OCR and CoNLL, the max oracle consists of the Viterbi algorithm (Viterbi, 1967); for HorseSeg – in graph cut (Boykov & Kolmogorov, 2004), for LSP – in belief propagation on a tree with messages passed by a generalized distance transform (Felzenszwalb & Huttenlocher, 2005). Note that the oracles of HorseSeg and LSP require positivity constraints on a subset of the weights in order to be tractable. The BCFW algorithm with positivity constraints is derived in App. H. We provide a detailed description of the datasets in App. I with a summary in Table 1.

The problems included in our experimental study vary in the number of objects n (from 100 to 25,000), in the number of features d (from 10^2 to 10^6), and in the computational cost of the max oracle (from 10^{-4} to 2 seconds).

5.1. Comparing the variants of BCFW

In this section, we evaluate the three modifications of BCFW presented in Section 3. We compare 8 methods obtained by all the combinations of three binary dimensions: gap-based vs. uniform sampling of objects, BCFW vs. BCPFW, caching oracle calls vs. no caching.



(a) OCR-large, $\lambda = 0.001$ (b) HorseSeg-medium, $\lambda = 10$ (c) LSP-small, $\lambda = 100$

Figure 3. Summary of the results of Section 5.1: the duality gap against the number of effective passes over data (top) and time (bottom).

We report the results of each method on 6 datasets (including 3 sizes of HorseSeg) for three values of the regularization parameter λ: the value leading to the best test performance, a smaller and a larger value. For each setup, we report the duality gap against both number of oracle calls and elapsed time. We run each method 5 times with different random seeds influencing the order of sampled objects and report the median (bold line), minimum and maximum values (shaded region). We summarize the results in Figure 3 and report the rest in App. J.

First, we observe that, aligned with our theoretical results, gap sampling always leads to faster convergence (both in terms of time and the number of effective passes). The effect is stronger when n is large (Figure 3b). Second, caching always helps in terms of number of effective passes, but an overhead caused by maintaining the cache is significant when the max oracle is fast (Figure 3a). In the case of expensive oracle (Figure 3c), the cache overhead is negligible. Third, the pairwise steps (BCPFW) lead to an improvement to get smaller values of duality gaps. The effect is stronger when the problem is more strongly convex, i.e., λ is bigger. However, maintaining the active sets results in computational overheads, which sometimes are significant. Note that the overhead of cache and active sets are shared, because they are maintained in the same data structure. Using a cache also greatly limits the memory requirements of BCPFW, because, when the cache is hit, the active set is guaranteed not to grow.

Recommendation. For off-the-shelf usage, we recommend to use the BCPFW + gap sampling + cache method when oracle calls are expensive, and the BCFW + gap sampling method when oracle calls are cheap.

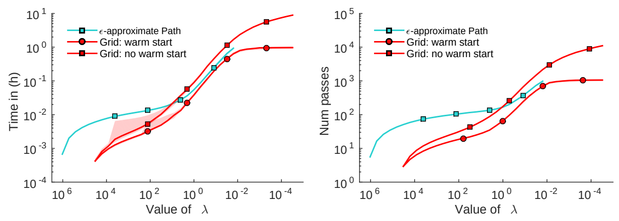


Figure 4. On HorseSeg-small, we compare ε-approximate regularization path against grid search with/without warm start. We report the cumulative running time (left) and the cumulative effective number of passes (right) required to get to each value of the regularization parameter λ.

5.2. Regularization path

In this section, we evaluate our regularization path algorithm presented in Section 4. We compare an ε-approximate regularization path with $\varepsilon = 0.1$ against the standard grid search approach with/without warm start (we use a grid of 31 values of λ: $2^{15}, 2^{14}, \ldots, 2^{-15}$). In Figure 4, we report the cumulative elapsed time and cumulative number of effective passes over the data required by the three methods to reach a certain value of λ on the HorseSeg-small dataset (starting from the initialization value for the path method and the maximum values of the grid for the grid search methods). The methods and additional experiments are detailed in App. K.

Interpretation. First, we observe that warm start speeds up the grid search. Second, the cost of computing the full regularization path is comparable with the cost of grid search. However, the regularization path algorithm finds solutions for all values of λ without the need to predefine the grid.

Acknowledgments

This work was partially supported by the MSR-Inria Joint Center and a Google Research Award.

References

Alayrac, J.-B., Bojanowski, P., Agrawal, N., Sivic, J., Laptev, I., and Lacoste-Julien, S. Learning from narrated instruction videos. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Beck, A., Pauwels, E., and Sabach, S. The cyclic block conditional gradient method for convex optimization problems. *SIAM Journal on Optimization*, 25(4):2024–2049, 2015.

Boykov, Y. and Kolmogorov, V. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 26(9):1124–1137, 2004.

Branson, S., Beijbom, O., and Belongie, S. Efficient large-scale structured learning. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

Chen, X. and Yuille, A. Articulated pose estimation by a graphical model with image dependent pairwise relations. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

Csiba, D., Qu, Z., and Richtárik, P. Stochastic dual coordinate ascent with adaptive probabilities. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2015.

Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. Least angle regression. *Annals of Statistics*, 2004.

Felzenszwalb, P. F. and Huttenlocher, D. P. Pictorial structures for object recognition. *International Journal of Computer Vision (IJCV)*, 61(1):55–79, 2005.

Franc, V. FASOLE: Fast Algorithm for Structured Output LEarning. In *Proceedings of European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, 2014.

Frank, M. and Wolfe, P. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:95–110, 1956.

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

Jegelka, S., Bach, F., and Sra, S. Reflection methods for user-friendly submodular optimization. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

Joachims, T., Finley, T., and Yu, C. Cutting-plane training of structural SVMs. *Machine Learning*, 77(1):27–59, 2009.

Johnson, S. and Everingham, M. Clustered pose and nonlinear appearance models for human pose estimation. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2010.

Joulin, A., Tang, K., and Fei-Fei, L. Efficient image and video co-localization with Frank-Wolfe algorithm. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014.

Kolesnikov, A., Guillaumin, M., Ferrari, V., and Lampert, C. H. Closed-form approximate CRF training for scalable image segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 550–565, 2014.

Lacoste-Julien, S. and Jaggi, M. On the global linear convergence of Frank-Wolfe optimization variants. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

Lacoste-Julien, S., Jaggi, M., Schmidt, M., and Pletscher, P. Block-coordinate Frank-Wolfe optimization for structural SVMs. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2013.

Mitchell, B., Demyanov, V. F., and Malozemov, V. Finding the point of a polyhedron closest to the origin. *SIAM Journal on Control*, 12(1), 1974.

Ñanculef, R., Frandi, E., Sartori, C., and Allende, H. A novel Frank-Wolfe algorithm. Analysis and applications to large-scale SVM training. *Information Sciences*, 285: 66–99, 2014.

Needell, D., Ward, R., and Srebro, N. Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

Nesterov, Y. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

Platt, J. C. Fast training of support vector machines using sequential minimal optimization. In Schölkopf, B., Burges, C., and Smola, A. (eds.), *Advances in Kernel Methods - Support Vector Learning*, pp. 185–208. MIT Press, 1999.

Ratliff, N., Bagnell, J. A., and Zinkevich, M. (Online) subgradient methods for structured prediction. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2007.

Rosset, S. and Zhu, J. Piecewise linear regularized solution paths. *The Annals of Statistics*, 35(3):1012–1030, 2007.

Shah, N., Kolmogorov, V., and Lampert, C. H. A multi-plane block-coordinate Frank-Wolfe algorithm for training structural SVMs with a costly max-oracle. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

Taskar, B. *Learning structured prediction models: A large margin approach*. PhD thesis, Stanford University, 2004.

Taskar, B., Guestrin, C., and Koller, D. Max-margin Markov networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

Tjong Kim Sang, E. F. and Buchholz, S. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of the 2nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language Learning (CoNLL)*, 2000.

Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research (JMLR)*, 6:1453–1484, 2005.

Viterbi, A. J. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967.

Wang, Y.-X., Sadhanala, V., Dai, W., Neiswanger, W., Sra, S., and Xing, E. P. Parallel and distributed block-coordinate Frank-Wolfe algorithms. *arXiv:1409.6086v2*, 2014.

Wolfe, P. Convergence Theory in Nonlinear Programming. In Abadie, J (ed.), *Integer and Nonlinear Programming*, pp. 1–23. North-Holland, 1970.

Zhao, P. and Zhang, T. Stochastic optimization with importance sampling for regularized loss minimization. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2015.