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.

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

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

[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

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

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).

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

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

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 $A := \left\{\frac{1}{\lambda n}\boldsymbol{\psi}_i(\boldsymbol{y}) \in \mathbb{R}^d \;\middle|\; i\in[n], \boldsymbol{y}\in\mathcal{Y}_i\right\}$, and the vector $\boldsymbol{b} \in \mathbb{R}^m$ is given by $\boldsymbol{b} := \left(\frac{1}{n}L_i(\boldsymbol{y})\right)_{i\in[n],\boldsymbol{y}\in\mathcal{Y}_i}$.

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):

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

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

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).

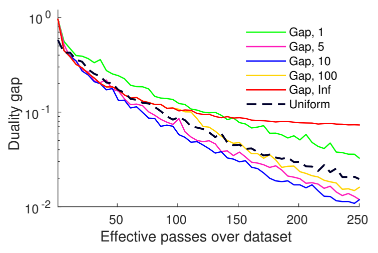
3.1. Adaptive non-uniform sampling

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).

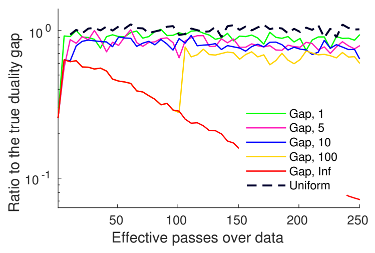
Convergence. Assume that at iteration k of Alg. 1, we have the probability $p_i^{(k)}$ of sampling block i. By minimizing the descent lemma bound (8) w.r.t. γ for each i independently under the assumption that $g_i(\boldsymbol{\alpha}^{(k)}) \leq C_f^{(i)}$, and then taking the conditional expectation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

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

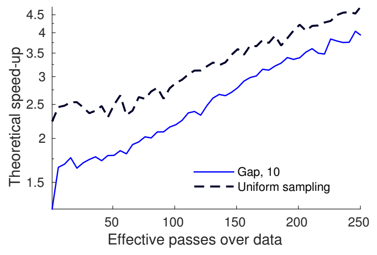
Intuitively, by adapting the probabilities $p_i^{(k)}$, we can obtain a better bound on the expected improvement of f. In the ideal scenario, one would choose deterministically the block i with the maximal value of $g_i^2(\boldsymbol{\alpha}^{(k)})/C_f^{(i)}$.



(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 $\mathcal{S}_i \subseteq \mathcal{C}_i$.

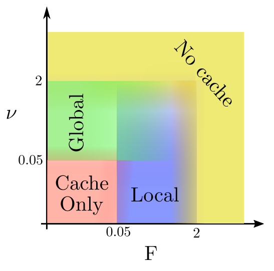
Cache hit criterion. An important part of a caching scheme is the criterion deciding whether the cache look up is sufficient or the max oracle needs to be called. Intuitively, we want to use the cache whenever it allows optimization to make large enough progress. We use as measure of potential progress 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 $\hat{g}_i^{(k)} := \lambda(\boldsymbol{w}_i^{(k)} - \boldsymbol{w}_{\boldsymbol{c}})^\top \boldsymbol{w}^{(k)} - \ell_i{}^{(k)} + \ell_{\boldsymbol{c}}$, 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 $\max(F g_i^{(k_i)}, \frac{\nu}{n} g^{(k_0)})$ 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).

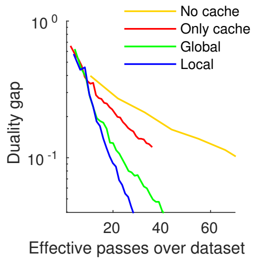
Theorem 3. *Consider the same notation as in Theorem 1. Let* $\tilde{\nu} := \frac{1}{n}\nu \leq 1$. *The iterate* $\boldsymbol{\alpha}^{(k)}$ *of Alg. 5 satisfies* $\mathbf{E}\big[f(\boldsymbol{\alpha}^{(k)})\big] - f(\boldsymbol{\alpha}^*) \leq \frac{2n}{\tilde{\nu}k+2n}\big(\frac{1}{\tilde{\nu}} C_f^{\otimes}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



(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.

choice of the parameters, so, in what follows, we use values $F = 0.25$ and $\nu = 0.01$.

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 $\boldsymbol{w} = \boldsymbol{0}$ 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 $\tilde{\boldsymbol{y}}_i = \operatorname{argmax}_{\boldsymbol{y}\in\mathcal{Y}_i} H_i(\boldsymbol{y};\boldsymbol{0}) = \operatorname{argmax}_{\boldsymbol{y}\in\mathcal{Y}_i} L_i(\boldsymbol{y})$ be the output of the max oracle for $\boldsymbol{w} = \boldsymbol{0}$. 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:

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

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

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

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*

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

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 $\boldsymbol{\psi}_i(\boldsymbol{y}_i) = \boldsymbol{0}$ 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

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

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

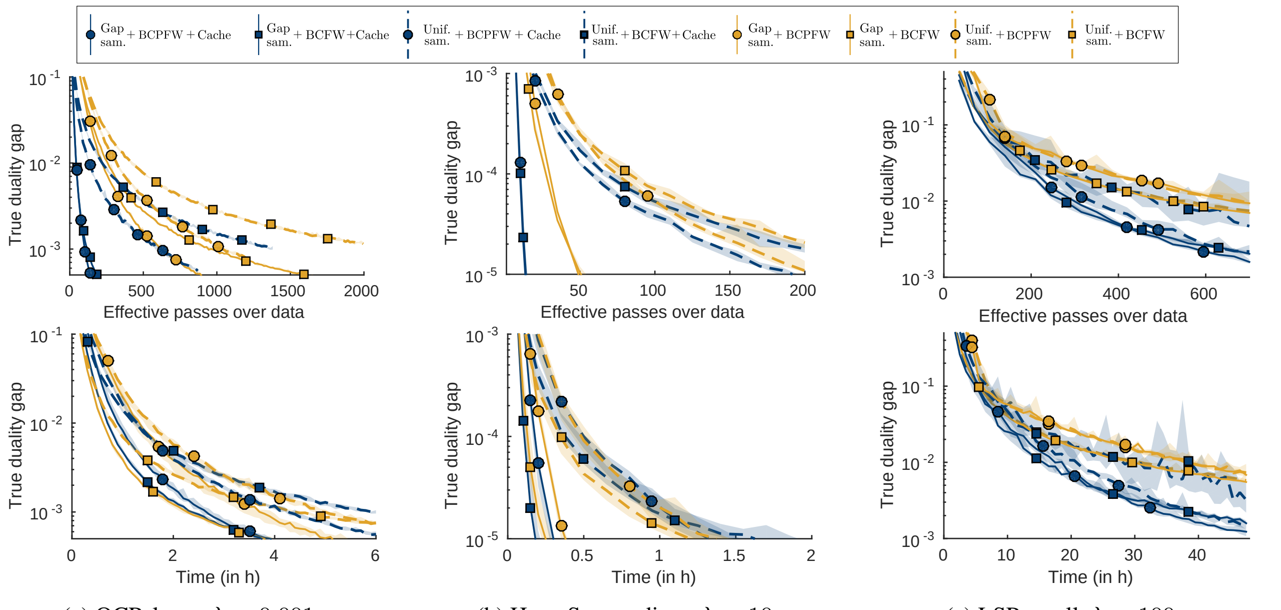
The experimental evaluation consists of two parts: Section 5.1 compares the different algorithms presented in Section 3; Section 5.2 evaluates our approach on the regularization path estimation.

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.

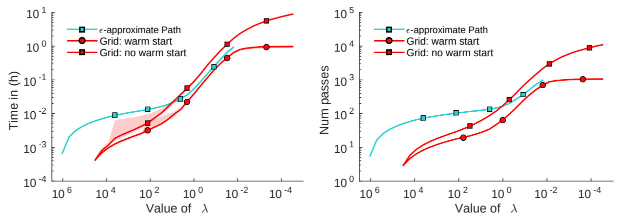


Figure 4. On HorseSeg-small, we compare ε-approximate regularization path against grid search with/without warm start. We report the cumulative running time (left) and the cumulative effective number of passes (right) required to get to each value of the regularization parameter λ.

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.

Supplementary Material
Minding the Gaps for Block Frank-Wolfe optimization of structured SVMs

Outline

A. Related work for regularization path

Efron et al. (2004), in their seminal paper, introduced the notion of regularization path and showed that the regularization path of LASSO (Tibshirani, 1996) is piecewise linear. Hastie et al. (2004) proposed the path following method to compute the exact regularization path for the binary SVM with L2-regularization. Exact path following algorithms suffer from numerical instabilities as they repeatedly invert a potentially badly-conditioned matrix (Allgower & Georg, 1993). In addition, Gärtner et al. (2012) show that, in the worst case, the regularization path for binary SVM contains an exponential number of break points. Although the exact path following methods for SVM are still developed (Sentelle et al., 2015), approximate methods might be more suited for practical use cases. Karasuyama & Takeuchi (2011) proposed a method providing a trade-off between accuracy of the path and its computational cost. Recently, Giesen et al. (2012) have developed a general framework to construct a piecewise-constant ε-approximate path with at most $\mathcal{O}(1/\sqrt{\varepsilon})$ break points and applied it, e.g., to binary SVM.

In contrast to binary SVM, regularization paths for multi-class SVMs are more complex and less studied. Lee & Cui (2006) proposed a path following method for multi-class SVM in the MSVM formulation of Lee et al. (2004). Wang & Shen (2006) analyzed the regularization path for the L1 version of MSVM. Finally, Jun-Tao & Ying-Min (2010) constructed the regularization path for the multi-class SVM with huberized loss. We are not aware of any work computing the regularization path for SSVM or, for its predecessor multi-class SVM in the formulation of Crammer & Singer (2001).

The induction step of our method is similar to Alg. 1 of (Giesen et al., 2012) applied to the case of binary SVM. They also construct a piecewise linear ε-approximate path by alternating the SVM solver and a procedure to identify the region where the output of the solver is accurate enough.

In contrast to our method, Giesen et al. (2012) construct the path only for the predefined segment of the values of λ. We do not require such a segment as input and are able to find the largest and smallest value automatically. Another difference to (Giesen et al., 2012) consists in using the λ-formulation of SVM instead of the C-formulation. In the two formulations, the accuracy parameter ε is scaled differently for the different values of the regularization parameters. The λ-formulation requests higher accuracy for the small values of λ and, thus, creates more break points in that region.

B. Block descent lemma for BCFW

Definition 2 (Block curvature constant)**.** *Consider a convex function f defined on a separable domain $\mathcal{M} = \mathcal{M}^{(1)} \times \cdots \times \mathcal{M}^{(n)}$. The curvature constant $C_f^{(i)}$ of the function f w.r.t. the individual block of coordinates $\mathcal{M}^{(i)}$ is defined by*

$$C_f^{(i)} := \sup \tfrac{2}{\gamma^2} \left(f(\boldsymbol{\beta}) - f(\boldsymbol{\alpha}) - \langle \boldsymbol{\beta}_{(i)} - \boldsymbol{\alpha}_{(i)}, \nabla_{(i)} f(\boldsymbol{\alpha}) \rangle \right)$$
$$\text{s.t. } \boldsymbol{\alpha} \in \mathcal{M},\ \boldsymbol{s}_{(i)} \in \mathcal{M}^{(i)}, \gamma \in [0,1],$$
$$\boldsymbol{\beta} = \boldsymbol{\alpha} + \gamma(\boldsymbol{s}_{[i]} - \boldsymbol{\alpha}_{[i]}). \tag{13}$$

Here $\boldsymbol{s}_{[i]} \in \mathbb{R}^m$ and $\boldsymbol{\alpha}_{[i]} \in \mathbb{R}^m$ are the zero-padded versions of $\boldsymbol{s}_{(i)} \in \mathcal{M}^{(i)}$ and $\boldsymbol{\alpha}_{(i)} \in \mathcal{M}^{(i)}$, respectively. Note that, although $\boldsymbol{s}_{[i]} \notin \mathcal{M}$ and $\boldsymbol{\alpha}_{[i]} \notin \mathcal{M}$, we have that $\boldsymbol{\beta} := \boldsymbol{\alpha} + \gamma(\boldsymbol{s}_{[i]} - \boldsymbol{\alpha}_{[i]}) \in \mathcal{M}$.

In the case of SSVM, the curvature constant $C_f^{(i)}$ can be upper bounded (tightly in the worst case) with $\frac{4R_i^2}{\lambda n^2}$ where $R_i := \max_{\boldsymbol{y}\in\mathcal{Y}_i} \|\boldsymbol{\psi}_i(\boldsymbol{y})\|_2$ (Lacoste-Julien et al., 2013, Appendix A).[6]

For reference, we restate below the key descent lemma used for the proof of convergence of BCFW and its variants. We note in passing that this is an affine invariant analog of the standard descent lemmas that use the Lipschitz continuity of the gradient function to show progress during first order optimization algorithms.

Lemma 3 (Block descent lemma). *For any $\boldsymbol{\alpha} \in \mathcal{M}$ and block i, let $\boldsymbol{s}_{(i)} \in \mathcal{M}^{(i)}$ be the Frank-Wolfe corner selected by the max oracle of block i at $\boldsymbol{\alpha}$. Let $\boldsymbol{\alpha}_{LS}$ be obtained by the line search between $\boldsymbol{\alpha}_{(i)} \in \mathcal{M}^{(i)}$ and $\boldsymbol{s}_{(i)}$, i.e., $f(\boldsymbol{\alpha}_{LS}) = \min_{\gamma\in[0,1]} f(\boldsymbol{\alpha}_\gamma)$ where $\boldsymbol{\alpha}_\gamma := \boldsymbol{\alpha} + \gamma(\boldsymbol{s}_{[i]} - \boldsymbol{\alpha}_{[i]})$. Then, it holds that for each $\gamma \in [0,1]$:*

$$f(\boldsymbol{\alpha}_{LS}) \leq f(\boldsymbol{\alpha}) - \gamma g_i(\boldsymbol{\alpha}) + \tfrac{\gamma^2}{2} C_f^{(i)} \tag{14}$$

where $C_f^{(i)}$ is the curvature constant of the function f over the factor $\mathcal{M}^{(i)}$ and $g_i(\boldsymbol{\alpha})$ is the block gap at the point $\boldsymbol{\alpha}$ w.r.t. the block i.

Proof. From Definition 2 of the curvature constant and the expression (6) for the block gap, we have

$$\begin{aligned} f(\boldsymbol{\alpha}_\gamma) =& f(\boldsymbol{\alpha} + \gamma(\boldsymbol{s}_{[i]} - \boldsymbol{\alpha}_{[i]})) \\ \leq& f(\boldsymbol{\alpha}) + \gamma\langle \boldsymbol{s}_{(i)} - \boldsymbol{\alpha}_{(i)}, \nabla_{(i)} f(\boldsymbol{\alpha})\rangle + \tfrac{\gamma^2}{2} C_f^{(i)} \\ =& f(\boldsymbol{\alpha}) - \gamma g_i(\boldsymbol{\alpha}) + \tfrac{\gamma^2}{2} C_f^{(i)}\,. \end{aligned}$$

The inequality $f(\boldsymbol{\alpha}_{LS}) \leq f(\boldsymbol{\alpha}_\gamma)$ completes the proof. □

C. Toy example for gap sampling

In this section, we construct a toy example of the structured SVM problem where the adaptive gap-based sampling is n times faster than non-adaptive sampling schemes such as uniform sampling or curvature-based sampling (the latter being the affine invariant analog of Lipschitz-based sampling).

General idea. The main idea is to consider a training set where there are $n-1$ "easy" objects that need to be visited only once to learn to classify them, and one "hard" object that requires at least $K \gg 1$ visits in order to get the optimal parameter. We can design the example in such a way that the curvature or Lipschitz constants are non-informative about which example is hard, and which is easy. The non-adaptive sampling schemes will thus have to visit the easy objects as often as the hard object, whereas the gap sampling technique can adapt to focus only on the single hard object after having visited the easy objects once, thus yielding an overall $\min\{n, K\}$-times speedup.

Note that large-scale learning datasets could have analogous features as this toy example: a subgroup of objects might be easier to learn than another, and moreover, they might share similar information, so that after visiting a subset, we do not need to linger on the other ones from the same subset as all the information has already been extracted. We cannot know in advance which subgroups are these subsets, and thus an adaptive scheme is needed.

C.1. Explicit construction

For simplicity, we set the weight of the regularizer λ to $1/n$ so that the scaling factor defining A in Problem (3) is $1/\lambda n = 1$. The matrix A thus consists of the difference feature maps, i.e., $A := \left\{\boldsymbol{\psi}_i(\boldsymbol{y}) := \boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}) \in \mathbb{R}^d \;\middle|\; i \in [n], \boldsymbol{y} \in \mathcal{Y}_i\right\}$. In our example, we use feature maps of dimensionality $d := K+1 := |\mathcal{Y}_i|$. Let $\mathcal{Y}_i := \{0, 1, \ldots, K\}$ be the set of labels for the object i and the label 0 be the correct label. We consider the zero-one loss, i.e., $L_i(0) = 0$ and $L_i(k) = 1$ for $k \geq 1$. In the following, let $\{\boldsymbol{e}_j\}_{j=1}^d$ be the standard basis vectors for $\mathbb{R}^d$.

Hard and easy objects. We construct the feature map $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$ so that only the last coordinate of the parameter vector is needed to classify correctly the easy object, whereas all the other coordinates are needed to classify correctly the hard object. By using a different set of coordinates between the easy and the hard objects, we simplify the analysis as the optimization for both block types decouples (become independent). Specifically, we set the feature map for the correct label to be $\boldsymbol{\phi}(\boldsymbol{x}_i, 0) := \mathbf{0}$ for all objects i. We let $i = 1$ be the hard object and we set $\boldsymbol{\phi}(\boldsymbol{x}_i, k) := -\frac{1}{\sqrt{2}}\boldsymbol{e}_k$ for $k = 1, \ldots, K$. For the easy object, $i \in \{2, \ldots, n\}$, we use the constant $\boldsymbol{\phi}(\boldsymbol{x}_i, k) := -\boldsymbol{e}_{K+1}$ for all $k \geq 1$. The normalization of the feature maps is made so that the curvature constants for all the objects are equal (see below). Note also here that $\boldsymbol{\psi}_1(k) \perp \boldsymbol{\psi}_i(l)$ for any labels k, l, and thus the optimization over block 1 decouples with the one for the other blocks $i = 2, \ldots, n$. The SSVM dual (3) takes here the following simple form:

$$\min_{\substack{\boldsymbol{\alpha}\in\mathbb{R}^m \\ \boldsymbol{\alpha}\succcurlyeq 0}} \quad \frac{1}{n}\sum_{k=1}^{K}\left(\tfrac{1}{4}\alpha_1(k)^2 - \alpha_1(k)\right) + \tfrac{1}{n}\left(\tfrac{1}{2}u^2 - u\right) \tag{15}$$

$$\text{s.t.} \quad \sum_{k=0}^{K}\alpha_i(k) = 1 \quad \forall i \in [n]\,, \quad u = \sum_{i=2}^{n}\sum_{k=1}^{K}\alpha_i(k)\,,$$

where we have introduced the auxiliary variable u to highlight the simple structure for the optimization over the easy

[6]More generally, let $\|\cdot\|_i$ be some norm defined on $\mathcal{M}^{(i)}$. Then suppose that L_i is the Lipschitz-continuity constant with respect to this norm for $\nabla_{(i)} f(\boldsymbol{\alpha})$ when only $\boldsymbol{\alpha}_{(i)}$ varies, i.e., $\|\nabla_{(i)} f(\boldsymbol{\alpha}) - \nabla_{(i)} f(\boldsymbol{\alpha} + \boldsymbol{s}_{[i]} - \boldsymbol{\alpha}_{[i]})\|_i^* \leq L_i \|\boldsymbol{s}_{(i)} - \boldsymbol{\alpha}_{(i)}\|_i$ for all $\boldsymbol{\alpha} \in \mathcal{M}$, $\boldsymbol{s}_{(i)} \in \mathcal{M}^{(i)}$, where $\|\cdot\|_i^*$ is the dual norm of $\|\cdot\|_i$. Then similarly to Lemma 7 in Jaggi (2013), we have $C_f^{(i)} \leq L_i\big(\operatorname{diam}_{\|\cdot\|_i} \mathcal{M}^{(i)}\big)^2$.

blocks. The unique[7] solution for the first (hard) block is easily seen to be $\alpha_1^*(k) = \frac{1}{K}$ for $k \geq 1$ and $\alpha_1^*(0) = 0$. For the easy blocks, any feasible combination of dual variables that gives $u^* = 1$ is a solution. This gives the optimal parameter $\boldsymbol{w}^* = A\boldsymbol{\alpha}^* = \boldsymbol{e}_{K+1} + \frac{1}{K}\sum_{k=1}^{K} \boldsymbol{e}_k$.

Optimization on the hard object. The objective for the hard object (block 1) in (15) is similar to the one used to show a lower bound of $\Omega(1/t)$ suboptimality error after t iterations for the Frank-Wolfe algorithm for t smaller than the dimensionality (e.g., see Lemma 3 and 4 in Jaggi (2013)), hence showing that the optimization is difficult on this block. The BCFW algorithm is initialized with $\boldsymbol{w} = \boldsymbol{0}$, which corresponds to putting all the mass on the correct label, i.e., $\alpha_i(0) = 1$ and $\alpha_i(k) = 0,\ k \geq 1$. At each iteration of BCFW, the mass can be moved only towards one corner, and all the corners (of the simplex) have exactly one non-zero coordinate. This means that after t iterations of BCFW on the first block, at most t non-ground truth dual variables can be non-zero. Minimizing the objective (15) over the first block with the constraint that at most t of these variables are non-zero give the similar solution $\alpha_1(k) = 1/t$ for $k = 1, \dots, t$, which gives a suboptimality of $\frac{1}{4n}(\frac{1}{t} - \frac{1}{K})$ for $t \leq K$. Similarly, this also yields the smallest FW gap[8] possible for this block after t iterations, which is $\frac{1}{n}\frac{1}{2t}$. This means that in order to get a suboptimality error smaller than ε, one needs at least

$$t \geq \Omega(\min\{K, \tfrac{1}{n\varepsilon}\}) \tag{16}$$

BCFW iterations on the first block.[9]

Optimization on easy objects. Finally, we now show that after one iteration on *any* easy object, the gaps g_i on all easy objects become zero (i.e., they are all optimal and then stay optimal as the optimization is decoupled with the first block). After this iteration, BCFW with gap sampling visits all the easy objects exactly once and sets their gap estimates to zero, thus never revisiting them again.

Note that before visiting any easy object i, we have $\langle \boldsymbol{w}, \boldsymbol{\psi}_i(k)\rangle = 0$ for all k as the features for the hard object are orthogonal and $\boldsymbol{w}$ is initialized to zero. Thus, at the first visit of an easy object $i \in \{2, \dots, n\}$, we have $H_i(0; \boldsymbol{w}) = 0$ and $H_i(k; \boldsymbol{w}) = 1$, $k \geq 1$, and the max oracle returns some (any) label $k \in \{1, \dots, K\}$. Following the steps of Algorithm 1, we have $\boldsymbol{w}_{\boldsymbol{s}} := \frac{1}{\lambda n}\boldsymbol{\psi}(k) = \boldsymbol{e}_{K+1}$ and $\ell_s = \frac{1}{n}L_i(k) = \frac{1}{n}$. Then $g_i = \frac{1}{n}$ and $\gamma = 1$ as $\boldsymbol{w}_i = \boldsymbol{0}$. The assignment $\boldsymbol{w}_i = \boldsymbol{e}_{K+1}$ implies the update $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{e}_{K+1}$ of the parameter vector. After such an update, at all iterations, for all easy objects $i \in \{2, \dots, n\}$ and for all labels $k \in \{1, \dots, K\}$, we have

$$H_i(k; \boldsymbol{w}) = L_i(k) - \langle \boldsymbol{w}, \boldsymbol{e}_{K+1}\rangle = 0 \tag{17}$$

because the coordinate w_{K+1} is never updated again. According to (7), the equalities (17) imply that the block gaps g_i equal zero for all the easy objects.

Curvature constants. The simple structure of the matrix A allows us to explicitly compute the curvature constants $C_f^{(i)}$ corresponding to both easy and hard objects.

The SSVM dual (3) is a quadratic function with a constant Hessian $H := \lambda A^\mathsf{T} A$, so the second-order Taylor expansion of f at a point $\boldsymbol{\alpha}$ allows us to rewrite the definition 13 as

$$\begin{aligned} C_f^{(i)} &= \sup_{\substack{\boldsymbol{\alpha}\in\mathcal{M} \\ \boldsymbol{s}_{(i)}\in\mathcal{M}^{(i)}}} (\boldsymbol{s}_{[i]} - \boldsymbol{\alpha}_{[i]})^\mathsf{T} H (\boldsymbol{s}_{[i]} - \boldsymbol{\alpha}_{[i]}) \\ &= \lambda \sup_{\substack{\boldsymbol{\alpha}\in\mathcal{M} \\ \boldsymbol{s}_{(i)}\in\mathcal{M}^{(i)}}} \|A(\boldsymbol{s}_{[i]} - \boldsymbol{\alpha}_{[i]})\|_2^2 \\ &= \lambda \max_{k,\,l} \|\boldsymbol{\phi}(\boldsymbol{x}_i, k) - \boldsymbol{\phi}(\boldsymbol{x}_i, l)\|_2^2. \end{aligned}$$

The last line uses the property that the maximum of a convex function over a convex set is obtained at a vertex.

In the case of the hard object, we can get

$$C_f^{(1)} = \lambda \|\boldsymbol{\phi}(\boldsymbol{x}_1, 1) - \boldsymbol{\phi}(\boldsymbol{x}_1, 2)\|_2^2 = \tfrac{1}{n}. \tag{18}$$

In the case of an easy object, we can get

$$C_f^{(i)} = \lambda \|\boldsymbol{\phi}(\boldsymbol{x}_i, 1) - \boldsymbol{\phi}(\boldsymbol{x}_i, 0)\|_2^2 = \tfrac{1}{n}. \tag{19}$$

Adaptive and non-adaptive sampling. Let t be the number of steps needed on the hard block. By (16), we need $t \geq \Omega(\min\{K, \frac{1}{n\varepsilon}\})$ to get a suboptimality smaller than ε. The uniform sampling scheme visits all the objects with the same probability. In the setting constructed above, it makes, on average, t visits to each easy object prior to visiting the hard object t times. Thus, the overall scheme will call the max-oracle $O(nt)$ times. All the curvature constants $C_f^{(i)}$ are equal, so the sampling proportional to the curvature constants is equivalent to uniform sampling.

The adaptive sampling scheme visits each easy object only once after the first visit to any of them. After such a visit to any easy object, its local gap estimate equals zero and this object is never visited again. The gap sampling scheme thus makes an overall $O(n + t)$ oracle calls. The adaptive scheme is thus approximately $\min\{n, t\} = \min\{n, K, \frac{1}{n\varepsilon}\}$ times faster than the non-adaptive ones. The speed-up can be made arbitrary large by setting both n and K large enough, and ε small enough.

[7]Uniqueness can be proved by noticing that the objective is strongly convex in $\boldsymbol{\alpha}_1$ after removing $\alpha_1(0)$ and replacing the equality constraint with an inequality.

[8]Recall that the FW gap here is the same as the Lagrangian duality gap (see Section 2.3), and so if one cares about the SSVM primal suboptimality, one needs a small FW gap.

[9]In fact, BCFW also has a $\mathcal{O}(\frac{1}{nt})$ gap after t iterations on the first block by the standard FW convergence theorem, as $C_f^{(1)} = \frac{1}{n}$ as we show in (18).

Lipschitz and curvature constants. The non-uniform sampling techniques used in the work of Nesterov (2012); Needell et al. (2014); Zhao & Zhang (2015) use Lipschitz constants of partial derivatives to obtain the sampling probabilities. In our discussion above, we use the curvature constants. Lacoste-Julien & Jaggi (2015, Appendix C) note that the curvature constants are affine invariant quantities and, thus, are more suited for the analysis of Frank-Wolfe methods compared to Lipschitz constants (which depend on a choice of norm). We illustrate this point on our toy example by explicitly computing the Lipschitz constants over blocks for the ℓ_2 and ℓ_1 norm. For both easy and hard blocks, the Lipschitz constant of the gradient with respect to the ℓ_2 norm equals the largest eigenvalues of the corresponding block Hessians. For an easy object, the block Hessian is a rank one matrix with the only non-zero eigenvalue equal to $\lambda(|\mathcal{Y}_i| - 1) = \frac{K}{n}$. For the hard object, the block Hessian is a diagonal matrix with non-zero entries equal to $\frac{\lambda}{2} = \frac{1}{2n}$. Here the Lipschitz constant for the easy block is about K times bigger than the one for the hard block, and thus for a large number of labels K, sampling according to Lipschitz constants can be much slower than sampling according to the curvature constants, which was itself slower than the adaptive sampling scheme.

This poor scaling of the Lipschitz constants is partly due to the bad choice of norm in relationship to the optimization domain. d'Aspremont et al. (2013) suggests to use the atomic norm of the domain $\mathcal{M}$ for the analysis. In the case of the simplex, we get the ℓ_1 norm to measure the diameter of the domain, and its dual norm (ℓ_∞) to measure the Lipschitz constant of the gradient. With this norm, the Lipschitz constant stays as $\frac{1}{2n}$ for the hard block, but decreases to the more reasonable $\frac{1}{n}$ for the easy blocks. As explained in footnote 6, we can use the bound $C_f^{(i)} \leq L_i\big(\operatorname{diam}_{\|\cdot\|_i} \mathcal{M}^{(i)}\big)^2$ for the curvature constant. As the diameter for the simplex measured with the ℓ_1-norm is 2, we get the bound $C_f^{(1)} \leq \frac{2}{n}$ for the hard block, very close to its exact value of $\frac{1}{n}$ as derived in (18). The ℓ_1 norm thus appears as a more appropriate choice for this problem.

D. Detailed algorithms.

In this section, we give the detailed versions of our BCFW variants applied to the SSVM objective presented in the main paper. Algorithm 2 describes BCFW with adaptive gap sampling. We give the block-coordinate version of pairwise FW (BCPFW) in Algorithm 3, and of away-step FW (BCAFW) in Algorithm 4. We note that these two algorithms are simply the *blockwise* application of the PFW and AFW algorithms as described in Lacoste-Julien & Jaggi (2015), but in the context of SSVM which complicates the notation. Algorithm 5 presents the BCFW algorithm with caching. Algorithm 7 presents our method for computing the regularization path and Algorithm 6 presents the initialization of the regularization path.

Algorithm 2 Block-coordinate Frank-Wolfe (BCFW) algorithm with gap sampling for structured SVM

```
1: Let $\boldsymbol{w}^{(0)} := \boldsymbol{w}_i{}^{(0)} := \mathbf{0}$;  $\ell^{(0)} := \ell_i{}^{(0)} := 0$;  $g_i^{(0)} := +\infty$;
2: $k_i := 0$                    // the last time $g_i$ was computed
3: for $k := 0, \dots, \infty$ do
4:     Pick $i$ at random with probability $\propto\ g_i^{(k_i)}$
5:     Solve $\boldsymbol{y}_i^* := \operatorname{argmax}_{\boldsymbol{y} \in \mathcal{Y}_i} H_i(\boldsymbol{y}; \boldsymbol{w}^{(k)})$
6:     Let $k_i := k$
7:     Let $\boldsymbol{w_s} := \frac{1}{\lambda n}\boldsymbol{\psi}_i(\boldsymbol{y}_i^*)$ and $\ell_{\boldsymbol{s}} := \frac{1}{n} L_i(\boldsymbol{y}_i^*)$
8:     Let $g_i^{(k_i)} := \lambda(\boldsymbol{w}_i^{(k)} - \boldsymbol{w_s})^\top \boldsymbol{w}^{(k)} - \ell_i^{(k)} + \ell_{\boldsymbol{s}}$
9:     Let $\gamma := \frac{g_i^{(k_i)}}{\lambda\|\boldsymbol{w}_i^{(k)} - \boldsymbol{w_s}\|^2}$ and clip to $[0,1]$
10:    Update $\boldsymbol{w}_i{}^{(k+1)} := (1-\gamma)\boldsymbol{w}_i{}^{(k)} + \gamma\,\boldsymbol{w_s}$
11:        and $\ell_i{}^{(k+1)} := (1-\gamma)\ell_i{}^{(k)} + \gamma\,\ell_{\boldsymbol{s}}$
12:    Update $\boldsymbol{w}^{(k+1)} := \boldsymbol{w}^{(k)} + \boldsymbol{w}_i{}^{(k+1)} - \boldsymbol{w}_i{}^{(k)}$
13:        and $\ell^{(k+1)} := \ell^{(k)} + \ell_i{}^{(k+1)} - \ell_i{}^{(k)}$
14:    if update global gap then
15:        for $i := 1, \dots, n$ do
16:            Let $k_i := k + 1$
17:            Solve $\boldsymbol{y}_i^* := \operatorname{argmax}_{\boldsymbol{y} \in \mathcal{Y}_i} H_i(\boldsymbol{y}; \boldsymbol{w}^{(k_i)})$
18:            Let $\boldsymbol{w_s} := \frac{1}{\lambda n}\boldsymbol{\psi}_i(\boldsymbol{y}_i^*)$ and $\ell_{\boldsymbol{s}} := \frac{1}{n} L_i(\boldsymbol{y}_i^*)$
19:            $g_i^{(k_i)} := \lambda(\boldsymbol{w}_i^{(k_i)} - \boldsymbol{w_s})^\top \boldsymbol{w}^{(k_i)} - \ell_i^{(k_i)} + \ell_{\boldsymbol{s}}$
20:        end for
21:    end if
22: end for
```

Note that the three modifications proposed in our paper (gap sampling, caching, pairwise/away steps) can be straightforwardly put together in any combination. In our experimental study, we evaluate all the possibilities.

When using gap sampling or caching and to guarantee convergence, we have to do a full pass over the data every so often to refresh the global gap estimates and to compensate for the staleness effect. In the experiments, we perform this computation every 10 passes over the data (this is the "update global gap" condition in the algorithms). This global gap can also be used as a certificate (upper bound) on the current suboptimality. We thus use the same frequency of global gap computation (every 10 passes) when we run a SSVM solver with a specific convergence tolerance threshold. This is used in our regularization path algorithm which runs a SSVM solver up to a fixed convergence tolerance at each breakpoint.

In our description of the regularization path algorithms (Algorithm 6 and Algorithm 7), we explicitly describe how to update the active sets over the dual variables when the regularization parameter is updated. This is needed when using a SSVM solver that requires the active set over the dual variables (such as BCPFW or BCAFW). When using the simpler BCFW solver, then lines 14–17 of Algorithm 6 and lines 17–21 of Algorithm 7 can simply be omitted.

Algorithm 3 Block-coordinate pairwise Frank-Wolfe (BCPFW) algorithm for structured SVM

```
1: Let $\boldsymbol{w}^{(0)} := \boldsymbol{w}_i{}^{(0)} := \mathbf{0}$;  $\ell^{(0)} := \ell_i{}^{(0)} := 0$;
2: $\mathcal{S}_i{}^{(0)} := \{\boldsymbol{y}_i\}$;   // active sets
3: $\alpha_i^{(0)}(\boldsymbol{y}) := 0, \boldsymbol{y} \neq \boldsymbol{y}_i$;  $\alpha_i^{(0)}(\boldsymbol{y}_i) := 1$
4: for $k := 0, \dots, \infty$ do
5:     Pick $i$ at random in $\{1, \dots, n\}$
6:     Solve $\boldsymbol{y}_i^* := \operatorname{argmax}_{\boldsymbol{y} \in \mathcal{Y}_i} H_i(\boldsymbol{y}; \boldsymbol{w}^{(k)})$   // FW corner
7:     Let $\boldsymbol{w_s} := \frac{1}{\lambda n} \boldsymbol{\psi}_i(\boldsymbol{y}_i^*)$ and $\ell_{\boldsymbol{s}} := \frac{1}{n} L_i(\boldsymbol{y}_i^*)$
8:     Solve $\boldsymbol{y}_i^a := \operatorname{argmin}_{\boldsymbol{y} \in \mathcal{S}_i^{(k)}} H_i(\boldsymbol{y}; \boldsymbol{w}^{(k)})$   // away corner
9:     Let $\boldsymbol{w_a} := \frac{1}{\lambda n} \boldsymbol{\psi}_i(\boldsymbol{y}_i^a)$ and $\ell_{\boldsymbol{a}} := \frac{1}{n} L_i(\boldsymbol{y}_i^a)$
10:    Let $\boldsymbol{w_d} := \boldsymbol{w_s} - \boldsymbol{w_a}$ and $\ell_{\boldsymbol{d}} = \ell_{\boldsymbol{s}} - \ell_{\boldsymbol{a}}$
11:    Let $\gamma := \frac{-\lambda \boldsymbol{w_d}^\mathsf{T} \boldsymbol{w}^{(k)} + \ell_{\boldsymbol{d}}}{\lambda \|\boldsymbol{w_d}\|^2}$ and clip to $[0, \alpha_i^{(k)}(\boldsymbol{y}_i^a)]$
12:    Update $\boldsymbol{w}_i{}^{(k+1)} := \boldsymbol{w}_i{}^{(k)} + \gamma\, \boldsymbol{w_d}$
13:        and $\ell_i{}^{(k+1)} := \ell_i{}^{(k)} + \gamma\, \ell_{\boldsymbol{d}}$
14:    Update $\boldsymbol{w}^{(k+1)} := \boldsymbol{w}^{(k)} + \boldsymbol{w}_i{}^{(k+1)} - \boldsymbol{w}_i{}^{(k)}$
15:        and $\ell^{(k+1)} := \ell^{(k)} + \ell_i{}^{(k+1)} - \ell_i{}^{(k)}$
16:    Update $\mathcal{S}_i{}^{(k+1)} := \mathcal{S}_i{}^{(k)} \cup \{\boldsymbol{y}_i^*\}$
17:        and $\alpha_i^{(k+1)}(\boldsymbol{y}_i^a) := \alpha_i^{(k)}(\boldsymbol{y}_i^a) - \gamma$
18:        and $\alpha_i^{(k+1)}(\boldsymbol{y}_i^*) := \alpha_i^{(k)}(\boldsymbol{y}_i^*) + \gamma$
19:    if $\gamma = \alpha_i^{(k)}(\boldsymbol{y}_i^a)$ then
20:        Set $\mathcal{S}_i{}^{(k+1)} := \mathcal{S}_i{}^{(k+1)} \setminus \{\boldsymbol{y}_i^a\}$   // drop step
21:    end if
22: end for
```

Algorithm 4 Block-coordinate away-step Frank-Wolfe (BCAFW) algorithm for structured SVM

```
1: Let $\boldsymbol{w}^{(0)} := \boldsymbol{w}_i{}^{(0)} := \mathbf{0}$;  $\ell^{(0)} := \ell_i{}^{(0)} := 0$;
2: $\mathcal{S}_i^{(0)} := \{\boldsymbol{y}_i\}$;   // active sets
3: $\alpha_i^{(0)}(\boldsymbol{y}) := 0, \boldsymbol{y} \neq \boldsymbol{y}_i$;  $\alpha_i^{(0)}(\boldsymbol{y}_i) := 1$
4: for $k := 0, \dots, \infty$ do
5:     Pick $i$ at random in $\{1, \dots, n\}$
6:     Solve $\boldsymbol{y}_i^* := \operatorname{argmax}_{\boldsymbol{y} \in \mathcal{Y}_i} H_i(\boldsymbol{y}; \boldsymbol{w}^{(k)})$   // FW corner
7:     Let $\boldsymbol{w_s} := \frac{1}{\lambda n} \boldsymbol{\psi}_i(\boldsymbol{y}_i^*)$ and $\ell_{\boldsymbol{s}} := \frac{1}{n} L_i(\boldsymbol{y}_i^*)$
8:     Solve $\boldsymbol{y}_i^a := \operatorname{argmin}_{\boldsymbol{y} \in \mathcal{S}_i^{(k)}} H_i(\boldsymbol{y}; \boldsymbol{w}^{(k)})$   // away corner
9:     Let $\boldsymbol{w_a} := \frac{1}{\lambda n} \boldsymbol{\psi}_i(\boldsymbol{y}_i^a)$ and $\ell_{\boldsymbol{a}} := \frac{1}{n} L_i(\boldsymbol{y}_i^a)$
10:    Let $g_i^{FW} := \lambda(\boldsymbol{w}_i^{(k)} - \boldsymbol{w_s})^\mathsf{T} \boldsymbol{w}^{(k)} - \ell_i^{(k)} + \ell_{\boldsymbol{s}}$
11:    Let $g_i^{A} := \lambda(\boldsymbol{w_a} - \boldsymbol{w}_i^{(k)})^\mathsf{T} \boldsymbol{w}^{(k)} + \ell_i^{(k)} - \ell_{\boldsymbol{a}}$
12:    if $g_i^{FW} > g_i^A$ then   // FW step
13:        Let $\gamma := \frac{g_i^{FW}}{\lambda \|\boldsymbol{w}_i^{(k)} - \boldsymbol{w_s}\|^2}$ and clip to $[0, 1]$
14:        Update $\mathcal{S}_i{}^{(k+1)} := \mathcal{S}_i{}^{(k)} \cup \{\boldsymbol{y}_i^*\}$
15:            and $\alpha_i^{(k+1)}(\boldsymbol{y}) := (1 - \gamma)\alpha_i^{(k)}(\boldsymbol{y})$
16:            and $\alpha_i^{(k+1)}(\boldsymbol{y}_i^*) := \alpha_i^{(k+1)}(\boldsymbol{y}_i^*) + \gamma$
17:        Set $\mathcal{S}_i{}^{(k+1)} := \{\boldsymbol{y}_i^*\}$ if $\gamma = 1$
18:    else   // away step
19:        Let $\gamma := \frac{g_i^{A}}{\lambda \|\boldsymbol{w}_i^{(k)} - \boldsymbol{w_a}\|^2}$ and clip to $[0, \frac{\alpha_i(\boldsymbol{y}_i^a)}{1 - \alpha_i(\boldsymbol{y}_i^a)}]$
20:        Update $\alpha_i^{(k+1)}(\boldsymbol{y}) := (1 + \gamma)\alpha_i^{(k)}(\boldsymbol{y})$
21:            and $\alpha_i^{(k+1)}(\boldsymbol{y}_i^a) := \alpha_i^{(k+1)}(\boldsymbol{y}_i^a) - \gamma$
22:            and $\mathcal{S}_i{}^{(k+1)} := \mathcal{S}_i{}^{(k)} \setminus \{\boldsymbol{y}_i^a\}$ if $\alpha_i^{(k+1)}(\boldsymbol{y}_i^a) = 0$
23:    end if
24:    Update $\boldsymbol{w}_i{}^{(k+1)} := \boldsymbol{w}_i{}^{(k)} + \gamma\, \boldsymbol{w_d}$
25:        and $\ell_i{}^{(k+1)} := \ell_i{}^{(k)} + \gamma\, \ell_{\boldsymbol{d}}$
26:    Update $\boldsymbol{w}^{(k+1)} := \boldsymbol{w}^{(k)} + \boldsymbol{w}_i{}^{(k+1)} - \boldsymbol{w}_i{}^{(k)}$
27:        and $\ell^{(k+1)} := \ell^{(k)} + \ell_i{}^{(k+1)} - \ell_i{}^{(k)}$
28: end for
```

E. Proof of Theorem 2 (convergence of BCFW with gap sampling)

Lemma 4 (Expected block descent lemma). *Let $g_j(\boldsymbol{\alpha}^{(k)})$ be the block gap for block j for the iterate $\boldsymbol{\alpha}^{(k)}$. Let $\boldsymbol{\alpha}^{(k+1)}$ be obtained by sampling a block i with probability p_i and then doing a (block) FW step with line-search on this block, starting from $\boldsymbol{\alpha}^{(k)}$. Consider* any *set of scalars $\gamma_j \in [0, 1]$, $j = 1, \dots, n$, which do not depend on the chosen block i. Then in conditional expectation over the random choice of block i with probabilities p_i, it holds:*

$$\mathbf{E}\big[f(\boldsymbol{\alpha}^{(k+1)}) \,|\, \boldsymbol{\alpha}^{(k)}\big] \leq f(\boldsymbol{\alpha}^{(k)}) - \sum_{i=1}^{n} \gamma_i p_i g_i(\boldsymbol{\alpha}^{(k)}) + \tfrac{1}{2} \sum_{i=1}^{n} \gamma_i^2 p_i C_f^{(i)}. \tag{20}$$

Proof. The proof is analogous to the proof of Lemma 3, but being careful with the expectation. Let block i be the chosen one that defined $\boldsymbol{\alpha}^{(k+1)}$ and let $\boldsymbol{\alpha}_\gamma := \boldsymbol{\alpha} + \gamma(\boldsymbol{s}_{[i]} - \boldsymbol{\alpha}_{[i]})$, where $\mathrm{s}_{(i)} \in \mathcal{M}^{(i)}$ is the FW corner on block i and $\mathrm{s}_{[i]} \in \mathbb{R}^m$ is its zero-padded version. By the line-search, we have $f(\boldsymbol{\alpha}^{(k+1)}) \leq f(\boldsymbol{\alpha}_\gamma)$ for any $\gamma \in [0, 1]$. By using $\gamma = \gamma_i$ in the bound (13) provided in the curvature Definition 2, and by the definition of the Frank-Wolfe gap, we get:

$$\begin{aligned} f(\boldsymbol{\alpha}^{(k+1)}) \leq f(\boldsymbol{\alpha}_{\gamma_i}) &= f(\boldsymbol{\alpha}^{(k)} + \gamma_i(\mathrm{s}_{[i]} - \boldsymbol{\alpha}_{[i]}^{(k)})) \\ &\leq f(\boldsymbol{\alpha}^{(k)}) + \gamma_i g_i(\boldsymbol{\alpha}^{(k)}) + \tfrac{\gamma_i^2}{2} C_f^{(i)}. \end{aligned}$$

Taking the expectation of the bound with respect to i, conditioned on $\boldsymbol{\alpha}^{(k)}$, proves the lemma. □

Definition 5. *The* nonuniformity 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 $\boldsymbol{x}$.

Lemma 6. *Let $\boldsymbol{x} \in \mathbb{R}_+^n$. The following relation between its ℓ_1-norm and ℓ_2-norm holds:*

$$\|\boldsymbol{x}\|_2 = \tfrac{\chi(\boldsymbol{x})}{\sqrt{n}} \|\boldsymbol{x}\|_1 \,.$$

Proof. We have that

$$\mathrm{Var}\left[\boldsymbol{p}\right] = \mathbf{E}\left[\boldsymbol{p}^2\right] - \mathbf{E}\left[\boldsymbol{p}\right]^2 = \tfrac{1}{n}\|\boldsymbol{p}\|_2^2 - \tfrac{1}{n^2}. \qquad (21)$$

Combining (21) and Definition 5 we prove the lemma. □

Remark. For any $\boldsymbol{x} \in \mathbb{R}_+^n$, the quantity $\chi(\boldsymbol{x})$ always belongs to the segment $[1, \sqrt{n}]$. We have $\chi(\boldsymbol{x}) = 1$ when all the elements of $\boldsymbol{x}$ are equal and $\chi(\boldsymbol{x}) = \sqrt{n}$ when all the elements, except one, equal zero.

Theorem 2. *Assume that at each iterate $\boldsymbol{\alpha}^{(k)}$, $k \geq 0$, BCFW with gap sampling (Algorithm 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 $\boldsymbol{\alpha}^* \in \mathcal{M}$ is a solution of problem (3), $h_0 := f(\boldsymbol{\alpha}^{(0)}) - f(\boldsymbol{\alpha}^*)$ is the suboptimality at the starting point of the algorithm, the constant $C_f^{\otimes} := \sum_{i=1}^n C_f^{(i)}$ is the sum of the curvature constants, and the constant $\chi^{\otimes}$ is an upper bound on $\mathbf{E}\Big[\frac{\chi(C_f^{(:)})}{\chi(\boldsymbol{g}_:(\boldsymbol{\alpha}^{(k)}))^3}\Big]$, which quantities the amount of non-uniformity of the $C_f^{(i)}$'s in relationship to the non-uniformity of the gaps obtained during the algorithm. The expectations are taken over the random choice of the sampled block at iterations $1, \dots, k$ of the algorithm.*

Proof. Starting from Lemma 4 with $\gamma_i := \gamma$ for some γ to be determined later and $p_i := \frac{g_i}{g}$ where $g_i := g_i(\boldsymbol{\alpha}^{(k)})$ and $g := g(\boldsymbol{\alpha}^{(k)})$, we get

$$\begin{aligned}\mathbf{E}\big[f(\boldsymbol{\alpha}^{(k+1)})\,|\,\boldsymbol{\alpha}^{(k)}\big] \leq f(\boldsymbol{\alpha}^{(k)}) - \gamma\sum_{i=1}^n \tfrac{g_i^2}{g} \\ + \tfrac{\gamma^2}{2}\sum_{i=1}^n C_f^{(i)}\tfrac{g_i}{g}.\end{aligned} \qquad (22)$$

The Cauchy-Schwarz inequality bounds the dot product between the vectors of curvature constants $\boldsymbol{c} := C_f^{(:)} := (C_f^{(i)})_{i=1}^n$ and block gaps $\boldsymbol{g} := \boldsymbol{g}_:(\boldsymbol{\alpha}^{(k)}) := (g_i)_{i=1}^n$

$$\sum_{i=1}^n C_f^{(i)} g_i \leq \|\boldsymbol{c}\|_2\,\|\boldsymbol{g}\|_2\,. \qquad (23)$$

Combining (23) and the result of Lemma 6 for the vectors of curvature constants and block gaps (with $\|\boldsymbol{g}\|_1 = g$ and $\|\boldsymbol{c}\|_1 = C_f^{\otimes}$), we can further bound (22):

$$\begin{aligned}\mathbf{E}\big[f(\boldsymbol{\alpha}^{(k+1)})\,|\,\boldsymbol{\alpha}^{(k)}\big] \leq f(\boldsymbol{\alpha}^{(k)}) - \tfrac{\gamma g}{n}\chi(\boldsymbol{g})^2 \\ + \tfrac{\gamma^2}{2n}\chi(\boldsymbol{g})\,\chi(\boldsymbol{c})\,C_f^{\otimes}.\end{aligned} \qquad (24)$$

Subtracting the minimal function value $f(\boldsymbol{\alpha}^*)$ from both sides of (24) and by using $h(\boldsymbol{\alpha}^{(k)}) := f(\boldsymbol{\alpha}^{(k)}) - f(\boldsymbol{\alpha}^*) \leq g$, we bound the conditional expectation of the suboptimality h with

$$\begin{aligned}\mathbf{E}[h(\boldsymbol{\alpha}^{(k+1)})\mid\boldsymbol{\alpha}^{(k)}] \leq h(\boldsymbol{\alpha}^{(k)}) - \tfrac{\gamma}{n}\chi(\boldsymbol{g})^2\,h(\boldsymbol{\alpha}^{(k)}) \\ + \tfrac{\gamma^2}{2n}\chi(\boldsymbol{g})\,\chi(\boldsymbol{c})\,C_f^{\otimes}\end{aligned} \qquad (25)$$

which is analogous to (Lacoste-Julien et al., 2013, Eq. (20)). In what follows, we use the modified induction technique of (Lacoste-Julien et al., 2013, Proof of Theorem C.1).

By induction, we are going to prove the following upper bound on the unconditional expectation of the suboptimality h:

$$\mathbf{E}\big[h(\boldsymbol{\alpha}^{(k)})\big] \leq \tfrac{2nC}{k+2n}, \quad \text{for } k \geq 0, \qquad (26)$$

that corresponds to the statement of the theorem with $C := C_f^{\otimes}\chi^{\otimes} + h_0$.

The basis of the induction $k = 0$ follows immediately from the definition of C, given that $C_f^{\otimes} \geq 0$ and $\chi^{\otimes} > 0$.

Consider the induction step. Assume that (26) is satisfied for $k \geq 0$. With a particular choice of step size $\gamma := \frac{2n}{\chi(\boldsymbol{g})^2(k+2n)} \in [0,1]$ (which does not depend on the picked i), we rewrite the bound (25) on the conditional expectation as

$$\begin{aligned}\mathbf{E}[h(\boldsymbol{\alpha}^{(k+1)})\mid\boldsymbol{\alpha}^{(k)}] \leq \Big(1 - \tfrac{2}{k+2n}\Big)\,h(\boldsymbol{\alpha}^{(k)}) \\ + \tfrac{2n}{(k+2n)^2}\,\tfrac{\chi(\boldsymbol{c})\,C_f^{\otimes}}{\chi(\boldsymbol{g})^3}\,.\end{aligned} \qquad (27)$$

Taking the unconditional expectation of (27), then the induction assumption (26) and the definition of $\chi^{\otimes}$ give us the deterministic inequality

$$\begin{aligned}\mathbf{E}[h(\boldsymbol{\alpha}^{(k+1)})] \leq \Big(1 - \tfrac{2}{k+2n}\Big)\tfrac{2nC}{k+2n} \\ + \tfrac{2n}{(k+2n)^2}\chi^{\otimes}\,C_f^{\otimes}.\end{aligned} \qquad (28)$$

Bounding $\chi^{\otimes}\,C_f^{\otimes}$ by C and rearranging the terms gives

$$\begin{aligned}\mathbf{E}[h(\boldsymbol{\alpha}^{(k+1)})] &\leq \tfrac{2nC}{k+2n}\Big(1 - \tfrac{2}{k+2n} + \tfrac{1}{k+2n}\Big) \\ &= \tfrac{2nC}{k+2n}\,\tfrac{k+2n-1}{k+2n} \\ &\leq \tfrac{2nC}{k+2n}\,\tfrac{k+2n}{k+2n+1} \\ &= \tfrac{2nC}{(k+1)+2n}\,,\end{aligned}$$

which completes the induction proof. □

Comparison with uniform sampling. We now compare the rates obtained by Theorem 2 for BCFW with gap sampling and by Theorem 1 for BCFW with uniform sampling. The only difference is in the constants: Theorem 2 has $C_f^{\otimes}\chi^{\otimes}$ and Theorem 1 has $C_f^{\otimes}$.

Recall that by definition

$$\chi^{\otimes} = \max_k\ \mathbf{E}\Big[\tfrac{\chi(C_f^{(:)})}{\chi(\boldsymbol{g}_:(\boldsymbol{\alpha}^{(k)}))^3}\Big]$$

In the best case for gap sampling, the curvature constants are uniform, $\chi(C_f^{(:)}) = 1$, and the gaps are nonuniform $\chi(\boldsymbol{g}_:(\boldsymbol{\alpha}^{(k)})) \approx \sqrt{n}$. Thus, $\chi^{\otimes} \approx \frac{1}{n\sqrt{n}}$.

In the worst case for gap sampling, the curvature constants are very non-uniform, $\chi(C_f^{(:)}) \approx \sqrt{n}$. The constant for gap sampling is still better if the gaps are non-uniform enough, i.e., $\chi(\boldsymbol{g}_{:}(\boldsymbol{\alpha}^{(k)})) \geq n^{\frac{1}{6}}$.

We note that to design a sampling scheme that always dominates uniform sampling (in terms of bounds at least), we would need to include the $C_f^{(i)}$'s in the sampling scheme (as was essentially done by Csiba et al. (2015) for SDCA). Unfortunately, computing good estimates for $C_f^{(i)}$'s is too expensive for structured SVM, thus motivating our simpler yet practically efficient scheme. See also the discussion after (10).

F. Proof of Theorem 3 (convergence of BCFW with caching)

Theorem 3. *Let $\tilde{\nu} := \frac{1}{n}\nu \leq 1$. Then, for each $k \geq 0$, the iterate $\boldsymbol{\alpha}^{(k)}$ of Algorithm 5 satisfies $\mathbf{E}\big[f(\boldsymbol{\alpha}^{(k)})\big] - f(\boldsymbol{\alpha}^*) \leq \frac{2n}{\tilde{\nu}k+2n}\big(\frac{1}{\tilde{\nu}}C_f^{\otimes} + h_0\big)$ where $\boldsymbol{\alpha}^* \in \mathcal{M}$ is a solution of 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 (see Definition 2) of f with respect to the domains $\mathcal{M}^{(i)}$ of individual blocks. The expectation is taken over the random choice of the sampled blocks at iterations $1, \ldots, k$ of the algorithm.*

Proof. The key observation of the proof consists in the fact that the combined oracle (the cache oracle in the case of a cache hit and the max oracle in the case of a cache miss) closely resembles an oracle with multiplicative approximation error (Lacoste-Julien et al., 2013, Eq. (12) of Appendix C).

In the case of a cache hit, Definition 2 of curvature constant for any step size $\gamma \in [0,1]$ gives us

$$\begin{aligned}
&f(\boldsymbol{\alpha}_\gamma^{(k+1)}) := f(\boldsymbol{\alpha}^{(k)} + \gamma(\boldsymbol{c}_{[i]} - \boldsymbol{\alpha}_{[i]}^{(k)})) \\
&\leq f(\boldsymbol{\alpha}^{(k)}) + \gamma\langle \boldsymbol{c}_{(i)} - \boldsymbol{\alpha}_{(i)}^{(k)}, \nabla_{(i)} f(\boldsymbol{\alpha}^{(k)})\rangle + \tfrac{\gamma^2}{2}C_f^{(i)} \\
&= f(\boldsymbol{\alpha}^{(k)}) - \gamma \hat{g}_i^{(k)} + \tfrac{\gamma^2}{2}C_f^{(i)} \\
&\leq f(\boldsymbol{\alpha}^{(k)}) - \gamma\tilde{\nu} g^{(k_0)} + \tfrac{\gamma^2}{2}C_f^{(i)}
\end{aligned}$$

where the corner $\boldsymbol{c}_{(i)} \in \mathcal{M}^{(i)}$ and its zero-padded version $\boldsymbol{c}_{[i]} \in \mathbb{R}^m$ are provided by the cache oracle, and $\tilde{\nu} = \frac{1}{n}\nu$ is the constant controlling the global part of the cache-hit criterion. In the case of a cache miss, similarly to Lemma 3, we get

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

Combining the two cases we get

$$f(\boldsymbol{\alpha}_\gamma^{(k+1)}) \leq f(\boldsymbol{\alpha}^{(k)}) - \gamma \tilde{g}_i^{(k)} + \tfrac{\gamma^2}{2}C_f^{(i)} \qquad (29)$$

where

$$\tilde{g}_i^{(k)} := [i \text{ is a cache miss}]g_i^{(k)} + [i \text{ is a cache hit}]\tilde{\nu} g^{(k_0)}.$$

Algorithm 5 Block-coordinate Frank-Wolfe (BCFW) algorithm with cache for structured SVM

```
1: Let $\boldsymbol{w}^{(0)} := \boldsymbol{w}_i^{(0)} := \mathbf{0}$;  $\ell^{(0)} := \ell_i^{(0)} := 0$;  $\mathcal{C}_i := \{\boldsymbol{y}_i\}$;
2: $g^{(0)} := g_i^{(0)} = +\infty$
3: $k_0 := k_i := 0$ ;        // the last time $g$ / $g_i$ was computed
4: for $k := 0, \ldots, \infty$ do
5:     Pick $i$ at random in $\{1, \ldots, n\}$  // either uniform or
6:                 with probability $\propto g_i^{(k_i)}$ for gap sampling
7:     Solve $\boldsymbol{y}_i^c := \operatorname{argmax}_{\boldsymbol{y} \in \mathcal{C}_i} H_i(\boldsymbol{y}; \boldsymbol{w})$ // cache corner
8:     Let $\boldsymbol{w_c} := \frac{\boldsymbol{\psi}_i(\boldsymbol{y}_i^c)}{\lambda n}$  and  $\ell_{\boldsymbol{c}} := \frac{1}{n}L_i(\boldsymbol{y}_i^c)$
9:     Let $\hat{g}_i^{(k)} := \lambda(\boldsymbol{w}_i^{(k)} - \boldsymbol{w_c})^\top \boldsymbol{w}^{(k)} - \ell_i^{(k)} + \ell_{\boldsymbol{c}}$
10:    if $\hat{g}_i^{(k)} \geq \max(F g_i^{(k_i)}, \frac{\nu}{n} g^{(k_0)})$ then      // cache hit
11:        $\boldsymbol{w_s} := \boldsymbol{w_c}$, $\ell_{\boldsymbol{s}} := \ell_{\boldsymbol{c}}$, $\hat{g}_i := \hat{g}_i^{(k)}$
12:    else                                         // cache miss
13:        Solve $\boldsymbol{y}_i^* := \operatorname{argmax}_{\boldsymbol{y} \in \mathcal{Y}_i} H_i(\boldsymbol{y}; \boldsymbol{w}^{(k)})$  // FW corner
14:        Let $\boldsymbol{w_s} := \frac{1}{\lambda n}\boldsymbol{\psi}_i(\boldsymbol{y}_i^*)$  and  $\ell_{\boldsymbol{s}} := \frac{1}{n}L_i(\boldsymbol{y}_i^*)$
15:        Let $g_i^{(k)} := \lambda(\boldsymbol{w}_i^{(k)} - \boldsymbol{w_s})^\top \boldsymbol{w}^{(k)} - \ell_i^{(k)} + \ell_{\boldsymbol{s}}$
16:        Set $k_i := k$, $\hat{g}_i := g_i^{(k)}$
17:        Update $\mathcal{C}_i := \mathcal{C}_i \cup \{\boldsymbol{y}_i^*\}$
18:    end if
19:    Let $\gamma := \frac{\hat{g}_i}{\lambda\|\boldsymbol{w}_i^{(k)} - \boldsymbol{w_s}\|^2}$ and clip to $[0,1]$
20:    Update $\boldsymbol{w}_i^{(k+1)} := (1-\gamma)\boldsymbol{w}_i^{(k)} + \gamma\,\boldsymbol{w_s}$
21:        and $\ell_i^{(k+1)} := (1-\gamma)\ell_i^{(k)} + \gamma\,\ell_{\boldsymbol{s}}$
22:    Update $\boldsymbol{w}^{(k+1)} := \boldsymbol{w}^{(k)} + \boldsymbol{w}_i^{(k+1)} - \boldsymbol{w}_i^{(k)}$
23:        and $\ell^{(k+1)} := \ell^{(k)} + \ell_i^{(k+1)} - \ell_i^{(k)}$
24:    if update global gap then
25:        Let $g^{(k_0)} := 0$, $k_0 := k+1$
26:        for $i := 1, \ldots, n$ do
27:            Solve $\boldsymbol{y}_i^* := \operatorname{argmax}_{\boldsymbol{y} \in \mathcal{Y}_i} H_i(\boldsymbol{y}; \boldsymbol{w}^{(k_0)})$
28:            Let $\boldsymbol{w_s} := \frac{1}{\lambda n}\boldsymbol{\psi}_i(\boldsymbol{y}_i^*)$  and  $\ell_{\boldsymbol{s}} := \frac{1}{n}L_i(\boldsymbol{y}_i^*)$
29:            $g^{(k_0)} \mathrel{+}= \lambda(\boldsymbol{w}_i^{(k_0)} - \boldsymbol{w_s})^\top \boldsymbol{w}^{(k_0)} - \ell_i^{(k_0)} + \ell_{\boldsymbol{s}}$
30:            Set $k_i := k_0$
31:        end for
32:    end if
33: end for
```

Subtracting $f(\boldsymbol{\alpha}^*)$ from both sides of (29) and taking the expectation of (29) w.r.t. the block index i we get

$$\mathbf{E}\big[h(\boldsymbol{\alpha}_\gamma^{(k+1)}) \,|\, \boldsymbol{\alpha}^{(k)}\big] \leq h(\boldsymbol{\alpha}^{(k)}) - \tfrac{\gamma}{n}\tilde{g}^{(k)} + \tfrac{\gamma^2}{2n}C_f^{\otimes} \qquad (30)$$

where $h(\boldsymbol{\alpha}) := f(\boldsymbol{\alpha}) - f(\boldsymbol{\alpha}^*)$ is the suboptimality of the function f and $\tilde{g}^{(k)} := \sum_{i=1}^n \tilde{g}_i^{(k)}$. We know that the duality gap upper-bounds the suboptimality, i.e., $g(\boldsymbol{\alpha}) \geq h(\boldsymbol{\alpha})$, and that cache miss steps, as well as cache hit steps, always decrease suboptimality, i.e., $h(\boldsymbol{\alpha}^{(k)}) \leq h(\boldsymbol{\alpha}^{(k_0)})$.

If at iteration k there is at least one cache hit, then we can bound the quantity $\tilde{g}^{(k)}$ from below:

$$\tilde{g}^{(k)} \geq \tilde{\nu} g(\boldsymbol{\alpha}^{(k_0)}) \geq \tilde{\nu} h(\boldsymbol{\alpha}^{(k_0)}) \geq \tilde{\nu} h(\boldsymbol{\alpha}^{(k)}). \qquad (31)$$

Algorithm 6 INIT-REG-PATH: Initialization of the regularization path for structured SVM

```
input $\kappa$, tolerance $\varepsilon$
 1: $\boldsymbol{w} := \boldsymbol{w}_i := 0$;  $\ell := \ell_i := 0$;  $\tilde{\boldsymbol{\psi}} := 0$
 2: for $i := 1, \dots, n$ do
 3:     $\tilde{\boldsymbol{y}}_i := \operatorname{argmax}_{\boldsymbol{y} \in \mathcal{Y}_i} H_i(\boldsymbol{y}; \mathbf{0})$
 4:     $\ell_i := \frac{1}{n} L(\boldsymbol{y}_i, \tilde{\boldsymbol{y}}_i)$
 5:     $\ell := \ell + \ell_i$
 6:     $\tilde{\boldsymbol{\psi}} := \tilde{\boldsymbol{\psi}} + \frac{1}{n} \boldsymbol{\psi}(\tilde{\boldsymbol{y}}_i)$
 7: end for
 8: for $i := 1, \dots, n$ do
 9:     $\theta_i := \max_{\boldsymbol{y} \in \mathcal{Y}_i} \big( -\tilde{\boldsymbol{\psi}}^\mathsf{T} \boldsymbol{\psi}(\boldsymbol{y}) \big)$
10: end for
11: Let $\lambda^\infty := \frac{1}{\kappa\varepsilon} \bigg( \|\tilde{\boldsymbol{\psi}}\|^2 + \frac{1}{n} \sum_{i=1}^n \theta_i \bigg)$
12: Let $\boldsymbol{w} := \frac{1}{\lambda^\infty} \tilde{\boldsymbol{\psi}}$;  $\boldsymbol{w}_i := \frac{1}{n\lambda^\infty} \boldsymbol{\psi}(\tilde{\boldsymbol{y}}_i)$
13: for $i := 1, \dots, n$ do $g_i := \frac{1}{n\lambda^\infty} \theta_i + \lambda^\infty \boldsymbol{w}_i^\mathsf{T} \boldsymbol{w}$
14: for $i := 1, \dots, n$ do                         // optional
15:     $\mathcal{S}_i := \{\tilde{\boldsymbol{y}}_i\}$
16:     $\alpha_i(\tilde{\boldsymbol{y}}_i) := 1$ and $\alpha_i(\boldsymbol{y}) := 0$ for $\boldsymbol{y} \neq \tilde{\boldsymbol{y}}_i$
17: end for
18: return $\boldsymbol{w}$, $\boldsymbol{w}_i$, $\ell$, $\ell_i$, $g_i$, $\lambda^\infty$, $\mathcal{S}_i$, $\boldsymbol{\alpha}$
```

In the case of no cache hits, we have

$$\tilde{g}^{(k)} = g(\boldsymbol{\alpha}^{(k)}) \geq h(\boldsymbol{\alpha}^{(k)}) \geq \tilde{\nu} h(\boldsymbol{\alpha}^{(k)})$$

where the last inequality holds because $\tilde{\nu} \leq 1$. Applying the lower bound on $\tilde{g}^{(k)}$ to (30), we get

$$\begin{aligned} \mathbf{E}\big[h(\boldsymbol{\alpha}_\gamma^{(k+1)}) \,|\, \boldsymbol{\alpha}^{(k)}\big] \leq h(\boldsymbol{\alpha}^{(k)}) - \tfrac{\gamma\tilde{\nu}}{n} h(\boldsymbol{\alpha}^{(k)}) \\ + \tfrac{\gamma^2}{2n} C_f^\otimes. \end{aligned} \tag{32}$$

Inequality (32) is identical to the inequality (Lacoste-Julien et al., 2013, Eq. (20)) in the proof of convergence of BCFW with a multiplicative approximation error in the oracle. We recopy their argument below for reference to finish the proof. First, we take the expectation of (32) w.r.t. the choice of previous blocks:

$$\mathbf{E}\big[h(\boldsymbol{\alpha}_\gamma^{(k+1)})\big] \leq (1 - \tfrac{\gamma\tilde{\nu}}{n}) \mathbf{E}\big[h(\boldsymbol{\alpha}^{(k)})\big] + \tfrac{\gamma^2}{2n} C_f^\otimes. \tag{33}$$

Following the proof of Theorem C.1 in Lacoste-Julien et al. (2013), we prove the bound of Theorem 3 by induction. The induction hypothesis consists in inequality

$$\mathbf{E}\big[h(\boldsymbol{\alpha}^{(k)})\big] \leq \tfrac{2nC}{\tilde{\nu}k+2n} \;\text{ for } k \geq 0$$

where $C := \big(\frac{1}{\tilde{\nu}} C_f^\otimes + h_0\big)$.

The base-case $k = 0$ follows directly from $C \geq h_0$. We now prove the induction step. Assume that the hypothesis is true for a given $k \geq 0$. Let us now prove that the hypothesis is true for $k + 1$. We use inequality (32) with the step size $\gamma_k := \frac{2n}{\tilde{\nu}k+2n} \in [0, 1]$:

$$\begin{aligned} \mathbf{E}\big[h(\boldsymbol{\alpha}_{\gamma_k}^{(k+1)})\big] &\leq (1 - \tfrac{\gamma_k\tilde{\nu}}{n}) \mathbf{E}\big[h(\boldsymbol{\alpha}^{(k)})\big] + (\gamma_k)^2 \tfrac{C\tilde{\nu}}{2n} \\ &= (1 - \tfrac{2\tilde{\nu}}{\tilde{\nu}k+2n}) \mathbf{E}\big[h(\boldsymbol{\alpha}^{(k)})\big] + (\tfrac{2n}{\tilde{\nu}k+2n})^2 \tfrac{C\tilde{\nu}}{2n} \\ &\leq (1 - \tfrac{2\tilde{\nu}}{\tilde{\nu}k+2n}) \tfrac{2nC}{\tilde{\nu}k+2n} + (\tfrac{1}{\tilde{\nu}k+2n})^2 2nC\tilde{\nu} \end{aligned}$$

where, in the first line, we use inequality $C_f^\otimes \leq C\tilde{\nu}$, and, in the last line, we use the induction hypothesis for $\mathbf{E}\big[h(\boldsymbol{\alpha}^{(k)})\big]$.

By rearranging the terms, we have

$$\begin{aligned} \mathbf{E}\big[h(\boldsymbol{\alpha}^{(k+1)})\big] \leq& \tfrac{2nC}{\tilde{\nu}k+2n} \left(1 - \tfrac{2\tilde{\nu}}{\tilde{\nu}k+2n} + \tfrac{\tilde{\nu}}{\tilde{\nu}k+2n}\right) \\ =& \tfrac{2nC}{\tilde{\nu}k+2n} \tfrac{\tilde{\nu}k+2n-\tilde{\nu}}{\tilde{\nu}k+2n} \\ \leq& \tfrac{2nC}{\tilde{\nu}k+2n} \tfrac{\tilde{\nu}k+2n}{\tilde{\nu}k+2n+\tilde{\nu}} \\ =& \tfrac{2nC}{\tilde{\nu}(k+1)+2n}\,, \end{aligned}$$

which finishes the proof. □

Algorithm 7 Regularization path for structured SVM

```
input $\kappa$, tolerance $\varepsilon$, $\lambda_{min}$
 1: Initialize regularization path using Algorithm 6.
    $\{\boldsymbol{w}^0, \boldsymbol{w}_i^0, \ell^0, \ell_i^0, g_i^0, \lambda^0, \mathcal{S}_i^0, \boldsymbol{\alpha}^0\}$:=INIT-REG-PATH $(\kappa, \varepsilon)$
 2: $J := 0$
 3: repeat
 4:     For $i := 1, \dots, n$ do $\delta_i := \ell_i^J - \lambda^J \langle \boldsymbol{w}^J, \boldsymbol{w}_i^J \rangle$
 5:     Compute excess gap $\tau := \varepsilon - \sum_{i=1}^n g_i^J$
 6:     Let $\Delta := \sum_i^n \delta_i$
 7:     if $\Delta \leq \tau$ then
 8:         Let $\rho := 1 - \frac{\tau}{\Delta}$
 9:     else
10:         $\boldsymbol{w}^J$ is $\varepsilon$-approximate for any $\lambda < \lambda^J$
11:         return $\{\lambda^j\}_{j=0}^J$, $\{\boldsymbol{w}^j\}_{j=0}^J$
12:     end if
13:     Let $\lambda^{J+1} := \rho\lambda^J$, $\ell^{J+1} := \rho\ell^J$,
14:     for $i := 1, \dots, n$ do          // update gaps using (11)
15:         Let $g_i^{J+1} := g_i^J + (1-\rho)\delta_i$ and $\ell_i^{J+1} := \rho\ell_i^J$
16:     end for
17:     for $i := 1, \dots, n$ do          // optional: update duals
18:         $\mathcal{S}_i^{J+1} := \mathcal{S}_i^J \cup \{\boldsymbol{y}_i\}$
19:         $\alpha_i^{J+1}(\boldsymbol{y}) := \rho\alpha_i^J(\boldsymbol{y})$ for $\boldsymbol{y} \in \mathcal{S}_i^{J+1} \setminus \{\boldsymbol{y}_i\}$
20:         $\alpha_i^{J+1}(\boldsymbol{y}_i) := 1 - \sum_{\boldsymbol{y} \in \mathcal{S}_i^{J+1} \setminus \{\boldsymbol{y}_i\}} \alpha_i^{J+1}(\boldsymbol{y})$
21:     end for
22:     Run SSVM-optimizer with tolerance $\kappa\,\varepsilon$
23:       to update $\boldsymbol{w}^{J+1}$, $\boldsymbol{w}_i^{J+1}$, $\ell^{J+1}$, $\ell_i^{J+1}$, $g_i^{J+1}$, $\mathcal{S}_i^{J+1}$, $\boldsymbol{\alpha}^{J+1}$
      // to have $\varepsilon$-appr. path, gaps $g_i^{J+1}$ have to be exact
24:     $J := J + 1$
25: until $\lambda^{J+1} < \lambda_{min}$
26: return $\{\lambda^j\}_{j=0}^J$, $\{\boldsymbol{w}^j\}_{j=0}^J$
```

G. Convergence of BCPFW and BCAFW

In this section, we prove Theorem 4 that states that the suboptimality error on (3) decreases geometrically in expectation for BCPFW and BCAFW for the iterates at which *no* block would have a drop step, i.e., when no atom would be removed from the active sets. We follow closely the notation and the results from Lacoste-Julien & Jaggi (2015) where the global linear convergence of the (batch) pairwise FW (PFW) and away-step FW (AFW) algorithms was shown. The main insight to get our result is that the "pairwise FW gap" decomposes also as a sum of block gaps. We give our result for the following more general setting (the block-separable analog of the setup in Appendix F of Lacoste-Julien & Jaggi (2015)):

$$\begin{aligned}\min_{\boldsymbol{\alpha}\in\mathcal{M}} f(\boldsymbol{\alpha}) \quad \text{with} \quad & f(\boldsymbol{\alpha}) := q(A\boldsymbol{\alpha}) + \boldsymbol{b}^\top \boldsymbol{\alpha} \\ \text{and} \quad & \mathcal{M} = \mathcal{M}^{(1)} \times \cdots \times \mathcal{M}^{(n)},\end{aligned} \tag{34}$$

where q is a strongly convex function, and $\mathcal{M}^{(i)} := \operatorname{conv}(\mathcal{A}^{(i)})$ for each i, where $\mathcal{A}^{(i)} \subseteq \mathbb{R}^{m_i}$ is a *finite* set of vectors (called *atoms*). In other words, each $\mathcal{M}^{(i)}$ is a polytope. For the example of the dual SSVM objective (3), $q(\cdot) := \frac{\lambda}{2}\|\cdot\|^2$ and $\mathcal{A}^{(i)}$ are the corners of a probability simplex in $m_i := |\mathcal{Y}_i|$ dimensions.

Suppose that we maintain an active set $\mathcal{S}_i$ for each block (as in the BCPFW algorithm). We first relate the batch PFW direction with the block PFW directions, as well as their respective batch and blockwise *PFW gaps* (the PFW gap is replacing the FW gap (5) in the analysis of PFW).

Definition 7 (Block PFW gaps)**.** *Consider the problem* (34) *and suppose that the point $\boldsymbol{\alpha}$ has each of its block $\boldsymbol{\alpha}_{(i)}$ with current active set $\mathcal{S}_i \subseteq \mathcal{A}^{(i)}$.*[10] *We define the corresponding* batch PFW gap *at $\boldsymbol{\alpha}$ with active set $\mathcal{S} := \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$ as:*

$$\begin{aligned} g^{\text{PFW}}(\boldsymbol{\alpha};\mathcal{S}) &:= \max_{\boldsymbol{s}\in\mathcal{M},\boldsymbol{v}\in\mathcal{S}} \langle -\nabla f(\boldsymbol{\alpha}),\, \boldsymbol{s}-\boldsymbol{v}\rangle \qquad (35)\\ &= \max_{\boldsymbol{s}\in\mathcal{M},\boldsymbol{v}\in\mathcal{S}} \sum_i \langle -\nabla_{(i)} f(\boldsymbol{\alpha}),\, \boldsymbol{s}_{(i)}-\boldsymbol{v}_{(i)}\rangle \\ &= \sum_i \underbrace{\max_{\substack{\boldsymbol{s}_{(i)}\in\mathcal{M}^{(i)}\\ \boldsymbol{v}_{(i)}\in\mathcal{S}_i}} \langle -\nabla_{(i)} f(\boldsymbol{\alpha}),\, \boldsymbol{s}_{(i)}-\boldsymbol{v}_{(i)}\rangle}\\ &=: \sum_i \qquad g_i^{\text{PFW}}(\boldsymbol{\alpha};\mathcal{S}_i), \qquad (36)\end{aligned}$$

where g_i^{PFW} is the PFW gap for block i. We recognize that the maximizing arguments for g_i^{PFW} are the FW corner $\boldsymbol{s}_{(i)}$ and the away corner $\boldsymbol{v}_{(i)}$ for block i that one would obtain when running BCPFW on this block.

We note that by maintaining independent active sets $\mathcal{S}_i$ for each block, the number of potential away corner combinations is exponential in the number of blocks, yielding many more possible directions of movement than in the batch PFW algorithm where the number of away corners is bounded by the number of iterations. Moreover, suppose that we have an explicit expansion for each block $\boldsymbol{\alpha}_{(i)}$ as a convex combination of atoms in the active set: $\boldsymbol{\alpha}_{(i)} = \sum_{\boldsymbol{v}_{(i)}\in\mathcal{S}_i} \beta_i(\boldsymbol{v}_{(i)})\, \boldsymbol{v}_{(i)}$, where $\beta_i(\boldsymbol{v}_{(i)}) > 0$ is the convex combination coefficient associated with atom $\boldsymbol{v}_{(i)}$. Then we can also express $\boldsymbol{\alpha}$ as an explicit convex combination of the (exponential size) active set $\mathcal{S}$ as follows: $\boldsymbol{\alpha} = \sum_{\boldsymbol{v}\in\mathcal{S}} \beta(\boldsymbol{v})\, \boldsymbol{v}$, where $\beta(\boldsymbol{v}) := \prod_{i=1}^n \beta_i(\boldsymbol{v}_{(i)})$.

We can now prove an analog of the expected block descent lemma (Lemma 4 for BCFW) in the case of BCPFW and BCAFW. For technical reasons, we need a slightly different block curvature constant $C_f^{\text{A}\,(i)}$ (cf. Eq. (26) in Lacoste-Julien & Jaggi (2015)).

Lemma 8 (Expected BCPFW descent lemma)**.** *Consider running the BCPFW algorithm on problem* (34)*. Let $\boldsymbol{\alpha}^{(k)}$ be the current iterate, and suppose that $\mathcal{S}_j$ is the current active set for each block $\boldsymbol{\alpha}^{(k)}_{(j)}$. Let $\mathcal{S}^{(k)} := \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$ be the current (implicit) active set for $\boldsymbol{\alpha}^{(k)}$. Suppose that* there is no drop set at $\boldsymbol{\alpha}^{(k)}$*, that is, that for each possible block i that could be picked at this stage, the PFW step with line-search on block i will not have its step size truncated (we say that the line-search* will succeed*). Then, conditioned on the current state, in expectation over the random choice of block i with uniform probability and for any $\gamma \in [0,1]$, it holds for the next iterate $\boldsymbol{\alpha}^{(k+1)}$ of BCPFW:*

$$\begin{aligned}&\mathbf{E}\big[f(\boldsymbol{\alpha}^{(k+1)}) \,|\, \boldsymbol{\alpha}^{(k)}, \mathcal{S}^{(k)}\big] \le \\ &\qquad f(\boldsymbol{\alpha}^{(k)}) - \tfrac{\gamma}{n} g^{\text{PFW}}(\boldsymbol{\alpha}^{(k)}\,;\,\mathcal{S}^{(k)}) + \tfrac{\gamma^2}{2n} C_f^{\text{A}\otimes},\end{aligned} \tag{37}$$

where $C_f^{\text{A}\otimes} := \sum_{i=1}^n C_f^{\text{A}\,(i)}$ is the total (away) curvature constant, and where $C_f^{\text{A}\,(i)}$ is defined as in Definition 2, but allowing the reference point $\boldsymbol{\alpha}_{(i)}$ in (13) *to be any point $\boldsymbol{v}_{(i)} \in \mathcal{M}^{(i)}$ instead, thus allowing a pairwise FW direction $\boldsymbol{s}_{[i]} - \boldsymbol{v}_{[i]}$ to be used in its definition.*

Moreover, (37) *also holds for BCAFW (again under the assumption of* no drop step*), but with an extra $1/2$ factor in front of $g^{\text{PFW}}(\boldsymbol{\alpha}^{(k)}\,;\,\mathcal{S}^{(k)})$ in the bound.*

Proof. Let block i be the chosen one that defined $\boldsymbol{\alpha}^{(k+1)}$ and let $\boldsymbol{\alpha}_\gamma := \boldsymbol{\alpha}^{(k)} + \gamma(\boldsymbol{s}_{[i]} - \boldsymbol{v}_{[i]})$, where $\mathbf{s}_{(i)} \in \mathcal{M}^{(i)}$ is the FW corner on block i with $\mathbf{s}_{[i]} \in \mathbb{R}^m$ its zero-padded version, and similarly $\boldsymbol{v}_{(i)} \in \mathcal{S}_i$ is the chosen away corner on block i. By assumption, we have that the line-search succeeds, i.e., the minimum of $\min_{\gamma\in[0,\gamma_{\max}]} f(\boldsymbol{\alpha}_\gamma)$ is achieved for $\gamma^* < \gamma_{\max}$, where $\gamma_{\max}$ is the maximum step size for this block for the PFW direction (this is because the optimal step size for the line-search cannot be truncated at $\gamma_{\max}$, as otherwise it would be a drop step). As f is a convex function, this means that $f(\boldsymbol{\alpha}^{(k+1)}) = \min_{\gamma\in[0,\gamma_{\max}]} f(\boldsymbol{\alpha}_\gamma) = \min_{\gamma\ge 0} f(\boldsymbol{\alpha}_\gamma)$ (removing inactive constraints does not change its minimum). By definition

[10] That is, $\boldsymbol{\alpha}_{(i)}$ is a convex combination of *all* the elements of $\mathcal{S}^{(i)}$ with non-zero coefficients.

of $C_f^{\mathrm{A}\,(i)}$, we thus have for any $\gamma \in [0,1]$:

$$\begin{aligned} f(\boldsymbol{\alpha}^{(k+1)}) &\le f(\boldsymbol{\alpha}_\gamma) \\ &= f(\boldsymbol{\alpha}^{(k)} + \gamma(\boldsymbol{s}_{[i]} - \boldsymbol{v}_{[i]})) \\ &\le f(\boldsymbol{\alpha}^{(k)}) + \gamma\langle\nabla_{(i)} f(\boldsymbol{\alpha}),\, \boldsymbol{s}_{(i)} - \boldsymbol{v}_{(i)}\rangle + \tfrac{\gamma^2}{2} C_f^{\mathrm{A}\,(i)} \\ &= f(\boldsymbol{\alpha}^{(k)}) - \gamma\, g_i^{\mathrm{PFW}}(\boldsymbol{\alpha}^{(k)}; \mathcal{S}_i) + \tfrac{\gamma^2}{2} C_f^{\mathrm{A}\,(i)}. \end{aligned} \quad (38)$$

Taking the expectation of the bound with respect to i, conditioned on $\boldsymbol{\alpha}^{(k)}$ and $\mathcal{S}^{(k)}$, yields (37) by using the block-decomposition relationship (36) in the definition of $g^{\mathrm{PFW}}(\boldsymbol{\alpha}^{(k)}\,;\,\mathcal{S}^{(k)})$. This completes the proof for BCPFW.

In the case of BCAFW, let $\boldsymbol{d}_i$ be the chosen direction for block i (either a FW direction or an away direction). Then since $\boldsymbol{d}_i$ is chosen to maximize the inner product with $-\nabla_{(i)} f(\boldsymbol{\alpha}^{(k)})$, we have $\langle -\nabla_{(i)} f(\boldsymbol{\alpha}^{(k)}), \boldsymbol{d}_i\rangle \ge \frac{1}{2} g_i^{\mathrm{PFW}}(\boldsymbol{\alpha}^{(k)}\,;\,\mathcal{S}_i)$ (with a similar argument as used to get Eq. (6) in Lacoste-Julien & Jaggi (2015) for AFW). We then follow the same argument to derive (38), but using $\boldsymbol{d}_i$ instead of $(\boldsymbol{s}_{(i)} - \boldsymbol{v}_{(i)})$, which gives an extra $1/2$ factor as $\langle -\nabla_{(i)} f(\boldsymbol{\alpha}^{(k)}), \boldsymbol{d}_i\rangle$ is potentially only half of $g_i^{\mathrm{PFW}}(\boldsymbol{\alpha}^{(k)}\,;\,\mathcal{S}_i)$. Taking again the expectation of (38) completes the proof. □

Remark 9. The important condition that there is *no drop step* at $\boldsymbol{\alpha}^{(k)}$ in the BCPFW descent lemma 8 is to allow the bound (38) to hold for *any* $\gamma \in [0,1]$. Otherwise, let I be the (non-empty) set of blocks for which there would be a drop step at $\boldsymbol{\alpha}^{(k)}$ and let $\gamma_I := \min_{i\in I} \gamma_{\max}^{(i)}$, where $\gamma_{\max}^{(i)}$ is the maximum step size for block i. Then in this case we could only show the bound (38) for $\gamma \le \gamma_I$. But γ_I could be arbitrarily small,[11] and so no significant progress is guaranteed in expectation in this case.

We also note that $C_f^{\mathrm{A}\,(i)}$ is used instead of $C_f^{(i)}$ in the lemma because $C_f^{(i)}$ can only be used with a feasible step from $\boldsymbol{\alpha}^{(k)}$, and thus again, the bound would only be valid for $\gamma \le \gamma_{\max}$ (as bigger step sizes can take you outside of $\mathcal{M}^{(i)}$). If the gradient of f is Lipschitz continuous, one can bound $C_f^{\mathrm{A}\,(i)} \le \tilde{L}_i\big(\operatorname{diam}_{\|\cdot\|_i} \mathcal{M}^{(i)}\big)^2$, which is almost the same bound as for $C_f^{(i)}$ explained in footnote 6, but with $\tilde{L}_i$ being the Lipschitz constant of $\nabla_{(i)} f$ for variations in the slightly extended domain $\mathcal{M}^{(i)} + (\mathcal{M}^{(i)} - \mathcal{M}^{(i)})$ (with set addition in the Minkowski sense).

Theorem 4 (Geometric convergence of BCPFW)**.** *Consider running BCPFW (or BCAFW) on problem* (34) *where q is a strongly convex function and $\mathcal{M}$ is a block-separable polytope. Let $h_k := f(\boldsymbol{\alpha}^{(k)}) - f(\boldsymbol{\alpha}^*)$ be the suboptimality of the iterate k, where $\boldsymbol{\alpha}^*$ is any optimal solution to* (34)*. Conditioned on any iterate $\boldsymbol{\alpha}^{(k)}$ with active set $\mathcal{S}^{(k)}$ such that* no block could give a drop set *(as defined in the conditions for Lemma 8), then the expected new suboptimality decreases geometrically, that is:*

$$\mathbf{E}\big[h_{k+1}\,|\,\boldsymbol{\alpha}^{(k)}, \mathcal{S}^{(k)}\big] \le (1-\rho)h_k, \quad (39)$$

with rate:

$$\rho := \tfrac{1}{2n}\min\{1, 2\tfrac{\tilde{\mu}_f}{C_f^{\mathrm{A}\otimes}}\} \text{ for the BCPFW algorithm}, \quad (40)$$

$$\rho := \tfrac{1}{4n}\min\{1, \tfrac{\tilde{\mu}_f}{C_f^{\mathrm{A}\otimes}}\} \text{ for the BCAFW algorithm}, \quad (41)$$

where $C_f^{\mathrm{A}\otimes} := \sum_{i=1}^n C_f^{\mathrm{A}\,(i)}$ is the total (away) curvature constant for problem (34) *as defined in Lemma 8, and $\tilde{\mu}_f$ is the generalized strong convexity constant for problem* (34) *as defined in Eq. (39) of Lacoste-Julien & Jaggi (2015) ($\tilde{\mu}_f$ is strictly greater than zero when q is strongly convex and $\mathcal{M}$ is a polytope).*

Proof. We first do the argument for BCPFW. Let $g_k := g^{\mathrm{PFW}}(\boldsymbol{\alpha}^{(k)}\,;\,\mathcal{S}^{(k)})$, and notice that $g_k \ge h_k$ always. Because we assume that there is no drop step at $\boldsymbol{\alpha}^{(k)}$, we can use the expected BCPFW descent lemma 8. By subtracting $f(\boldsymbol{\alpha}^*)$ on both side of the descent inequality (37), we get (for any $\gamma \in [0,1]$):

$$\mathbf{E}\big[h_{k+1}\,|\,\boldsymbol{\alpha}^{(k)}, \mathcal{S}^{(k)}\big] \le h_k - \tfrac{\gamma}{n} g_k + \tfrac{\gamma^2}{2n} C_f^{\mathrm{A}\otimes}. \quad (42)$$

We can minimize the RHS of (42) with $\gamma^* = \frac{g_k}{C_f^{\mathrm{A}\otimes}}$. If $g_k > C_f^{\mathrm{A}\otimes}$ (i.e. $\gamma^* > 1$), then use $\gamma = 1$ in (42) to get:

$$\mathbf{E}\big[h_{k+1}\,|\,\boldsymbol{\alpha}^{(k)}, \mathcal{S}^{(k)}\big] \le h_k - \tfrac{1}{2n} g_k \le (1 - \tfrac{1}{2n})h_k. \quad (43)$$

This gives a geometric rate of $\rho = \frac{1}{2n}$. So now suppose that $g_k \le C_f^{\mathrm{A}\otimes}$ (so that $\gamma^* \le 1$); putting $\gamma = \gamma^*$ in (42), we get:

$$\mathbf{E}\big[h_{k+1}\,|\,\boldsymbol{\alpha}^{(k)}, \mathcal{S}^{(k)}\big] \le h_k - \tfrac{1}{2nC_f^{\mathrm{A}\otimes}}\, g_k{}^2. \quad (44)$$

We now use the key relationship between the suboptimality h_k and the PFW gap g_k derived in inequality (43) of Lacoste-Julien & Jaggi (2015) (which is true for any function f by definition of $\tilde{\mu}_f$ if we allow it to be zero):

$$h_k \le \tfrac{g_k{}^2}{2\tilde{\mu}_f}. \quad (45)$$

Substituting (45) into (44), we get:

$$\mathbf{E}\big[h_{k+1}\,|\,\boldsymbol{\alpha}^{(k)}, \mathcal{S}^{(k)}\big] \le (1 - \tfrac{\tilde{\mu}_f}{nC_f^{\mathrm{A}\otimes}})h_k, \quad (46)$$

which gives the $\rho = \tilde{\mu}_f / nC_f^{\mathrm{A}\otimes}$ rate. Taking the worst rate of (43) and (46) gives the rate (40), completing the proof for BCPFW.

[11] Small maximum step sizes happen when the current coordinate value for an away corner is small (perhaps because a small step size was used by the line-search when they were added as a FW corner previously).

In the case of BCAFW, Lemma 8 yields the inequality (42) but with an extra $1/2$ factor in front of g_k. Re-using the same argument as above, we get a rate of $\rho = 1/4n$ when $\gamma^* > 1$, and $\rho = \tilde{\mu}_f / 4nC_f^{A\otimes}$ when $\gamma^* \leq 1$, showing (41) as required.

Finally, the fact that $\tilde{\mu}_f > 0$ when q is μ-strongly convex and $\mathcal{M}$ is a polytope comes from the lower bound given in Theorem 10 of Lacoste-Julien & Jaggi (2015) in terms of the *pyramidal width* of $\mathcal{M}$ (a strictly positive geometric quantity for polytopes), and the *generalized strong convexity* of f as defined in Lemma 9 of Lacoste-Julien & Jaggi (2015). The generalized strong convexity of f is simply μ if f is μ-strongly convex. In the more general case of problem (34) where only q is μ-strongly convex, the generalized strong convexity depends both on μ and the *Hoffman constant* (Hoffman, 1952) associated with the linear system of problem (34). See Lacoste-Julien & Jaggi (2015) for more details, as well as Lemma 2.2 of Beck & Shtern (2015). □

Interpretation. Theorem 4 only guarantees progress of BCPFW or BCAFW when there would not be any drop step for *any* block i for the current iterate. For the batch AFW algorithm, one can easily lower bound the number of times that these "good steps" can happen as a drop step reduces the size of the active set and thus cannot happen more than half of the time. On the other hand, in the block coordinate setting, we can be unlucky and always have one block that could give a drop step (while we pick other blocks during the algorithm, this bad block affects the expectation). This means that without a refined analysis of the drop step possibility, we cannot guarantee any progress in the worst case for BCPFW or BCAFW. As a safeguard, one can modify BCPFW or BCAFW so that it also has the option to do a standard BCFW step on a block if it yields better progress on f – this way, the algorithm inherits at least the (sublinear) convergence guarantees of BCFW.

Empirical linear convergence. In our experiments, we note that BCPFW always converged empirically, and had an empirical linear convergence rate for the SSVM objective when λ was big enough ($q(\cdot) = \frac{\lambda}{2}\|\cdot\|^2$ for the SSVM objective (3)). See Figure 5 for OCR-large (c) for example. We also tried the modified BCPFW algorithm where a choice is made between a FW step, a pairwise FW step or an away step on a block by picking the one which gives the biggest progress. We did not notice any significant speedup for this modified method.

On the dimension of SSVM. Finally, we note that the rate constant ρ in Theorem 4 has an implicit dependence on the dimensionality (in particular, through the pyramidal width of $\mathcal{M}$). Lacoste-Julien & Jaggi (2015) showed that the largest possible pyramidal width of a polytope in dimension m (for a fixed diameter) is achieved by the probability simplex and is $\Theta(1/\sqrt{m})$. For the SSVM in the general form (3), the dimensionality of $\mathcal{M}^{(i)}$ is the number of possible structured outputs for input i, which is typically an exponentially large number, and thus the pyramidal width lower bound would be useless in this case. Fortunately, the matrix A (feature map) and vector $\boldsymbol{b}$ (loss function) are highly structured, and thus many $\boldsymbol{\alpha}$'s are mapped to the same objective value. For a feature mapping $\boldsymbol{\psi}_i(\boldsymbol{y})$ representing the sufficient statistics for an energy function associated with a graphical model (as for a conditional random field (Lafferty et al., 2001)), then the SSVM objective is implicitly optimizing over the *marginal polytope* for the graphical model (Wainwright & Jordan, 2008). More specifically, let A_i be the $d \times m_i$ submatrix of A associated with example i. Then we can write $A_i = B_i M_i$ where M_i is a $p \times m_i$ *marginalization* matrix, that is, $\boldsymbol{\mu} = M_i \boldsymbol{\alpha}$ is an element of the marginal polytope for the graphical model, where p is the dimensionality of the marginal polytope – which is a polynomial number in the size of the graph, rather than exponential. By the affine invariance property of the FW-type algorithms, we can thus instead use the pyramidal width of the marginal polytope for the convergence analysis (and similarly for the Hoffman constant). Lacoste-Julien & Jaggi (2015) conjectured that the pyramidal width of a marginal polytope in dimension p was also $\Theta(1/\sqrt{p})$, thus giving a more reasonable bound for the convergence rate of BCPFW for SSVM.

H. BCFW for SSVM with box constraints

H.1. Problem with box constraints

Problem (1) can be equivalently rewritten as a quadratic program (QP) with an exponential number of constraints:

$$\begin{aligned} \min_{\boldsymbol{w},\boldsymbol{\xi}} \quad & \tfrac{\lambda}{2}\|\boldsymbol{w}\|^2 + \tfrac{1}{n}\sum_{i=1}^{n}\xi_i \\ \text{s.t.} \quad & \langle \boldsymbol{w}, \boldsymbol{\psi}_i(\boldsymbol{y})\rangle \geq L(\boldsymbol{y}_i,\boldsymbol{y}) - \xi_i \quad \forall i, \forall \boldsymbol{y} \in \mathcal{Y}_i \end{aligned} \tag{47}$$

where the slack variable ξ_i measures the surrogate loss for the i-th datapoint. Problem (47) is often referred to as the n-slack structured SVM with margin-rescaling (Joachims et al., 2009, Optimization Problem 2).

In this section, we consider the problem (47) with additional box constraints on the parameter vector $\boldsymbol{w}$:

$$\begin{aligned} \min_{\boldsymbol{w},\boldsymbol{\xi}} \quad & \tfrac{\lambda}{2}\|\boldsymbol{w}\|^2 + \tfrac{1}{n}\sum_{i=1}^{n}\xi_i \\ \text{s.t.} \quad & \langle \boldsymbol{w}, \boldsymbol{\psi}_i(\boldsymbol{y})\rangle \geq L(\boldsymbol{y}_i,\boldsymbol{y}) - \xi_i \quad \forall i, \forall \boldsymbol{y} \in \mathcal{Y}_i, \\ & \boldsymbol{l} \preccurlyeq \boldsymbol{w} \preccurlyeq \boldsymbol{u}, \end{aligned} \tag{48}$$

where $\boldsymbol{l} \in \mathbb{R}^d$ and $\boldsymbol{u} \in \mathbb{R}^d$ denote the lower and upper bounds, respectively, and the symbol "$\preccurlyeq$" is the elementwise "less or equal to" sign. In the following, we assume that the box constraints are feasible, i.e., $\boldsymbol{l} \preccurlyeq \boldsymbol{u}$. Note that the following discussion can be directly extended to the case where only some dimension of the weight vector have to respect the box constraints. The Lagrangian of problem (48) can be written as

$$L(\boldsymbol{w},\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\beta}_l,\boldsymbol{\beta}_u) = \tfrac{\lambda}{2}\langle \boldsymbol{w},\boldsymbol{w}\rangle + \tfrac{1}{n}\sum_{i=1}^{n}\xi_i + \sum_{i\in[n],\,\boldsymbol{y}\in\mathcal{Y}_i} \tfrac{1}{n}\alpha_i(\boldsymbol{y})\left(-\xi_i + \langle \boldsymbol{w}, -\boldsymbol{\psi}_i(\boldsymbol{y})\rangle + L_i(\boldsymbol{y})\right) + \lambda\langle \boldsymbol{\beta}_u, \boldsymbol{w}-\boldsymbol{u}\rangle + \lambda\langle \boldsymbol{\beta}_l, -\boldsymbol{w}+\boldsymbol{l}\rangle \quad (49)$$

where $\boldsymbol{\beta}_l \in \mathbb{R}^d$ and $\boldsymbol{\beta}_u \in \mathbb{R}^d$ are the dual variables associated with the lower and upper bound constraints, respectively. From the KKT conditions, we obtain

$$\boldsymbol{w} = A\boldsymbol{\alpha} - (\boldsymbol{\beta}_u - \boldsymbol{\beta}_l), \quad (50)$$

$$\sum_{\boldsymbol{y}\in\mathcal{Y}_i} \alpha_i(\boldsymbol{y}) = 1 \quad \forall i \in [n]. \quad (51)$$

Finally, the dual of problem (48) (here written in a minimization form) can be written as follows:

$$\min_{\substack{\boldsymbol{\alpha}\in\mathbb{R}^m\\ \boldsymbol{\alpha}\succcurlyeq 0}} f(\boldsymbol{\alpha},\boldsymbol{\beta}_l,\boldsymbol{\beta}_u) := \tfrac{\lambda}{2}\left\|A\boldsymbol{\alpha} - (\boldsymbol{\beta}_u - \boldsymbol{\beta}_l)\right\|^2 - \boldsymbol{b}^\mathsf{T}\boldsymbol{\alpha} + \lambda(\boldsymbol{\beta}_u^\mathsf{T}\boldsymbol{u} - \boldsymbol{\beta}_l^\mathsf{T}\boldsymbol{l})$$
$$\text{s.t.} \sum_{\boldsymbol{y}\in\mathcal{Y}} \alpha_i(\boldsymbol{y}) = 1 \quad \forall i\in[n],$$
$$\text{and } \boldsymbol{\beta}_u \succcurlyeq 0, \boldsymbol{\beta}_l \succcurlyeq 0. \quad (52)$$

A modified block optimization method. Ideally, we should optimize $f(\boldsymbol{\alpha},\boldsymbol{\beta}_l,\boldsymbol{\beta}_u)$ jointly w.r.t. all the dual variables. This task is not directly suitable for the Frank-Wolfe approach as the domain for $\boldsymbol{\beta}_l$ and $\boldsymbol{\beta}_u$ is unbounded. However, joint optimization w.r.t. $\boldsymbol{\beta}_l$ and $\boldsymbol{\beta}_u$ with $\boldsymbol{\alpha}$ kept fixed can be done in closed form. After that, optimization w.r.t. $\boldsymbol{\alpha}$ can be performed using the Frank-Wolfe blockwise approach. Therefore, we resort to optimizing in a blockwise fashion: we iterate either a batch FW or a BCFW step on $\boldsymbol{\alpha}$ with an exact block-update on $(\boldsymbol{\beta}_u, \boldsymbol{\beta}_l)$. As we will see below, this principled approach is similar to a commonly used heuristic of truncating the value of $\boldsymbol{w}$ to make it feasible during an algorithm which works on the dual. In fact, our approach will be equivalent to run FW or BCFW with a truncation making $\boldsymbol{w}(\boldsymbol{\alpha})$ feasible after each FW step, but with a *change in the optimal step-size computation* (line 8 in Algorithm 8 for FW; line 7 in Algorithm 9 for BCFW) due to the different nature of the optimization problem.

H.2. Optimizing w.r.t $\boldsymbol{\beta}_u$ and $\boldsymbol{\beta}_l$ while fixing $\boldsymbol{\alpha}$

The optimization w.r.t. $\boldsymbol{\beta}_u$ with $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}_l$ fixed can be easily solved in closed form via a simple thresholding operation:

$$\boldsymbol{\beta}_u^* = [A\boldsymbol{\alpha} + \boldsymbol{\beta}_l - \boldsymbol{u}]_+ \,. \quad (53)$$

The optimization w.r.t. $\boldsymbol{\beta}_l$ with $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}_u$ fixed is analogous:

$$\boldsymbol{\beta}_l^* = [-A\boldsymbol{\alpha} + \boldsymbol{\beta}_u + \boldsymbol{l}]_+ \,. \quad (54)$$

Algorithm 8 Batch Frank-Wolfe algorithm for structured SVM with box constraints

```
1: Let $\boldsymbol{v}^{(0)} := \mathbf{0}$; $\ell^{(0)} := 0$
2: $\boldsymbol{w}^{(0)} := [\boldsymbol{v}^{(0)}]_{\boldsymbol{l}}^{\boldsymbol{u}}$    // truncation to the feasible set
3: for $k := 0, \dots, \infty$ do
4:     for $i := 1, \dots, n$ do
5:         Solve $\boldsymbol{y}_i^* := \operatorname{argmax}_{\boldsymbol{y}\in\mathcal{Y}_i} H_i(\boldsymbol{y}; \boldsymbol{w}^{(k)})$
6:     end for
7:     Let $\boldsymbol{v}_{\boldsymbol{s}} := \sum_{i=1}^{n} \frac{1}{\lambda n}\boldsymbol{\psi}_i(\boldsymbol{y}_i^*)$ and $\ell_{\boldsymbol{s}} := \frac{1}{n}\sum_{i=1}^{n} L_i(\boldsymbol{y}_i^*)$
8:     Let $\gamma := \frac{\lambda(\boldsymbol{v}^{(k)}-\boldsymbol{v}_{\boldsymbol{s}})^\mathsf{T}\boldsymbol{w}^{(k)} - \ell^{(k)} + \ell_{\boldsymbol{s}}}{\lambda\|\boldsymbol{v}^{(k)}-\boldsymbol{v}_{\boldsymbol{s}}\|^2}$ and clip to $[0,1]$
9:     Update $\boldsymbol{v}^{(k+1)} := (1-\gamma)\boldsymbol{v}^{(k)} + \gamma\,\boldsymbol{v}_{\boldsymbol{s}}$
10:        and $\ell^{(k+1)} := (1-\gamma)\ell^{(k)} + \gamma\,\ell_{\boldsymbol{s}}$
11:        and $\boldsymbol{w}^{(k+1)} := [\boldsymbol{v}^{(k+1)}]_{\boldsymbol{l}}^{\boldsymbol{u}}$
12: end for
```

Algorithm 9 Block-coordinate Frank-Wolfe algorithm for structured SVM with box constraints

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

Denote the p-th variable of $\boldsymbol{\beta}_u$ and $\boldsymbol{\beta}_l$ with $\boldsymbol{\beta}_u(p)$ and $\boldsymbol{\beta}_l(p)$, respectively. For any index p, both $\boldsymbol{\beta}_u(p)$ and $\boldsymbol{\beta}_l(p)$ cannot be nonzero simultaneously, because if one of the constraints is violated (either the upper or the lower bound), then the other constraint must be satisfied. Hence, $\boldsymbol{\beta}_u(p) \neq 0$ implies $\boldsymbol{\beta}_l(p) = 0$ and vice versa. Therefore, the final update equations can equivalently be written as

$$\boldsymbol{\beta}_u^*(\boldsymbol{\alpha}) = [A\boldsymbol{\alpha} - \boldsymbol{u}]_+ \,, \quad (55)$$

$$\boldsymbol{\beta}_l^*(\boldsymbol{\alpha}) = [-A\boldsymbol{\alpha} + \boldsymbol{l}]_+ \,. \quad (56)$$

Introducing $\boldsymbol{v}(\boldsymbol{\alpha}) := A\boldsymbol{\alpha}$, we get $\boldsymbol{\beta}_u^* = [\boldsymbol{v} - \boldsymbol{u}]_+$ and $\boldsymbol{\beta}_l^* = [-\boldsymbol{v} + \boldsymbol{l}]_+$. Hence, the operation $\boldsymbol{w} = \boldsymbol{v} - (\boldsymbol{\beta}_u^* - \boldsymbol{\beta}_l^*)$ is simply the projection (truncation) of $\boldsymbol{v}$ on the feasible set defined by the upper and lower bounds. In the final algorithm, we maintain $\boldsymbol{v}(\boldsymbol{\alpha})$ and directly update the primal variables $\boldsymbol{w}$ without updating $\boldsymbol{\beta}_u$ and $\boldsymbol{\beta}_l$.

H.3. Batch setting: optimizing w.r.t α with β_u and β_l fixed

When the variables β_u and β_l are fixed, the convex problem (52) has a compact domain and so we can use the Frank-Wolfe algorithm on it. In the following, we highlight the differences with the setting without box constraints. We denote by $w(\alpha)$ the truncation of $v(\alpha)$ on the box constraints, i.e.,

$$w(\alpha) := v(\alpha) - (\beta_u^*(\alpha) - \beta_l^*(\alpha)) . \tag{57}$$

The derivations below assume that β_u and β_l are fixed to their optimal values $\beta_u^*(\alpha)$ and $\beta_l^*(\alpha)$ for a specific α.

Linear subproblem. The Frank-Wolfe linear subproblem can be written as

$$s = \operatorname*{argmin}_{s' \in \mathcal{M}} \langle s', \nabla_\alpha f(\alpha, \beta_l, \beta_u)\rangle \tag{58}$$

where $\nabla_\alpha f(\alpha, \beta_l, \beta_u)$ can be easily computed:

$$\begin{aligned}\nabla_\alpha f(\alpha, \beta_l, \beta_u) &= \lambda A^\top (A\alpha - (\beta_u - \beta_l)) - b \\ &= \lambda A^\top w - b.\end{aligned} \tag{59}$$

Analogously to the problem without box constraints, the linear subproblem used by the Frank-Wolfe algorithm is equivalent to the loss-augmented decoding subproblem (2). The update of α can be made using the corner s. In what follows, we show that this update can be performed without explicitly keeping the dual variables at the cost of storing the extra vector v.

The duality gap. The Frank-Wolfe gap for problem (52) can be written as

$$\begin{aligned} g(\alpha) :=& \max_{s' \in \mathcal{M}} \langle \alpha - s', \nabla_\alpha f(\alpha, \beta_l, \beta_u)\rangle \\ =& (\alpha - s)^\top (\lambda A^\top (A\alpha - (\beta_u - \beta_l)) - b) \\ =& \lambda (v - v_s)^\top w - b^\top \alpha + b^\top s \end{aligned}$$

where $v_s := As$. Below, we prove that the Frank-Wolfe duality gap $g(\alpha)$ for the problem (52) when β_u and β_l are fixed at their current optimal value for the current α equals to a Lagrange duality gap, analogously to the case without box constraints (Lacoste-Julien et al., 2013, Appendix B.2).[12]

Proof. Consider the difference between the primal objective of (48) at $w := A\alpha - (\beta_u - \beta_l)$ with the optimal slack variables ξ and the dual objective of (52) at α (in the maximization form). We get

$$\begin{aligned} g_{\text{Lag.}}(w, \alpha) &= \tfrac{\lambda}{2} w^\top w + \tfrac{1}{n} \sum_{i=1}^n \tilde{H}_i(w) \\ &\quad - \left(b^\top \alpha - \tfrac{\lambda}{2} w^\top w - \lambda(\beta_u^\top u - \beta_l^\top l)\right) \\ &= \lambda w^\top w - b^\top \alpha + \tfrac{1}{n} \sum_{i=1}^n \max_{y \in \mathcal{Y}_i} H_i(y; w) \\ &\quad + \lambda(\beta_u^\top u - \beta_l^\top l) . \end{aligned}$$

Recalling

$$\begin{aligned} \tfrac{1}{n} \sum_{i=1}^n \max_{y \in \mathcal{Y}_i} H_i(y; w) &= \max_{s' \in \mathcal{M}} -s'^\top \nabla_\alpha f(\alpha, \beta_l, \beta_u) \\ &= -s^\top \nabla_\alpha f(\alpha, \beta_l, \beta_u), \end{aligned}$$

we can write

$$\begin{aligned} g_{\text{Lag.}}(w, \alpha) &= \lambda w^\top (A\alpha - (\beta_u - \beta_l)) - b^\top \alpha \\ &\quad - s^\top \nabla_\alpha f(\alpha, \beta_l, \beta_u) + \lambda(\beta_u^\top u - \beta_l^\top l) \\ &= (\lambda w^\top A - b^\top)\alpha - s^\top \nabla_\alpha f(\alpha, \beta_l, \beta_u) \\ &\quad - \lambda(w^\top \beta_u - w^\top \beta_l) + \lambda(\beta_u^\top u - \beta_l^\top l) \\ &= (\alpha - s)^\top \nabla_\alpha f(\alpha, \beta_l, \beta_u) \\ &\quad + \lambda(\beta_u^\top (u - w) - \beta_l^\top (l - w)). \end{aligned}$$

As we assumed that $\beta_u = \beta_u^*(\alpha)$ and $\beta_l = \beta_l^*(\alpha)$, we have $\beta_u^{*\top}(u - w) = 0$ and $\beta_l^{*\top}(l - w) = 0$, and thus

$$g_{\text{Lag}}(w, \alpha, \beta_u^*, \beta_l^*) = g(\alpha).$$

$\square$

Line-Search. Line search can be performed efficiently using $\gamma_{\text{opt}s} := \frac{\langle \alpha - s, \nabla f(\alpha)\rangle}{\lambda \|A(\alpha - s)\|^2} = \frac{g(\alpha)}{\lambda \|v - v_s\|^2}$.

Algorithm. Algorithm 8 contains the batch Frank-Wolfe algorithm with box constraints. The main idea consists in maintaining the vector $v := A\alpha$ in order to perform all the updates of α using only the primal variables. Optimization w.r.t. β_l and β_u corresponds to the truncation of v. Given these variables, the gap and the optimal step size are easy to compute.

H.4. The block-coordinate setting

Algorithm 9 describes the block-coordinate version of the Frank-Wolfe algorithm with box constraints. The algorithm is obtained from the batch version (Algorithm 8) in exactly the same way as BCFW (Algorithm 1) is obtained from the batch Frank-Wolfe method (Lacoste-Julien et al., 2013, Algorithm 2).

[12]We stress that this relationship is only valid for the pair $w = w(\alpha)$ in the primal, and $\beta_u = \beta_u^*(\alpha), \beta_l = \beta_l^*(\alpha)$ in the dual.

Table 1. Statistics of the datasets used in the experimental evaluation. For the oracle time, we report the sum of the average running times for both the max oracle and the joint feature map computation (the starred numbers indicate that the input features were stored on disk instead of RAM, thus slowing down the computation).

Dataset	Task	Sructure (Oracle)	Citation	Version	n	d	Sparsity	Box constraints	Oracle time (in s)
OCR	character recognition	chain (Viterbi)	(Taskar et al., 2003)	small	626	4,082	No	No	5×10^{-4}
				large	6,251	4,082	No	No	5×10^{-4}
CoNLL	text chunking	chain (Viterbi)	(Tjong Kim Sang & Buchholz, 2000)		8,936	1,643,026	Yes	No	2×10^{-2}
HorseSeg	binary segmentation	grid (graph cut)	(Kolesnikov et al., 2014)	small	147	1,969	No	Yes	1×10^{-3}
				medium	6,121	1,969	No	Yes	1×10^{-3}
				large	25,438	1,969	No	Yes	2×10^{-2} (*)
LSP	pose estimation	tree (max sum)	(Johnson & Everingham, 2010)	small	100	2,676	No	Yes	2 (*)

I. Dataset description

In our experiments, we use four structured prediction datasets: OCR (Taskar et al., 2003) for character recognition; CoNLL (Tjong Kim Sang & Buchholz, 2000) for text chunking; HorseSeg (Kolesnikov et al., 2014) for binary image segmentation; LSP (Johnson & Everingham, 2010) for pose estimation. In this section, we provide the description of the datasets and the corresponding models. Table 1 summarizes quantitative statistics for all the datasets. For the OCR and CoNLL datasets, the features and models described below are exactly the same as used by Lacoste-Julien et al. (2013); we give a detailed description for reference. For HorseSeg and LSP, we had to build the models ourselves from previous work referenced in the relevant section.

I.1. OCR

The Optical Character Recognition (OCR) dataset collected by Taskar et al. (2003) poses the task of recognizing English words from sequences of handwritten symbols represented by binary images. The average length of sequences equals 7.6 symbols. For a sequence of length T, the input feature representation $\boldsymbol{x}$ consists of T binary images of size 16×8. The output object $\boldsymbol{y}$ is a sequence of length T with each symbol taking 26 possible values.

The OCR dataset contains $6,877$ words. In the small version, 626 words are used for training and the rest for testing. In the large version, $6,251$ words are used for training and the rest for testing.

The prediction model is a chain. The feature map $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$ contains features of three types: emission, transition and bias. The $16 \times 8 \times 26$ emission features count the number of times along the chain a specific position of the 16×8 binary image equals 1 when associated with a specific output symbol. The 26×26 transition features count the number of times one symbol follows another. The 26×3 bias features represent three biases for each element of the output alphabet: one model bias, and a bias for when the letter appears at the beginning or at the end of the sequence.

As the structured error between output vectors $L(\boldsymbol{y}_i, \boldsymbol{y})$, the OCR dataset uses the Hamming distance normalized by the length of the sequences. The loss-augmented structured score $H_i(\boldsymbol{y}; \boldsymbol{w})$ is a function with unary and pairwise potentials defined on a chain and is exactly maximized with the dynamic programming algorithm of Viterbi (1967).

I.2. CoNLL

The CoNLL dataset released by Tjong Kim Sang & Buchholz (2000) poses the task of text chunking. Text chunking, also known as shallow parsing (Sha & Pereira, 2003), consists in dividing the input text into syntactically related non-overlapping groups of words, called phrase or chunks. The task of text chunking can be cast as a sequence labeling where a sequence of labels $\boldsymbol{y}$ is predicted from an input sequence of tokens $\boldsymbol{x}$. For a given token x_t (a word with its corresponding part-of-speech tag), the associated label y_t gives the type of phrase the token belongs to, i.e., says whether or not it corresponds to the beginning of a chunk, or encodes the fact that the token does not belong to a chunk.

The CoNLL dataset contains $8,936$ training English sentences extracted from the Wall Street Journal part of the Penn Treebank II (Marcus et al., 1993). Each output label y_t can take up to 22 different values.

We use the feature map $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$ proposed by Sha & Pereira (2003). First, for each position t of the input sequence $\boldsymbol{x}$, we construct a unary feature representation, containing the local information. We start with extracting several attributes representing the words and the part-of-speech tags at the positions neighboring to t.[13] Each attribute is encoded with an indicator vector of length equal to either the dictionary size or the number of part-of-speech tags. We concatenate these vectors to get a unary feature representation, which is a sparse binary vector of dimensional-

[13]We extract the attributes with the CRFsuite library (Okazaki, 2007) and refer to its documentation for the exact list of attributes: http://www.chokkan.org/software/crfsuite/tutorial.html.

ity $74,658$. Note that these representation can be precomputed outside of the training process.

Given a labeling $\boldsymbol{y}$ and the unary representations, the feature map $\boldsymbol{\phi}$ is constructed by concatenating features of three types (as in the chain model for OCR): emission, transition and bias. The $74,658 \times 22$ emission features count the number of times each coordinate of the unary representation of token x_t is nonzero and the corresponding output variable y_t is assigned a particular value. The transition map of size 22×22 encodes the number of times one label follows another in the output $\boldsymbol{y}$. The 22×3 bias features encode biases for all the possible values of the output variables and, specifically, biases for the first and the last variables.

As in the OCR task, the structured error $L(\boldsymbol{y}_i, \boldsymbol{y})$ is the normalized Hamming distance and thus the max-oracle can be efficiently implemented using the dynamic programming algorithm of Viterbi (1967).

I.3. HorseSeg

The HorseSeg dataset[14] released by Kolesnikov et al. (2014) poses the task of object/background segmentation of images containing horses, i.e., assigning a label "horse" or "background" to each pixel of the image. HorseSeg contains $25,438$ training images, 147 of which are manually annotated, $5,974$ annotations are constructed from object bounding boxes by the automatic method of Guillaumin et al. (2014), while the remaining $19,317$ annotations were constructed by the same method but without any human supervision. The test set of HorseSeg consists of 241 images with manual annotations. In our experiments, we use training sets of three different sizes: 147 images for HorseSeg-small, $6,121$ images for HorseSeg-medium and $25,438$ for HorseSeg-large.

In addition to images and their pixel-level annotations, Kolesnikov et al. (2014) released[15] oversegmentations (superpixels) of the images precomputed with the SLIC algorithm (Achanta et al., 2012) and the unary features of each superpixel computed similarly to the work of Lempitsky et al. (2011). On average, each image contains 147 superpixels. The $1,969$ unary features include 1 constant feature, 512-bin histograms of densely sampled visual SIFT words (Lowe, 2004), 128-bin histograms of RGB colors, 16-bin histograms of locations (each pixel of a region of interest is matched to a cell of the 4×4 uniform grid). The three aforementioned histograms are computed on the superpixels themselves, on the superpixels together with their neighboring superpixels, and on the second-order neighborhoods.

For each pair of adjacent superpixels, we construct 100 pairwise features: a constant feature; and quantities $\exp(-\eta d_{pq})$ where d_{pq} is a χ^2-distance between 9 pairs of the corresponding histograms of each type for neighbors p and q, and η is a parameter taking 11 values from the set $2^{-5}, 2^{-4}, \dots, 2^{5}$.

The structured feature map is defined in such a way that the corresponding structured score function contains unary and pairwise Potts potentials

$$\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}) \rangle = \sum_{p \in \mathcal{V}_i} \langle \boldsymbol{w}_U, \boldsymbol{x}^U_{i,p} \rangle ([y_p = 1] - [y_p = 0]) + \sum_{\{p,q\} \in \mathcal{E}_i} \langle \boldsymbol{w}_P, \boldsymbol{x}^P_{i,pq} \rangle [y_p \neq y_p]$$

where the vector $\boldsymbol{x}_i = ((\boldsymbol{x}^U_{i,p})_{p \in \mathcal{V}_i}, (\boldsymbol{x}^P_{i,pq})_{\{p,q\} \in \mathcal{E}_i})$ denotes all the features of image i, the vector $\boldsymbol{y} = (y_p)_{p \in \mathcal{V}_i} \in \{0,1\}^{\mathcal{V}_i}$ is a feasible labeling, the set $\mathcal{V}_i$ is the set of the superpixels of the image i, and the set $\mathcal{E}_i$ represents the adjacency graph.

The structured error is measured with a Hamming loss with class-normalized penalties

$$L(\boldsymbol{y}_i, \boldsymbol{y}) = \sum_{p \in \mathcal{V}_i} \omega_{y_{i,p}} [y_{i,p} \neq y_p]$$

where $\boldsymbol{y}_i = (y_{i,p})_{p \in \mathcal{V}_i}$ is the labeling of superpixels closest to the ground-truth annotation and the weights ω_0 and ω_1 are proportional to the ground-truth area of each class.

The loss-augmented score function

$$L(\boldsymbol{y}_i, \boldsymbol{y}) - \langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}) \rangle$$

is a discrete function defined w.r.t. a cyclic graph and can be maximized in polynomial time when it is supermodular. By construction, all our pairwise features are nonnegative, so we can ensure supermodularity by adding positivity constraints on the weights corresponding to the pairwise features $\boldsymbol{w}_P \succcurlyeq 0$. The version of BCFW with positivity constraints is described in Appendix H.

The discrete optimization problem arising in the max-oracle is solved by the min-cut/max-flow algorithm of Boykov & Kolmogorov (2004). The running time or the max-flow is small compared to the operations with the features required to compute the potentials.

I.4. LSP

The Leeds Sports Pose (LSP) dataset introduced by (Johnson & Everingham, 2010) poses the tasks of full body pose estimation from still images containing sports activities. Based on the input image with a centered prominent person, the task is to predict the locations of 14 body-parts (joints), e.g., "left knee" or "right ankle".

We cast the task of pose estimation as a structured prediction problem and build our model based on the work

[14] https://pub.ist.ac.at/~akolesnikov/HDSeg/HDSeg.tar

[15] https://pub.ist.ac.at/~akolesnikov/HDSeg/data.tar

of Chen & Yuille (2014), which is one of the state-of-the-art methods for pose estimation. First, we construct an acyclic graph where the nodes $p \in \mathcal{V}$ correspond to the different body-parts. The set of body parts is extended from the original 14 parts of interest by the midway points to get the 26 nodes of the graph. Second, the graph is converted into the directed one by utilizing the arcs of both orientations for each original edge. We denote the resulting graph by $\mathcal{G} = (\mathcal{V}, \vec{\mathcal{E}})$.

In the model of Chen & Yuille (2014), each node $p \in \mathcal{V}$ has a variable $\boldsymbol{l}_p \in \mathcal{P} \subset \mathbb{R}^2$ denoting the spatial position of the corresponding joint that belongs to a finite set of possibilities $\mathcal{P}$; each arc $(p, q) \in \vec{\mathcal{E}}$ has a variable $t_{pq} \in \mathcal{T} = \{1, \dots, 13\}$ representing the type of spacial relationship between the two nodes. The output variable $\boldsymbol{y}$ is constructed by concatenating the unary and pairwise variables $\boldsymbol{y} = \big((\boldsymbol{l}_p)_{p\in\mathcal{V}},\ (t_{pq})_{(p,q)\in\vec{\mathcal{E}}}\big)$.

The structure score function is a function of discrete variables $\boldsymbol{l}_p$ and t_{pq} that is defined w.r.t. the graph $\mathcal{G}$:

$$\begin{aligned}\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y})\rangle = &\sum_{p\in\mathcal{V}} w_{U,p}\phi^U_p(\boldsymbol{I}_i, \boldsymbol{l}_p)\\ &+ \sum_{(p,q)\in\mathcal{E}} w_{T,pq}\phi^P_{pq}(\boldsymbol{I}_i, \boldsymbol{l}_p, t_{pq})\\ &+ \sum_{(p,q)\in\mathcal{E}} \langle \boldsymbol{w}_{P,pq,t_{pq}}, \Delta(\boldsymbol{l}_p - \boldsymbol{l}_q - \boldsymbol{r}^{t_{pq}}_{pq})\rangle.\end{aligned}$$

Here, the input $\boldsymbol{x}_i = \big(\boldsymbol{I}_i, (\boldsymbol{r}^t_{pq})^{t\in\mathcal{T}}_{(p,q)\in\vec{\mathcal{E}}}\big)$ consists of the original image $\boldsymbol{I}_i$ and the mean relative positions $\boldsymbol{r}^t_{pq} \in \mathbb{R}^2$ of each type of spatial relationship corresponding to each arc $(p, q) \in \vec{\mathcal{E}}$. Functions $\phi^U_p(\boldsymbol{I}_i, \cdot)$ and $\phi^P_{pq}(\boldsymbol{I}_i, \cdot, \cdot)$ compute the scores for each possible value of the discrete variables $\boldsymbol{l}_p$ and $(\boldsymbol{l}_p, t_{pq})$, respectively. The vector-valued function $\Delta(\boldsymbol{l}) = (l_1, l_1^2, l_2, l_2^2)$, $\boldsymbol{l} \in \mathbb{R}^2$, measures different types of mismatch between the preferred relative displacement $\boldsymbol{r}^{t_{pq}}_{pq}$ and the displacement $\boldsymbol{l}_p - \boldsymbol{l}_q$ coming from the labeling $\boldsymbol{y}$. The vector $\boldsymbol{w} = \big((w_{U,p})_{p\in\mathcal{V}}, (w_{T,pq})_{(p,q)\in\vec{\mathcal{E}}}, (\boldsymbol{w}_{P,pq,t})^{t\in\mathcal{T}}_{(p,q)\in\vec{\mathcal{E}}}\big)$ is the joint vector of parameters learned by structured SVM. Overall, this setup has $2,676$ parameters.

Displacements $\boldsymbol{r}^t_{pq}$ and functions ϕ^U_p, ϕ^P are computed at the preprocessing stage of the structured SVM. We follow Chen & Yuille (2014) and obtain the displacements $\boldsymbol{r}^t_{pq}$ with the K-means clustering of the displacements of the training set. Functions ϕ^U_p, ϕ^P consists in a Convolutional Neural Network (CNN) and are also learned from the training set. We refer the reader to the work of Chen & Yuille (2014) and their project page[16] for further details. Note that the last training stage of Chen & Yuille (2014) is different from ours, i.e., it consists in binary SVM on the carefully sampled sets of positive and negative examples.

To run SSVM, we define the structured error $L(\boldsymbol{y}_i, \boldsymbol{y})$ as a decomposable function w.r.t. the positions of the joints

$$L(\boldsymbol{y}_i, \boldsymbol{y}) = \tfrac{1}{|\mathcal{V}|}\sum_{p\in\mathcal{V}} \max\Big(1, \tfrac{\|\boldsymbol{l}_{i,p} - \boldsymbol{l}_p\|_2^2}{s_i^2}\Big)$$

where $\boldsymbol{l}_{i,p}$ belongs to the ground-truth labeling $\boldsymbol{y}_i$ and $\boldsymbol{l}_p$ belongs to the labeling $\boldsymbol{y}$. The quantity $s_i \in \mathbb{R}$ is the scaling factor and is defined by the distance between the left shoulder and the right hip in the ground-truth labeling $\boldsymbol{y}_i$. Similarly to Osokin & Kohli (2014), the complex dependence of the loss on the ground-truth labeling does not influence the complexity of the max oracle.

For the defined structured score and loss, the optimization problem of the max oracle can be exactly solved by the max-sum belief propagation algorithm on an acyclic graph with messages computed with the generalized distance transform (GDT) (Felzenszwalb & Huttenlocher, 2005). The usage of GDTs allows to significantly reduce the oracle running time, but requires positivity constraints on the connection weights $\boldsymbol{w}_{P,pq,t}$. BCFW with positivity constraints is described in Appendix H.

The resulting max oracle is quite slow (2 seconds per image) even when the belief propagation algorithm is optimized with GDTs. Slow running time made it intractable for us to run the experiments on the full original training set consisting of $1,000$ images. We use only the first 100 images and refer to this dataset as LSP-small. However, both the CNN training and clustering of the displacements were done on the original training set.

[16] http://www.stat.ucla.edu/~xianjie.chen/projects/pose_estimation/pose_estimation.html

J. Full experimental evaluation: comparing BCFW variants

In Figures 5 and 6, we give the detailed results of the experiments described in section 5.1. As a reminder, we are comparing different methods (caching vs no caching, gap sampling vs uniform sampling, pairwise FW steps vs regular FW steps) on different datasets in three main regimes w.r.t. λ. For each dataset, we use the good value of λ ("good" meaning the smallest possible test error) together with its smaller and larger values. The three regimes are displayed in the middle (b), top (a) and bottom (c) of each subfigure, respectively.

K. Full experimental evaluation: regularization path

In this section, we evaluate the regularization path method proposed in Section 4. Our experiments are organized in three stages. First, we choose the κ parameter for Algorithm 7 computing the ε-approximate regularization path. Second, we define and evaluate the heuristic regularization path. Finally, we compare both ε-approximate and heuristic paths against the grid search on multiple datasets.

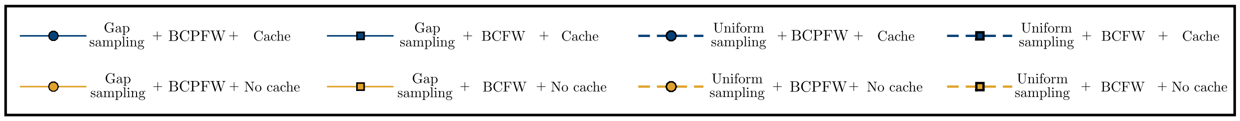
ε-approximate path. Algorithm 7 for computing the ε-approximate regularization path has one parameter κ controlling how large are the induction steps in terms of λ. This parameter provides the trade-off between the number of breakpoints and the accuracy of optimization for each breakpoint. We explore this trade-off in Figure 7a. For several values of κ, we report the cumulative number of effective passes (bottom plots) and cumulative time (top plots) required to get ε-approximate solution for each λ. We report both plots for the two methods used as the SSVM solver: BCPFW with gap sampling and caching; BCFW with gap sampling and without caching. We conclude that both too small (< 0.5) and too large (≈ 1) values of parameter κ result in slower methods, but overall the method is not too sensitive to κ. In all remaining experiments, we use $\kappa = 0.9$ when computing the ε-approximate path.

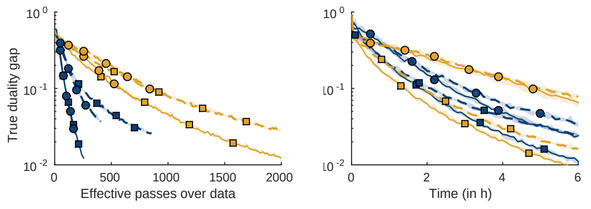
Heuristic path. When computing an ε-approximate regularization path, Algorithm 7 needs to perform at least one pass over the dataset at each breakpoint to check the convergence criterion, i.e., that the gap is not larger than ε. When having many breakpoints, this extra pass can be of significant cost, especially for large values of λ where the SSVM solver converges very quickly. However, in practice, we observe that the stale gap estimates are often good enough to determine convergence and actually checking the convergence criterion is not necessary. At the cost of loosing the guarantees, we can expect computational speed-up. We refer to the result of Algorithm 7 when the SSVM solver (at line 22) uses stale gap estimates instead of the exact gaps to check the convergence as a *heuristic path*. In the case of heuristic path, the parameter κ provides a trade-off between the running time and accuracy of the path. In Figure 7b, we illustrate this trade-off on the OCR-small dataset. For several values of κ, we report the true value of the duality gap at each breakpoint (the SSVM solver terminates when the stale gap estimate is below $\kappa\varepsilon$, $\varepsilon = 0.1$) and the cumulative time. We use BCFW + gap sampling and BCPFW + gap sampling + cache as the SSVM solvers. For large κ, we observe that the solutions for small λ are not ε-approximate and that the method in this regime requires less number of passes over the data, i.e., runs faster. In what follows we always use $\kappa = 0.7$ for the heuristic path as it is the largest value providing accurate enough results.

Comparison of paths against grid search. Finally, we compare the regularization path methods to the standard grid search approach. We define a grid of 31 values chosen to cover all the values of λ used in the experiments of Section J ($2^{15}, 2^{14}, \dots, 2^{-15}$). For each value of the grid, we optimize the SSVM objective optimize either independently or by warm-starting from the nearest larger value. We have considered two variants of warm start: keeping the primal variables and rescaling the dual variables, or keeping the dual variables and rescaling the primal variables. In our experiments, we do not notice any consistent difference between the two approaches, so we only use the first type of warm start.

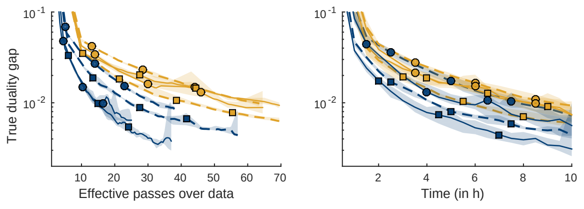
Figure 8 presents the results on the four datasets: HorseSeg-small (Figure 8a), OCR-small (Figure 8b), HorseSeg-medium (Figure 8c), OCR-large (Figure 8d).

For both the OCR-large and HorseSeg-medium datasets, neither heuristic, nor ε-approximate path methods did not reach their stopping criterion and were terminated at the time limit of 24 hours. In the case of OCR-large, the regularization path reached a value of λ smaller than the lower limit of the predefined grid, i.e., 2^{-15}. In the case of HorseSeg-medium, the grid search methods did not reach the lower bound of the grid and were terminated at the 24-hour time limit.

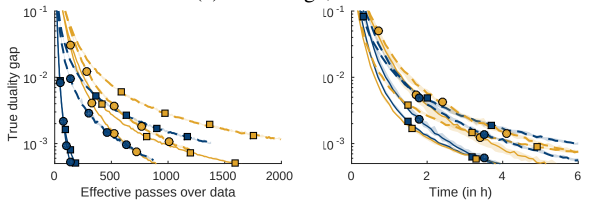

Gap sampling + BCPFW + Cache
Gap sampling + BCFW + Cache
Uniform sampling + BCPFW + Cache
Uniform sampling + BCFW + Cache
Gap sampling + BCPFW + No cache
Gap sampling + BCFW + No cache
Uniform sampling + BCPFW + No cache
Uniform sampling + BCFW + No cache



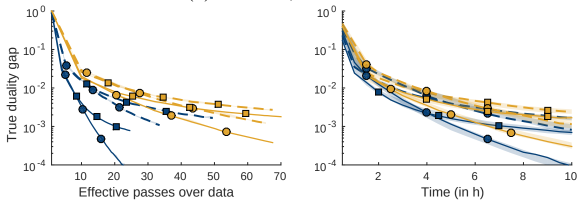
(a) OCR-large, $\lambda = 0.0001$



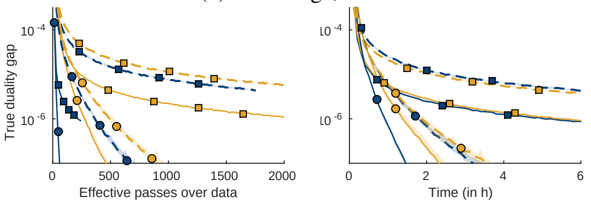
(a) CoNLL, $\lambda = 0.0001$



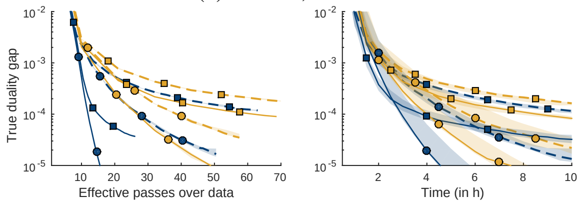
(b) OCR-large, $\lambda = 0.001$



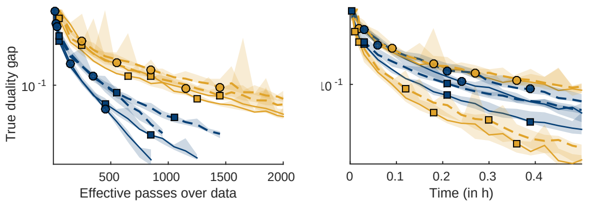
(b) CoNLL, $\lambda = 0.01$



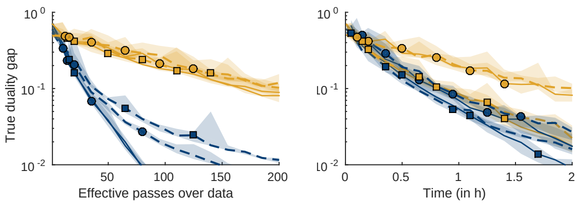
(c) OCR-large, $\lambda = 0.1$



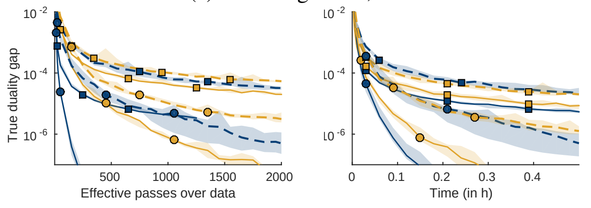
(c) CoNLL, $\lambda = 0.1$



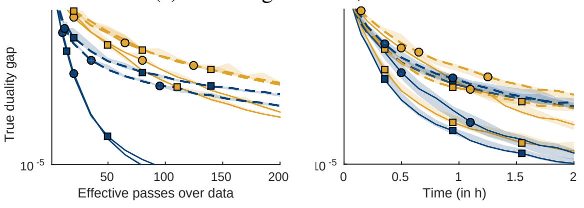
(a) HorseSeg-small, $\lambda = 0.1$



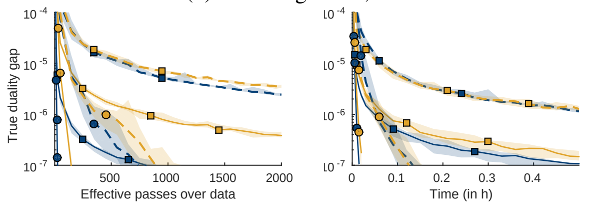
(a) HorseSeg-medium, $\lambda = 0.1$



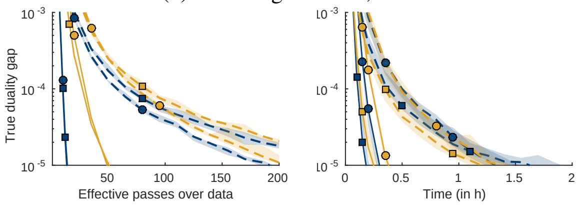
(b) HorseSeg-small, $\lambda = 100$



(b) HorseSeg-medium, $\lambda = 1$

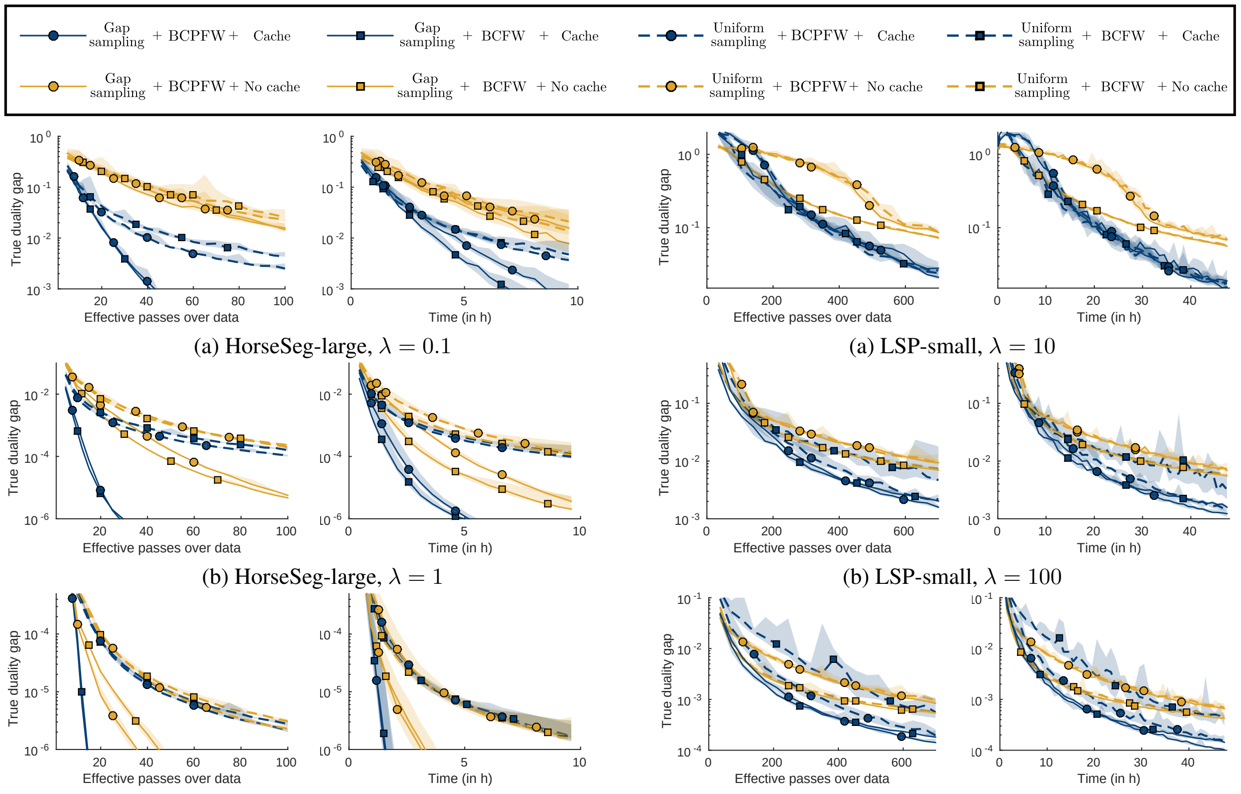


(c) HorseSeg-small, $\lambda = 1000$



(c) HorseSeg-medium, $\lambda = 10$

Figure 5. Comparison of the variants of BCFW. We compare 8 different methods that can be represented by 3 binary dimensions: object sampling, caching, type of FW steps. We represent these dimensions in different ways: the dimension of caching (in blue) versus no caching (in orange) is represented through colors, the dimension of gap sampling (solid lines) versus uniform sampling (dashed lines) is represented through line style, the dimension of pairwise FW steps (circle markers) versus regular FW steps (square markers) is represented through markers. For each method, we report both the number of effective passes over data (n oracle calls) and the running time against obtained duality gap (computed offline). *The figure is continued in Figure 6.*



(a) HorseSeg-large, $\lambda = 0.1$

(a) LSP-small, $\lambda = 10$

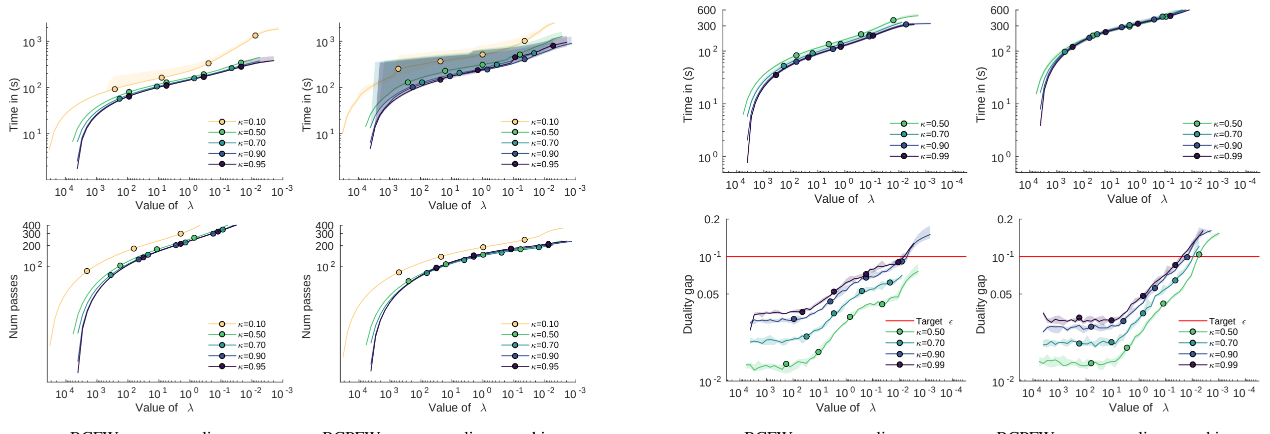
(b) HorseSeg-large, $\lambda = 1$

(b) LSP-small, $\lambda = 100$

(c) HorseSeg-large, $\lambda = 10$

(c) LSP-small, $\lambda = 1000$

Figure 6. Continuation of Figure 5. Comparison of the variants of BCFW on HorseSeg-large and LSP-small.



BCFW + gap sampling

BCPFW + gap sampling + caching

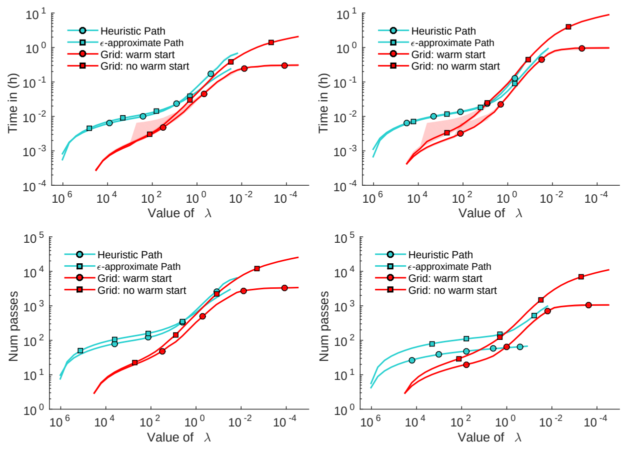
BCFW + gap sampling

BCPFW + gap sampling + caching

(a) ε-approximate regularization paths. For different values of κ, we report the cumulative number of effective passes (bottom) and the cumulative time (top) required to get an ε-approximate solution for each λ. We analyze the two methods: BCFW + gap sampling (left) and BCPFW + gap sampling + caching (right).

(b) Heuristic regularization paths. For different values of κ, we report the cumulative time required to compute a path until λ (top) along with the true value of the duality gap obtained for each λ (bottom).We analyze the two methods: BCFW + gap sampling (left) and BCPFW + gap sampling + caching (right).

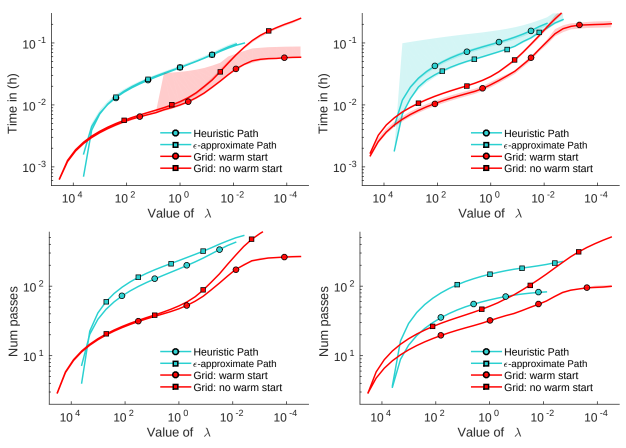
Figure 7. Experiment exploring the effect of κ for regularization paths computed on the OCR-small dataset.



BCFW + gap sampling

BCPFW + gap sampling + caching

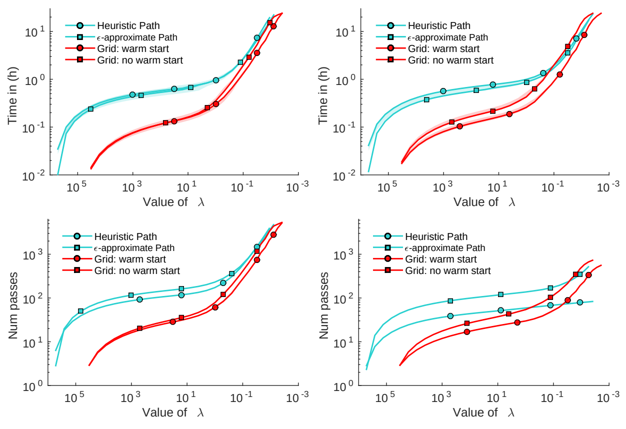
(a) HorseSeg-small



BCFW + gap sampling

BCPFW + gap sampling + caching

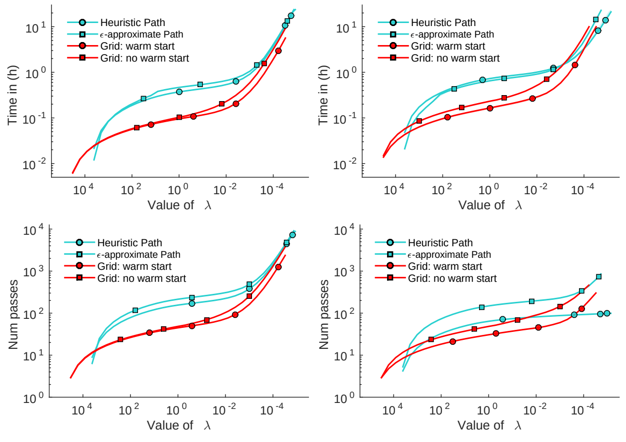
(b) OCR-small



BCFW + gap sampling

BCPFW + gap sampling + caching

(c) HorseSeg-medium



BCFW + gap sampling

BCPFW + gap sampling + caching

(d) OCR-large

Figure 8. Comparison of regularization path methods. In each subfigure, we compare the ε-approximate regularization path against the heuristic path and the grid search with/without warm start for a specific dataset. In each subfigure, we report the cumulative running time (top) and the cumulative effective number of passes (bottom) required to get to each value of the regularization parameter λ. We report results using two different methods as the SSVM solver: BCFW + gap sampling (left) and BCPFW + gap sampling + caching (right). Note that for OCR-large, the time limit of 24 hours was reached for the regularization path methods. However, the reached value of λ is smaller than the lower boundary of the grid 2^{-15}. For HorseSeg-medium, the time limit of 24 hours was reached for both the regularization path and grid search methods.

Supplementary References

Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., and Süsstrunk, S. SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 34 (11):2274–2282, 2012.

Allgower, E. and Georg, K. Continuation and path following. *Acta Numerica*, 2:1–64, 1993.

Beck, A. and Shtern, S. Linearly convergent away-step conditional gradient for non-strongly convex functions. *arXiv:1504.05002v1*, 2015.

Crammer, K. and Singer, Y. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research (JMLR)*, 2:265–292, 2001.

d'Aspremont, A., Guzmán, C., and Jaggi, M. An optimal affine invariant smooth minimization algorithm. *arXiv:1301.0465v2*, 2013.

Gärtner, B., Jaggi, M., and Maria, C. An exponential lower bound on the complexity of regularization paths. *Journal of Computational Geometry*, 3(1):168–195, 2012.

Giesen, J., Mueller, J., Laue, S., and Swiercy, S. Approximating concavely parameterized optimization problems. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

Guillaumin, M., Küttel, D., and Ferrari, V. ImageNet auto-annotation with segmentation propagation. *International Journal of Computer Vision (IJCV)*, 110(3):328–348, 2014.

Hastie, T., Rosset, S., Tibshirani, R., and Zhu, J. The entire regularization path for the support vector machine. *Journal of Machine Learning Research (JMLR)*, 2004.

Hoffman, A. J. On approximate solutions of systems of linear inequalities. *Journal of Research of the National Bureau of Standards*, 49(4):263–265, 1952.

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

Jun-Tao, L. I. and Ying-Min, J. I. A. Huberized multiclass support vector machine for microarray classification. *Acta Automatica Sinica*, 36(3):399–405, 2010.

Karasuyama, M. and Takeuchi, I. Suboptimal solution path algorithm for support vector machine. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2011.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2001.

Lee, Y. and Cui, Z. Characterizing the solution path of multicategory support vector machines. *Statistica Sinica*, 16: 391–409, 2006.

Lee, Y., Lin, Y., and Wahba, G. Multicategory support vector machines, theory, and application to the classification of microarray data and satellite radiance data. *Journal of the American Statistical Association*, 99:67–81, 2004.

Lempitsky, V., Vedaldi, A., and Zisserman, A. A Pylon model for semantic segmentation. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

Lowe, D. G. Distinctive image features from scale-invariant keypoint. *International Journal of Computer Vision (IJCV)*, 60(2):91–110, 2004.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. Building a large annotated corpus of English: The Penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

Okazaki, N. CRFsuite: a fast implementation of conditional random fields (CRFs), 2007. URL `http://www.chokkan.org/software/crfsuite/`.

Osokin, A. and Kohli, P. Perceptually inspired layout-aware losses for image segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014.

Sentelle, C. G., Anagnostopoulos, G. C., and Georgiopoulos, M. A simple method for solving the SVM regularization path for semidefinite kernels. *IEEE Transactions on Neural Networks and Learning Systems*, 2015.

Sha, F. and Pereira, F. Shallow parsing with conditional random fields. In *NAACL*, 2003.

Tibshirani, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society*, 1996.

Wainwright, M. J. and Jordan, M. I. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

Wang, L. and Shen, X. Multi-category support vector machines, feature selection and solution path. *Statistica Sinica*, 16:617–633, 2006.