A Multi-Plane Block-Coordinate Frank-Wolfe Algorithm for Training Structural SVMs with a Costly max-Oracle

Neel Shah, Vladimir Kolmogorov, Christoph H. Lampert

IST Austria

{neelshah,vnk,chl}@ist.ac.at

Abstract

Structural support vector machines (SSVMs) are amongst the best performing methods for structured computer vision tasks, such as semantic image segmentation or human pose estimation. Training SSVMs, however, is computationally costly, because it requires repeated calls to a structured prediction subroutine (called max-oracle*), which has to solve an optimization problem itself, e.g. a graph cut.*

In this work, we introduce a new algorithm for SSVM training that is more efficient than earlier techniques when the max-oracle is computationally expensive, as it is frequently the case in computer vision tasks. The main idea is to (i) combine the recent stochastic Block-Coordinate Frank-Wolfe algorithm with efficient hyperplane caching, and (ii) use an automatic selection rule for deciding whether to call the exact max-oracle or to rely on an approximate one based on the cached hyperplanes.

We show experimentally that this strategy leads to faster convergence towards the optimum with respect to the number of required oracle calls, and that this also translates into faster convergence with respect to the total runtime when the max-oracle is slow compared to the other steps of the algorithm. A C++ implementation is provided at http://www.ist.ac.at/~vnk.

1. Introduction

Many computer vision problems have a natural formulation as *structured prediction tasks*: given an input image the goal is to predict a structured output object, for example a segmentation mask or a human pose. Structural support vector machines (SSVMs) [28, 30], are currently one of the most popular methods for learning models that can perform this task from training data. In contrast to ordinary support vector machines (SVMs) [9], which only predict single values, e.g. a class label, SSVMs are designed such that, in principle, they can predict arbitrary structured objects. However, this flexibility comes at a cost: training an SSVM requires solving a more difficult optimization problem than training an ordinary SVM. In particular, SSVM

training requires repeated runs of the structured prediction step (the so called *max-oracle*) across the training set. Each of these steps is an optimization problem itself, e.g. finding the minimum energy labeling of a graph, and often computationally costly. In fact, the more challenging the problem is, the more the max-oracle calls become a computational bottleneck. This is also a major factor why SSVMs are typically only used for problems with small and medium-sized training sets, not for large scale training as it is common these days, for example, in object categorization [10].

In this work, we introduce a new variant of the Frank-Wolfe algorithm that is specifically designed for training SSVMs in situations where the calls to the max-oracle are the computational bottleneck. It extends the recently proposed block-coordinate Frank-Wolfe (BCFW) algorithm [20] by introducing a caching mechanism that keeps the results of earlier oracle calls in memory. In each step of the optimization, the method decides whether to call the exact max-oracle, or to reuse one of the results from the cache. The first option might allow the algorithm to make a larger steps towards the optimum, but it is slow. The second option will make smaller steps, but this might be justified, since every step will be much faster. Overall, a trade-off between both options will be optimal, and a second contribution of the manuscript is a geometrically motivated criterion for dynamically deciding at any time during the runtime of the algorithm, which choice is more promising.

We report on experiments on four different datasets that reflect a range of structured prediction scenarios: multiclass classification, sequence labeling, figure-ground segmentation and semantic image segmentation.

2. Structural Support Vector Machines

The task of *structured prediction* is to predict structured objects, $y \in \mathcal{Y}$, for given inputs, $x \in \mathcal{X}$. Structural support vector machines (SSVMs) [28, 30] offer a principled way for learning a structured prediction function, $h : \mathcal{X} \to \mathcal{Y}$, from training data in a maximum margin framework. We parameterize $h(x) = \arg\max_{y \in \mathcal{Y}} \langle w, \phi(x, y) \rangle$,

where $\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$ is a joint feature function of inputs and outputs, and $\langle \cdot, \cdot \rangle$ denotes the inner product in $\mathbb{R}^d$. The weight vector, w, is learned from a training set, $\{(x_1, y_1), \dots, (x_n, y_n)\}$, by solving the following convex optimization problem:

$$\min_w \quad \frac{\lambda}{2} \|w\|^2 + \sum_{i=1}^n H_i(w), \quad (1)$$

where $\lambda \geq 0$ is a regularization parameter. $H_i(w)$ is the (scaled) *structured hinge loss* that is defined as

$$H_i(w) = \frac{1}{n} \max_{y \in \mathcal{Y}} \big\{ \Delta(y_i, y) - \langle w, \phi(x_i, y_i) - \phi(x_i, y) \rangle \big\}, (2)$$

where $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is a task-specific loss function, for example the Hamming loss for image segmentation tasks.

Computing the value of $H_i(w)$, or the label that realizes this value, requires solving an optimization problem over the label set. We refer to the procedure to do so as the *max-oracle*, or just *oracle*. Other names for this in the literature are *loss-augmented inference*, or just *the* arg max *step*. It depends on the problem at hand how the max-oracle is implemented. We provide details of three choices and their properties in the extended technical report [26].

Structural SVMs have proved useful for numerous complex computer vision tasks, including *human pose estimation* [17, 31], *object detection* [2], *semantic image segmentation* [1, 23, 27], *scene reconstruction* [14, 25] and *tracking* [15, 21]. In this work, we concentrate not on the question if SSVMs learn better predictors than other methods, but we study Equation (1) from the point of a challenging optimization problem. We are interested in the question how fast for given data and parameters we can find the optimal, or a close-to-optimal, solution vector w. This is a question of high practical relevance, since training structured SVMs is known to be computationally costly, especially in a computer vision context where the output set, $\mathcal{Y}$, is large and the max-oracle requires solving a combinatoric optimization problem [23].

2.1. Related Work

Many algorithms have been proposed to solve the optimization problem (1) or equivalent formulations. In [28] and [30], where the problem was originally introduced, the authors derive a quadratic program (QP) that is equivalent to (1) but resembles the SVM optimization problem with slack variables and a large number of linear constraints.

The QP can be solved by a cutting-plane algorithm that alternates between calling the max-oracle once for each training example and solving a QP with a subset of constraints (cutting planes) obtained from the oracle. The algorithm was proved to reach a solution ϵ-close to the optimal one within $O(\frac{1}{\epsilon^2})$ step, i.e. $O(\frac{n}{\epsilon^2})$ calls to the max-oracle.

Joachims *et al.* improved this bound in [18] by introducing the *one-slack* formulation. It is also based on finding cutting planes, but keeps their number much smaller, achieving an improved convergence rate of $O(\frac{n}{\epsilon})$. The same convergence rate can also be achieved using *bundle methods* [29].

Ratliff *et al.* observed in [24] that one can also apply the *subgradient method* directly to the objective (1), which also allows for stochastic and online training. A drawback of this is that the speed of convergence depends crucially on the choice of a learning rate, which makes subgradient-based SSVM training often less appealing for practical tasks.

The Frank-Wolfe algorithm (FW) [12] is an elegant alternative: it resembles subgradient methods in the simplicity of its updates, but does not require a manual selection of the step size. Recently, Lacoste-Julien *et al.* introduced a block-coordinate variant of the Frank-Wolfe algorithm (BCFW) [20]. It achieves higher efficiency than the original FW algorithm by exploiting the fact that the SSVM objective can be decomposed additively into n terms, each of which is structured. Experiments in [20] show a significant speedup of BCFW compared to the original FW algorithm as well as the previously proposed techniques.

Almost simultaneously with [20], Branson et al. [6] proposed an "SVM-IS" method. It assumes an application-specific procedure called "ImportanceSample" that returns K solutions rather than 1 (the first of which is the result of the max-oracle). For $K = 1$ the method appears to be equivalent to the BCFW method in [20], except for a slight variation in the formulation of the dual problem.

BCFW can be considered the current state-of-the-art for SSVM training. However, in the next section we show that it can be significantly improved upon for computer vision tasks, in which the training time is dominated by calls to the max-oracle.

Besides the above techniques that use the max-oracle purely as a black box, several other approaches for accelerating SSVM training have been developed, e.g., using dual decomposing [19], restricting the search space to tractable subclasses [11], solving relaxed or smoothed optimization problems [16, 22], or exploiting the similarity between solutions [7]. Such techniques can also yield a significant speedup, but their applicability typically depends on the concrete form of the max-oracle.

3. Efficient SSVM Training

In this section we introduce our main contribution, the *multi-plane block-coordinate Frank-Wolfe* (MP-BCFW) algorithm for SSVM training. Because it builds on top of the FW and BCFW methods, we start by giving a more detailed explanation of the working mechanisms of these two algorithms. Afterwards, we highlight the improvements we make to tackle the situation when the max-oracle is computationally very costly.
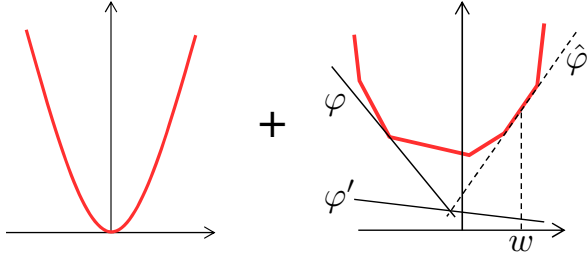
Figure 1. Frank-Wolfe algorithm for minimizing $\frac{\lambda}{2}||w||^2 + H(w)$. The vector φ specifies the current linear lower bound on H. One iteration involves three steps: (i) Compute the vector w that minimizes $\frac{\lambda}{2}||w||^2 + \langle \varphi, [w\ 1] \rangle$. (ii) Obtain a new linear bound $\hat{\varphi}$ on $H(\cdot)$ by computing a subgradient of H at w. (iii) Compute a linear interpolation φ' between φ and $\hat{\varphi}$ that maximizes $\mathcal{F}(\varphi')$, and set $\varphi \leftarrow \varphi'$.

First, we rewrite the structured Hinge loss term (2) more compactly as

$$H_i(w) = \max_{y \in \mathcal{Y}} \langle \varphi^{iy}, [w\ 1] \rangle, \qquad (3)$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product in $\mathbb{R}^{d+1}$, and $[w\ 1]$ is the concatenation of w with a single 1 entry. For a vector $\varphi \in \mathbb{R}^{d+1}$ we denote its first d components as $\varphi_\star \in \mathbb{R}^d$ and its last component as $\varphi_\circ \in \mathbb{R}$. The data vector φ^{iy} in (3) for $i = 1, \ldots, n$ and $y \in \mathcal{Y}$ is given by $\varphi^{iy}_\star = \frac{1}{n}(\phi(x_i, y) - \phi(x_i, y_i))$ and $\varphi^{iy}_\circ = \frac{1}{n}\Delta(y^i, y)$. Note that $\langle \varphi^{iy}, [w\ 1] \rangle = \langle \varphi^{iy}_\star, w \rangle + \varphi^{iy}_\circ$.

3.1. Frank-Wolfe algorithm

The Frank-Wolfe (FW) algorithm solves the SSVM training problem in its dual form. Writing $H(w) = \sum_{i=1}^n H_i(w)$ and introducing concatenated vectors $\varphi^{\bar{y}} = \varphi^{(y_1, \ldots, y_n)} = \sum_{i=1}^n \varphi^{iy_i}$, the primal problem becomes

$$\min_w \frac{\lambda}{2}||w||^2 + H(w), \ \text{for } H(w) = \max_{\bar{y} \in \overline{\mathcal{Y}}} \langle \varphi^{\bar{y}}, [w\ 1] \rangle, \ (4)$$

for $\overline{\mathcal{Y}} = \mathcal{Y} \times \cdots \times \mathcal{Y}$. Note that evaluating $H(w)$ for a given w requires n calls to the max-oracle, one for each of the terms $H_1(w), \ldots, H_n(w)$.

The FW algorithm maintains a (hyper)plane specified by a vector $\varphi \in \mathbb{R}^{d+1}$ that corresponds to a lower bound on $H(\cdot)$: $\langle \varphi, [w\ 1] \rangle \leq H(w)$ for all $w \in \mathbb{R}^d$. Such a plane exists, because H is convex. In fact, any $\varphi^{\bar{y}}$ for $\bar{y} \in \overline{\mathcal{Y}}$ has this property, as well as any convex combination of such planes, $\varphi = \sum_{\bar{y} \in \overline{\mathcal{Y}}} \alpha_{\bar{y}} \varphi^{\bar{y}}$ for $\sum_{\bar{y} \in \overline{\mathcal{Y}}} \alpha_{\bar{y}} = 1$ and $\alpha_{\bar{y}} \geq 0$. We call vectors φ of this form *feasible*.

Any feasible $\varphi = [\varphi_\star \ \varphi_\circ]$ provides a lower bound on (4), which we can evaluate analytically

$$\mathcal{F}(\varphi) = \min_w \left\{ \frac{\lambda}{2}||w||^2 + \langle \varphi, [w\ 1] \rangle \right\} = -\frac{1}{2\lambda}||\varphi_\star||^2 + \varphi_\circ. \ (5)$$

Maximizing $\mathcal{F}(\varphi)$ over all feasible vectors φ yields the tightest possible bound. This maximization problem is the

---

Algorithm 1 Frank-Wolfe algorithm for the dual of (4)

1: set $\varphi \leftarrow \varphi^{\bar{y}}$ for some $\bar{y} \in \overline{\mathcal{Y}}$
2: **repeat**
3:     compute $w \leftarrow \arg\min_w \frac{\lambda}{2}||w||^2 + \langle \varphi, [w\ 1] \rangle$;
    the solution is given by $w = -\frac{1}{\lambda}\varphi_\star$
4:     call oracle for vector w: compute $\hat{\varphi} \leftarrow \arg\max_{\varphi^{\bar{y}} : \bar{y} \in \overline{\mathcal{Y}}} \langle \varphi^{\bar{y}}, [w\ 1] \rangle$
5:     compute $\gamma \leftarrow \arg\max_{\gamma \in [0,1]} \mathcal{F}((1-\gamma)\varphi + \gamma\hat{\varphi})$ as follows:
    set $\gamma \leftarrow \frac{\langle \varphi_\star - \hat{\varphi}_\star, \varphi_\star \rangle - \lambda(\varphi_\circ - \hat{\varphi}_\circ)}{||\varphi_\star - \hat{\varphi}_\star||^2}$ and clip γ to $[0,1]$
    set $\varphi \leftarrow (1-\gamma)\varphi + \gamma\hat{\varphi}$
6: **until** some stopping criterion

---

dual to (4). The Frank-Wolfe algorithm is iterative procedure for maximizing $\mathcal{F}(\varphi)$. It is stated in pseudo code in Algorithm 1, and it is illustrated in Figure 1. Each iteration monotonically increases $\mathcal{F}(\varphi)$ (unless the maximum is reached), and the algorithm converges to an optimal solution with the rate $O(\frac{1}{\epsilon})$ with respect to the number of iterations, i.e. $O(\frac{n}{\epsilon})$ oracle calls [20].

3.2. Block-coordinate Frank-Wolfe algorithm

The *block-coordinate Frank-Wolfe algorithm* [20] also solves the dual of problem (1), but it improves over the FW algorithm by making use of the additive structure of the objective (4). Instead of keeping a single plane, φ, it maintains n planes, $\varphi^1, \ldots, \varphi^n$, such that the i-th plane is a lower bound on H_i: $\langle \varphi^i, [w\ 1] \rangle \leq H_i(w)$ for all $w \in \mathbb{R}^d$. Each such plane is obtained as a convex combination of the planes that define $H_i(\cdot)$, i.e. $\varphi^i = \sum_{y \in \mathcal{Y}} \alpha_{iy}\varphi^{yi}$ where $\sum_{y \in \mathcal{Y}} \alpha_{iy} = 1$ and $\alpha_{iy} \geq 0$.

We now call a vector $(\varphi^1, \ldots, \varphi^n)$ *feasible* if each φ^i is as above. The sum $\varphi = \sum_{i=1}^n \varphi^i$ then defines a plane that lower bounds $H(w) = \sum_{i=1}^n H_i(w)$, i.e. $\langle \varphi, [w\ 1] \rangle \leq H(w)$ for all $w \in \mathbb{R}^d$. Therefore, $\mathcal{F}(\varphi)$, as defined by (5), is again a lower bound on problem (1), and the goal is again to maximize this bound over all feasible vectors $(\varphi^1, \ldots, \varphi^n)$.

BCFW does so by a block-coordinate strategy. It picks an index $i \in [n] = \{1, \ldots, n\}$ and updates the component φ^i while keeping all other components fixed. During this step the terms $H_j(w)$ for $j \neq i$ are approximated by linear functions $\langle \varphi^j, [w\ 1] \rangle$, and the algorithm tries to find a new linear approximation for $H_i(\cdot)$ that gives a larger bound $\mathcal{F}(\sum_{j=1}^n \varphi^j)$. Pseudo code for BCFW is given in Algorithm 2, and it is illustrated in Figure 2.

3.3. Multi-Plane Block-Coordinate Frank-Wolfe

It has been shown in [20] that for training SSVMs, BCFW needs much fewer passes through the training data than the FW algorithm as well as earlier approaches, such as the cutting plane and stochastic subgradient methods. However, it still has one suboptimal feature that can be improved upon: the computation efforts in each BCFW step are very unbalanced. For each oracle call (line 6 in Alg. 2) there is

Algorithm 2 Block-coordinate Frank-Wolfe (BCFW) algorithm for the dual of (1)

1: for each $i \in [n]$ set $\varphi^i \leftarrow \varphi^{iy}$ for some $y \in \mathcal{Y}$
2: set $\varphi = \sum_{i=1}^n \varphi^i$
3: **repeat**
4:   pick $i \in [n]$, e.g. uniformly at random
5:   compute $w \leftarrow \arg\min_w \frac{\lambda}{2}||w||^2 + \langle \varphi, [w\ 1] \rangle$
    the solution is given by $w = -\frac{1}{\lambda}\varphi_\star$
6:   call i-th oracle for vector w: $\ \hat{\varphi}^i \leftarrow \arg\max_{\varphi^{iy}:y\in\mathcal{Y}} \langle \varphi^{iy}, [w\ 1] \rangle$
7:   compute $\gamma \leftarrow \arg\max_{\gamma\in[0,1]} \mathcal{F}(\varphi - \varphi^i + (1-\gamma)\varphi^i + \gamma\hat{\varphi}^i)$:
    set $\gamma \leftarrow \frac{\langle \varphi^i_\star - \hat{\varphi}^i_\star, \varphi_\star \rangle - \lambda(\varphi^i_\circ - \hat{\varphi}^i_\circ)}{||\varphi^i_\star - \hat{\varphi}^i_\star||^2}$ and clip γ to $[0,1]$
8:   set $\varphi^i_{\text{old}} \leftarrow \varphi^i$, $\ \varphi^i \leftarrow (1-\gamma)\varphi^i + \gamma\hat{\varphi}^i$, $\ \varphi \leftarrow \varphi + \varphi^i - \varphi^i_{\text{old}}$
9: **until** some stopping criterion

---

only $\Theta(d)$ amount of additional work (lines 5,7,8), and this is often negligible compared to the time taken by the oracle. We could easily afford to do more work per oracle call without significantly changing the running time of one iteration. Our goal is therefore to exploit this extra freedom to accelerate convergence, thereby decreasing the number of required oracle calls and the total runtime of the algorithm.

Our main insight is that BCFW acts wastefully by discarding the plane $\hat{\varphi}^i$ after completing the iteration for the term H_i, even though it required an expensive call to the max-oracle to obtain $\hat{\varphi}^i$. We propose to retain some of these planes, maintaining a *working set* $\mathcal{W}_i$ for each $i = 1, \ldots, n$. We proceed similarly to [18], where a working set was used in a cutting-plane framework. Whenever the oracle for H_i is called, the obtained plane $\hat{\varphi}^i$ is added to $\mathcal{W}_i$. Planes are removed again from $\mathcal{W}_i$ after a certain time, unless in the mean time they have become *active* (see below). Consequently, at any iteration the algorithm has access to multiple planes instead of just one, which is why we name the proposed algorithm *multi-plane block-coordinate Frank-Wolfe (MP-BCFW)*.

The working set, $\mathcal{W}_i$, allows us to define an alternative mechanism for increasing the objective value that does not require a call to the costly max-oracle. We define an approximation $\tilde{H}_i(w)$ of $H_i(w)$ by

$$\tilde{H}_i(w) = \max_{\tilde{\varphi}^i \in \mathcal{W}_i} \langle \tilde{\varphi}^i, [w\ 1] \rangle. \tag{6}$$

Note that $\tilde{H}_i(w) \leq H_i(w)$ for all $w \in \mathbb{R}^d$, since the maximization is performed over a smaller set. For any $i \in [n]$, we can perform block updates like in BCFW, but with the term $\tilde{H}_i$ instead of H_i, i.e. in step 6 of Algorithm 2 we set $\hat{\varphi}^i \leftarrow \arg\max_{\tilde{\varphi}^i \in \mathcal{W}_i} \langle \tilde{\varphi}^i, [w\ 1] \rangle$. Such approximate oracle steps will increase $\mathcal{F}(\varphi)$, but potentially less so than a BCFW update using the exact expression.[1]

---

[1]Note that the function $\langle \varphi^i, [w\ 1] \rangle$ may not be a lower bound on $\tilde{H}(w)$: the plane φ^i is a convex combination of planes $\{\varphi^{iy} : y \in \mathcal{Y}\}$, but some of these planes may have been removed from the working set $\mathcal{W}_i$.

Algorithm 3 Multi-Plane Block-Coordinate Frank-Wolfe (MP-BCFW) algorithm. Parameters: N, M, T.

1: for each $i \in [n]$ set $\varphi^i \leftarrow \varphi^{iy}$ for some $y \in \mathcal{Y}$,
2: set $\varphi = \sum_{i=1}^n \varphi^i$
3: **if** $N > 0$ **then** $\mathcal{W}_i = \{\varphi^i\}$ **else** $\mathcal{W}_i = \varnothing$ for each $i \in [n]$
4: **repeat** until some stopping criterion
5:   do one pass through $i \in [n]$ in random order,
    for each i do the following:
      run BCFW update using original term $H_i(w)$
      add obtained vector $\hat{\varphi}^i$ to $\mathcal{W}_i$
      if $|\mathcal{W}_i| > N$ then remove longest inactive plane from $\mathcal{W}_i$
6:   do up to M passes (see text) through $i \in [n]$ in random order,
    for each i do the following:
      run BCFW update with term $\tilde{H}_i(w) = \max_{\tilde{\varphi}^i \in \mathcal{W}_i} \langle \tilde{\varphi}^i, [w\ 1] \rangle$
      remove planes from $\mathcal{W}_i$ that have not been active during the last T outer iterations
7: **end repeat**

We propose to interleave the approximate updates steps with exact updates. The order of operations in our current implementation is shown in Algorithm 3. We refer to one pass through the data in lines 5 and 6 as an *exact* and an *approximate pass* respectively, and to steps 5-6 as an *(outer) iteration*. Thus, each iteration contains 1 exact pass and up to M approximate passes. The parameter N bounds the number of stored planes per term: $|\mathcal{W}_i| \leq N$ for each i. In this algorithm a plane is considered *active* at a given moment if it is returned as optimal by either an exact or an approximate oracle call.

The complexity of one approximate update in step 6 is $O(Nd)$, therefore the algorithm performs $O(MNd)$ additional work per each oracle call. For $M = N = 0$, MP-BCFW reduces to the classical BCFW algorithm. Since MP-BCFW in particular performs all steps that BCFW does, it inherits all convergence guarantees from the earlier algorithm, such as a convergence rate of $O(\frac{1}{\epsilon})$ towards the optimum, as well as the guarantee of convergence even when the max-oracle can solve the problem only approximately (see [20] for details). However, as we will show in Section 4, MP-BCFW gets more "work done" per iteration and therefore converges faster with respect to the number of max-oracle calls.

3.4. Automatic Parameter Selection

The optimal number of planes to keep per term as well as the optimal number of efficient approximate passes to run depends on several factors, such as the number of support vectors and how far the current solution still is from the optimal one. Therefore, we propose not to set these parame-

---

However, this property is not required for the correctness of the method. Indeed, it follows from the construction that in Algorithm 3 the vector φ^i is a always convex combination of planes $\{\varphi^{iy} : y \in \mathcal{Y}\}$ for each i, and each step is guaranteed not to decrease the bound $\mathcal{F}(\sum_{i=1}^n \varphi^i)$. As a consequence, the convergence properties proven in [20] still hold.
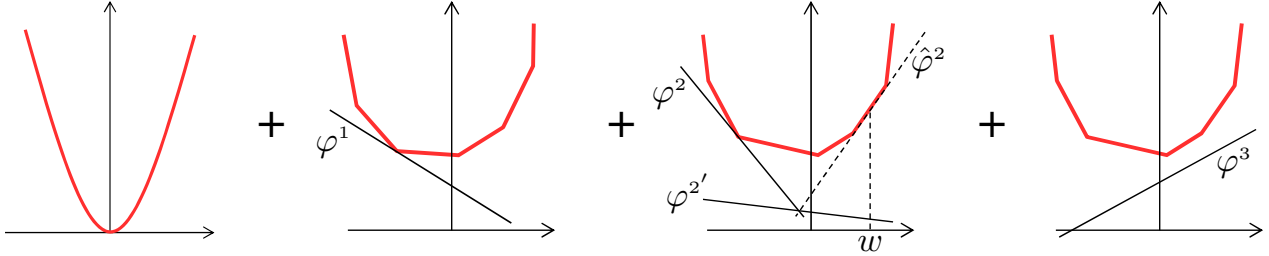
Figure 2. Block-coordinate Frank-Wolfe algorithm for minimizing $\frac{\lambda}{2}||w||^2 + \sum_{i=1}^{n} H_i(w)$ with $n = 3$. Each vector φ^i specifies the current linear lower bound on H_i. One iteration for term $i = 2$ is similar to the classical Frank-Wolfe algorithm, except that the terms H_1 and H_3 are approximated by linear lower bounds φ^1 and φ^3, respectively, which are kept fixed during this step.

ters to fixed values but to adapt them dynamically over time in a data-dependent way. The first criterion described below is fairly standard [18], but the second criterion is specific to MP-BCFW and forms a second contribution of this work.

Working set size. We observe that the working set is bounded not only by its upper bound parameter, N, but also by the mechanism that automatically removes inactive planes. Since the second effect is more interpretable, we suggest to set N to a large value, and rely on the parameter T to control the working set size. In effect, the actual number of planes is adjusted in a data-dependent way for each training instance. In particular, for terms with few relevant planes (support vectors) the working set will be small. A side effect of this is an acceleration of the algorithm, since the runtime of the approximate oracle is proportional to the actual number of planes in their working set.

Number of approximate passes. We set the maximal number of approximate passes per iteration, M, to a large value and rely on the following geometrically motivated criterion instead. After each approximate pass we compute two quantities: 1) the increase in $\mathcal{F}(\varphi)$ per time unit (i.e. the difference of function values divided by the runtime) of the most recent approximate pass, and 2) the increase in $\mathcal{F}(\varphi)$ per unit time of the complete sequence of steps since starting the current iteration. If the former value is smaller than the latter, we stop making approximate passes and start a new iteration with an exact pass.

The above criterion can be understood as an extrapolation of the recent behavior of the runtime-vs-function value graph into the future, see Figure 3. If the slope of the last segment is higher than the slope of the current iteration so far, then the expected increase from another approximate pass is high enough to justify its cost. Otherwise, it is more promising to start a new iteration.

3.5. Weighted averaging of iterates

The convergence speed of stochastic optimization methods can often be improved further by taking weighted averages of iterates [3]. In this work, we use the same averaging scheme as [20]: writing $\varphi^{(k)}$ for the vector pro-
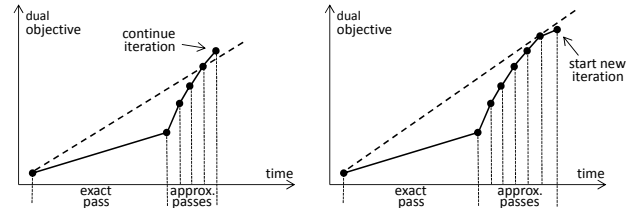

Figure 3. Geometric criterion for number of approximate passes. After each approximate pass, we compare the relative progress (increase in objective per time = slope of the last black line segment) to the relative progress of the complete iteration so far (slope of the dashed line). If the former is higher (left), we make another approximate pass. Otherwise, we start a new iteration (right).

duced after the k-th oracle call in Algorithm 2 (BCFW), the k-th averaged vector is $\bar{\varphi}^{(k)} = \frac{2}{k(k+1)} \sum_{t=1}^{k} t\varphi^{(t)}$. It can also be computed incrementally via $\bar{\varphi}^{(k+1)} = \frac{k}{k+2}\bar{\varphi}^{(k)} + \frac{2}{k+2}\varphi^{(k+1)}$ for $k \geq 1$. While this step has no impact on the dual objective, several studies have shown that the iterates $\bar{w}^{(k)} = -\frac{1}{\lambda}\bar{\varphi}_\star^{(k)}$ typically converge to the optimum significantly faster than the iterates $w^{(k)} = -\frac{1}{\lambda}\varphi_\star^{(k)}$. This is also confirmed in our experiments.

For inclusion into MP-BCFW we extended the above scheme by maintaining two vectors, $\bar{\varphi}^{(k)}$ and $\bar{\bar{\varphi}}^{(k')}$. They are updated after every exact and after every approximate oracle call, respectively, using the above formula. When we need to extract a solution, we compute the interpolation between $\bar{\varphi}^{(k)}$ and $\bar{\bar{\varphi}}^{(k')}$ that gives the best dual objective score. By this construction we overcome the problem that the two types of oracle calls have quite different characteristics, and thus may require different weights.

We refer to the averaged variants of BCFW and MP-BCFW as BCFW-avg and MP-BCFW-avg, respectively.

4. Experiments

We implemented the MP-BCFW algorithm in C++[2] and analyzed its effectiveness by performing experiments for four different setting: multi-class classification, sequence labeling, figure-ground segmentation and semantic image
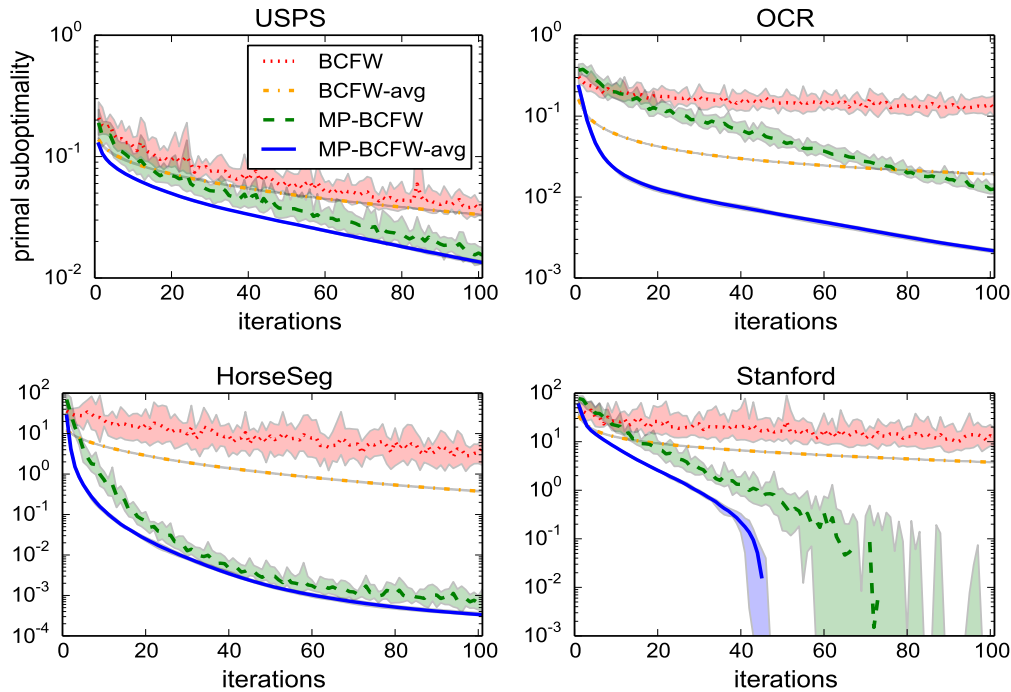

Figure 4. *Oracle convergence* on the four benchmark datasets. Shaded areas indicate minimum and maximum values over 10 repeats. In all cases, the multi-plane algorithms, MP-BCFW and MP-BCFW-avg, require fewer iterations to reach high quality solutions than the single plane variants, BCFW and BCFW-avg.

segmentation. The first two rely on generic datasets that were used previously to benchmark SSVM training (multi-class classification on the USPS dataset[3] and sequence labeling on the OCR dataset[4]). The third task (figure-ground segmentation on part of the HorseSeg dataset[5]) and the fourth (multiclass semantic image segmentation on the Stanford background dataset[6]) show a typical feature of computer vision tasks: the max-oracle is computationally costly, much more so than in the previous two cases, which results in a strong computational bottleneck for training. In the case of multiclass segmentation the max-oracle is even NP-hard and can be solved only approximately, another common property for computer vision problems. Exact details of dataset characteristics, feature representations and implementation of the oracles for the four datasets are provided in the technical report [26].

We focus on comparing MP-BCFW with BCFW (with and without averaging), since in [20] it has already been shown that BCFW offers a substantial improvements over earlier algorithms, in particular classical FW [12], cutting plane training [18], exponentiated gradient [8] and stochastic subgradient training [24]. Since all algorithms solve the same convex optimization problem and will ultimately arrive at the same solution, we are only interested in the convergence speed, not in the error rates of the resulting predictors. This allows us to adopt an easy experimental setup in which we use all available training data for learning and do not have to set aside data for performing model selection. In line with earlier work, we regularize using $\lambda = 1/n$.

To be comparable to earlier work, in particular [20], we report the progress of the optimization in terms of the *primal suboptimality*. This value is the difference between the current value of the primal objective and the highest value of the dual objective (i.e. the lower bound) that we observed during any of our experiments for any of the methods. As such, the primal suboptimality is a conservative estimate of how far a current solution is from the (unknown) optimal one. For the Stanford dataset evaluating the primal objective exactly is intractable, so the reported values are only approximate. As a consequence, the (approximate) primal suboptimality can become negative for this dataset, in which case we interrupt the logarithmic plots. Results for other quality measures, in particular the *dual suboptimality* and *duality gap* are reported in the technical report [26].

We visualize the results in two coordinate frames: i) with respect to the number of iterations, and ii) with respect to the actual runtime. The first quantity, which we refer to as *oracle convergence*, measures how efficiently the algorithm uses the statistical information that is present in the

---

[3] http://www-i6.informatik.rwth-aachen.de/˜keysers/usps.html
[4] http://www.seas.upenn.edu/˜taskar/ocr/
[5] http://www.ist.ac.at/˜akolesnikov/HDSeg/
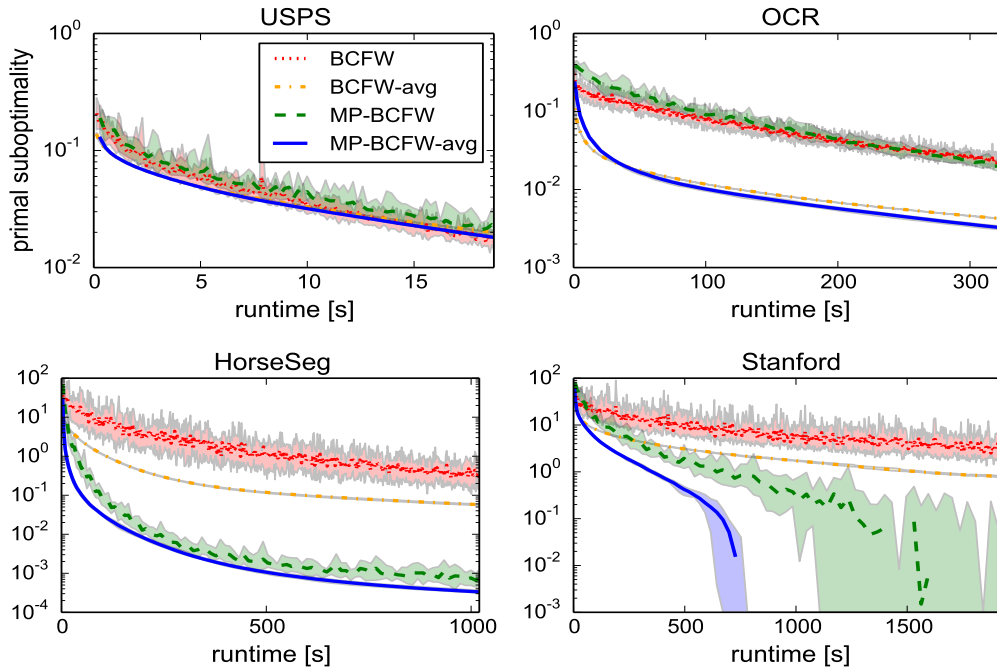[6] http://dags.stanford.edu/projects/scenedataset.html

Figure 5. *Runtime convergence* on the four benchmark datasets. Shaded areas indicate minimum and maximum values over 10 repeats. When the max-oracle is fast (USPS and OCR), the multi-plane algorithms (MP-BCFW, MP-BCFW-avg) behave similarly to their single-plane counterparts (BCFW, BCFW-avg) due to the automatic parameter adjustment. When the max-oracle is computationally costly (HorseSeg, Stanford) the multi-plane variants converge substantially faster.

training examples. It is independent of the implementation and therefore comparable between publications. The second value, called *runtime convergence*, is of practical interest, because it reflects the computational resources required to achieve a certain solution quality. However, it depends on the concrete implementation and computing hardware.[7] To nevertheless obtain fair runtime comparisons, we use the same code base for all methods, making use of the fact that BCFW can be recovered from MP-BCFW with minimal overhead by deactivating the working sets and approximate passes ($N = 0$, $M = 0$). For MP-BCFW we rely on the automatic parameter selection mechanism and set $T = 10$, $N = 1000$, $M = 1000$, where the latter two just act as high upper bounds that do not influence the system's behaviour.

4.1. Results

Figure 4 shows the *oracle convergence* results for all datasets. One can see that within the same number of iterations (and thereby calls to the max-oracle), MP-BCFW always achieves lower primal suboptimality than BCFW. Similarly, MP-BCFW-avg improves over BCFW-avg.

This effect is stronger for OCR, HorseSeg and Stanford than for USPS, which makes sense, since the latter dataset has a very small label space ($|\mathcal{Y}| = 10$), so the number of support vectors per example is small, which limits the ben-

efits of having access to more than one plane. The graph labeling tasks OCR, HorseSeg and Stanford have a larger label spaces, so one can expect more support vectors to contribute to the score. Reusing planes from previous iterations can be expected to have a beneficial effect.

Figure 5 illustrates the *runtime convergence*, i.e. the values on the vertical axis are identical to Figure 4, but the horizontal axis shows the actual runtime instead of the number of iterations. One can see that for the USPS dataset, the better oracle convergence did not translate to actually faster convergence, and for OCR, the difference between single-plane and multi-plane methods is small.

The situation is different for the HorseSeg and Stanford datasets, which are more typical computer vision tasks. For them, MP-BCFW and MP-BCFW-avg converge substantially faster than BCFW and BCFW-avg. The differences can be explained by the characteristics of the max-oracle in the different optimization problems: for USPS and OCR these are efficient and not do not form major computational bottlenecks. For USPS, the max-oracle requires only computing ten inner products and identifying their maximum. This takes only a few microseconds on modern hardware. Overall, the BCFW algorithm spends approximately 15% of its total runtime on oracle calls. For OCR, the max-oracle is implemented efficiently via Dynamic Programming (Viterbi algorithm), which takes approximate 50µs on our hardware. Overall, oracle calls make up for approximately 70% of

---

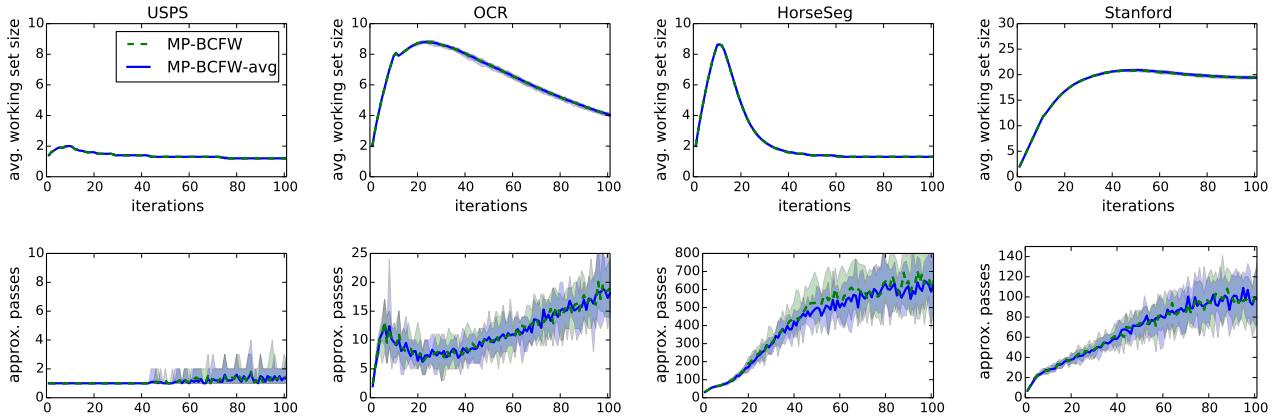[7] All experiments were done on a PC with 3.6GHz Intel Core i7 CPU.

Figure 6. *Automatic parameter selection.* The size of the working set (top) and the number of approximate passes per iteration (bottom) are adjusted in a data- and runtime-dependent way. See Section 4.2 for details.

BCFW's runtime. In both settings, the ratio of time spent for the oracle calls and the time spent elsewhere is not high enough to justify frequent use of the approximate oracle.

Note, however, that even for USPS, MP-BCFW is also not significantly slower than BCFW, either. This indicates that the automatic parameter selection does its job as intended: if the overhead of a large working set and many approximate passes would be larger than the benefit they offer, the selection rule falls back to the established (and efficient) baseline behavior.

For the other two datasets, the max-oracle calls are strong computational bottlenecks. For HorseSeg, each oracle call consists of minimizing a submodular energy function by means of solving a maximum flow problem [4]. For the Stanford dataset, the oracle consists of minimizing a *Potts*-type energy function for which we rely on the efficient preprocessing from [13] followed by the approximation algorithm from [5]. Even with optimized implementations the oracle calls take 1 to 5 milliseconds. As a consequence, BCFW spends almost 99% of the total training time on oracle calls. With MP-BCFW, the fraction drops to less than 25% because the parameter selection mechanism decides that the time it takes to make one exact oracle call is often spent better for making several approximate oracle calls.

4.2. Automatic parameter selection

Figure 6 illustrates the dynamic behavior of the parameter selection for MP-BCFW in more detail. The first row shows the average size of the working set per term over the course of the optimization, the second row shows the number of approximate passes per iteration.

One can see that for USPS the number of planes and iterations remain small through the runtime of the algorithm, making MP-BCFW behave almost as BCFW. For OCR, the algorithm initially identifies several relevant planes, but this number is reduced for later iterations. The number of approximate passes per iteration ranges between 10 to 20.

HorseSeg shows a similar pattern but in more extreme form. After an initial exploration phase the number of relevant planes stabilizes at a low value, while the number of approximate passes grows quickly to several hundred. For Stanford, the working set grows steadily to reach a stable state, but the number of approximate passes grows slowly during the course of the optimization.

In summary, one can see that the automatic parameter selection shows a highly dynamic behaviour. It allows MP-BCFW to adapt to the complexity of the objective function as well as to dynamic properties of the optimization, such as how far from the objective the current solution is.

5. Summary and Conclusion

We have presented MP-BCFW, a new algorithm for training structured SVMs that is specifically designed for the situation of computationally costly max-oracles, as they are common in computer vision tasks. The main improvement is the option to re-use previously computed oracle results by means of a per-example working set, and to alternate between calls to the exact but slow oracle and calls to an approximate but fast oracle. We also introduce rules for dynamically choosing the working set size and number of approximate passes depending on the algorithm's runtime behaviour and its progress towards the optimum. Overall, the result is a easy-to-use method with default settings that work well for a range of different scenarios.

Our experiments showed that MP-BCFW always requires fewer iterations to reach a certain solution quality than the BCFW algorithm, which had previously been shown to be superior to earlier methods. This leads to faster convergence towards the optimum for situations in which the max oracle is the computational bottleneck, as it is typical for structured learning tasks in computer vision.

In future work we plan to explore the situation of high-dimensional or kernelized SSVMs, where a further acceleration can be expected by caching inner product evaluations.

Acknowledgements

References

[1] L. Bertelli, T. Yu, D. Vu, and B. Gokturk. Kernelized structural SVM learning for supervised object segmentation. In *CVPR*, 2011. 2

[2] M. B. Blaschko and C. H. Lampert. Learning to localize objects with structured output regression. In *ECCV*, 2008. 2

[3] L. Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. 2012. 5

[4] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *PAMI*, 26(9):1124–1137, 2004. 8

[5] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *PAMI*, 23(11), 2001. 8

[6] S. Branson, O. Beijbom, and S. Belongie. Efficient large-scale structured learning. In *CVPR*, 2013. 2

[7] K.-W. Chang, S. Upadhyay, G. Kundu, and D. Roth. Structural learning with amortized inference. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2015. 2

[8] M. Collins, A. Globerson, T. Koo, X. Carreras, and P. L. Bartlett. Exponentiated gradient algorithms for conditional random fields and max-margin Markov networks. *JMLR*, 9:1775–1822, 2008. 6

[9] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995. 1

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale hierarchical image database. In *CVPR*, 2009. 1

[11] V. Franc and B. Savchynskyy. Discriminative learning of max-sum classifiers. *JMLR*, 9:67–104, 2008. 2

[12] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:149–154, 1956. 2, 6

[13] I. Gridchyn and V. Kolmogorov. Potts model, parametric maxflow and k-submodular functions. In *ICCV*, 2013. 8

[14] A. Gupta, M. Hebert, T. Kanade, and D. M. Blei. Estimating spatial layout of rooms using volumetric reasoning about objects and surfaces. In *NIPS*, 2010. 2

[15] S. Hare, A. Saffari, and P. H. Torr. Struck: Structured output tracking with kernels. In *ICCV*, 2011. 2

[16] T. Hazan and R. Urtasun. A primal-dual message-passing algorithm for approximated large scale structured prediction. In *NIPS*, 2010. 2

[17] C. Ionescu, L. Bo, and C. Sminchisescu. Structural SVM for visual localization and continuous state estimation. In *ICCV*, 2009. 2

[18] T. Joachims, T. Finley, and C. Yu. Cutting-plane training of structural SVMs. *Machine Learning*, 1:27–59, 2009. 2, 4, 5, 6

[19] N. Komodakis. Efficient training for pairwise or higher order CRFs via dual decomposition. In *CVPR*, 2011. 2

[20] S. Lacoste-Julien, M. Jaggi, M. Schmidt, and P. Pletscher. Block-coordinate Frank-Wolfe optimization for structural SVMs. In *ICML*, 2013. 1, 2, 3, 4, 5, 6

[21] X. Lou and F. A. Hamprecht. Structured learning for cell tracking. In *NIPS*, 2011. 2

[22] O. Meshi, D. Sontag, T. Jaakkola, and A. Globerson. Learning efficiently with approximate inference via dual losses. In *ICML*, 2010. 2

[23] S. Nowozin, P. V. Gehler, and C. H. Lampert. On parameter learning in CRF-based approaches to object class image segmentation. In *ECCV*, 2010. 2

[24] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich. (Online) subgradient methods for structured prediction. In *Conference on Uncertainty in Artificial Intelligence (AISTATS)*, 2007. 2, 6

[25] A. G. Schwing, T. Hazan, M. Pollefeys, and R. Urtasun. Efficient structured prediction for 3d indoor scene understanding. In *CVPR*, 2012. 2

[26] N. Shah, V. Kolmogorov, and C. H. Lampert. A multi-plane block-coordinate Frank-Wolfe algorithm for training structural SVMs with a costly max-oracle. *CoRR*, abs/1408.6804, 2014. 2, 6

[27] M. Szummer, P. Kohli, and D. Hoiem. Learning CRFs using graph cuts. In *ECCV*, 2008. 2

[28] B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. In *NIPS*, 2004. 1, 2

[29] C. H. Teo, S. Vishwanthan, A. J. Smola, and Q. V. Le. Bundle methods for regularized risk minimization. *JMLR*, 11:311–365, 2010. 2

[30] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *JMLR*, 6:1453–1484, 2005. 1, 2

[31] W. Yang, Y. Wang, and G. Mori. Recognizing human actions from still images with latent poses. In *CVPR*, 2010. 2