Fast and Scalable Structural SVM with Slack Rescaling

Heejin Choi **Ofer Meshi** **Nathan Srebro**
Toyota Technological Institute at Chicago
6045 S. Kenwood Ave. Chicago, Illinois 60637 USA

Abstract

We present an efficient method for training slack-rescaled structural SVM. Although finding the most violating label in a margin-rescaled formulation is often easy since the target function decomposes with respect to the structure, this is not the case for a slack-rescaled formulation, and finding the most violated label might be very difficult. Our core contribution is an efficient method for finding the most-violating-label in a slack-rescaled formulation, given an oracle that returns the most-violating-label in a (slightly modified) margin-rescaled formulation. We show that our method enables accurate and scalable training for slack-rescaled SVMs, reducing runtime by an order of magnitude compared to previous approaches to slack-rescaled SVMs.

1 Introduction

Many problems in machine learning can be seen as structured output prediction tasks, where one would like to predict a set of labels with rich internal structure [1]. This general framework has proved useful for a wide range of applications from computer vision, natural language processing, computational biology, and others. In order to achieve high prediction accuracy, the parameters of structured predictors are learned from training data. One of the most effective and commonly used approaches for this supervised learning task is *Structural SVM*, a method that generalizes binary SVM to structured outputs [15]. Since the structured error is non-convex, Tsochantaridis et al. [15] propose to replace it with a convex surrogate loss function. They formulate two such surrogates, known as *margin* and *slack rescaling*.

While slack rescaling often produces more accurate predictors, margin rescaling has been far more popular due to its better computational requirements. In particular, both formulations require optimizing over the output space, but while margin rescaling preserves the structure of the score and error functions, the slack-rescaling does not. This results in harder inference problems during training. To address this challenge, Sarawagi and Gupta [11] propose a method to reduce the problem of slack rescaling to a series of modified margin rescaling problems. They show that their method outperforms margin rescaling in several domains. However, there are two main caveats in their approach. First, the optimization is only heuristic, that is, it is not guaranteed to solve the slack rescaling objective exactly. Second, their method is specific to the cutting plane training algorithm and does not easily extend to stochastic algorithms. More recently, Bauer et al. [2] proposed an elegant dynamic programming approach to the slack rescaling optimization problem. However, their formulation is restricted to sequence labeling and hamming error, and does not apply to more general structures.

In this paper we propose an efficient method for solving the optimization problem arising from the slack rescaling formulation. Similar to Sarawagi and Gupta [11] our method reduces finding the most violated label in slack rescaling to a series of margin rescaling problems. However, in contrast to their approach, our approach can be easily used with training algorithms like stochastic gradient descent (SGD) [10] and block Frank-Wolfe (FW) [7], which often scale much better than cutting plane. We first propose a very simple approach that minimizes an upper bound on the slack rescaling objective function, and only requires access to a margin rescaling oracle. This formulation is quite general and can be used with any error function and model structure, and many training algorithms such as cutting plane, SGD and FW. However, this method is not guaranteed to always find the most violating label. Indeed, we show that always finding the most violating label for a slack-rescaled formulation is impossible using only a margin rescaling oracle. To address this, we suggest using a modified oracle, that is typically as easy to implement as margin rescaling, and present a more sophisticated algorithm which solves the slack rescaling formulation exactly, and also enjoys good approximation guarantees after a small number of it-

Appearing in Proceedings of the 19^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2016, Cadiz, Spain. JMLR: W&CP volume 51.

erations. We demonstrate empirically that our algorithm outperforms existing baselines on several real-world applications, including hierarchical and multi-label classification.

2 Problem Formulation

In this section we review the basics of structured output prediction and describe the relevant training objectives. In structured output prediction the task is to map data instances x to a set of output labels $y \in \mathcal{Y}$. Structured SVMs use a linear discriminant mapping of the form $y(x; w) = \operatorname{argmax}_{y \in \mathcal{Y}} w^\top \phi(x, y)$, where $\phi(x, y) \in \mathbb{R}^d$ is a feature function relating input-output pairs, and $w \in \mathbb{R}^d$ is a corresponding vector of weights. Our interest is in the supervised learning setting, where w is learned from training data $\{x_i, y_i\}_{i=1}^n$ by minimizing the empirical risk. The prediction quality is measured by an error function $L(y, y_i) \geq 0$ which determines how bad it is to predict y when the ground-truth is in fact y_i.

Since optimizing $L(y, y_i)$ directly is hard due to its complicated dependence on w, several alternative formulations minimize a convex upper bound instead. Structural SVM is an elegant max-margin approach which uses a structured hinge loss surrogate [15, 14]. Two popular surrogates are margin and slack rescaling. In particular, denoting the model score by $f(y) = w^\top \phi(x, y)$ (we omit the dependence on x and w to simplify notation), the margin rescaling training objective is given by:

$$\begin{aligned} \min_{w,\xi} \ & \frac{C}{2}\|w\|_2^2 + \frac{1}{n}\sum_i \xi_i & (1) \\ \text{s.t.} \ & f(y_i) - f(y) \geq L(y, y_i) - \xi_i \qquad \forall i, y \neq y_i \\ & \xi_i \geq 0 \qquad \forall i \end{aligned}$$

where C is the regularization constant. Similarly, the slack rescaling formulation scales the slack variables by the error term:

$$\begin{aligned} \min_{w,\xi} \ & \frac{C}{2}\|w\|_2^2 + \frac{1}{n}\sum_i \xi_i & (2) \\ \text{s.t.} \ & f(y_i) - f(y) \geq 1 - \frac{\xi_i}{L(y, y_i)} \qquad \forall i, y \neq y_i \\ & \xi_i \geq 0 \qquad \forall i \end{aligned}$$

Intuitively, both formulations seek to find a w which assigns high scores to the ground-truth compared to the other possible labellings. When y is very different than the true y_i (L is large) then the difference in scores should also be larger. There is, however, an important difference between the two forms. In margin rescaling, high loss can occur for labellings with high error even though they are already classified correctly with a margin. This may divert training from the interesting labellings where the classifier errs, especially when L can take large values, as in the common case of hamming error. In contrast, in slack rescaling labellings that are classified with a margin incur no loss. Another difference between the two formulations is that the slack rescaling loss is invariant to scaling of the error term, while in margin rescaling such scaling changes the meaning of the features ϕ.

In many cases it is easier to optimize an unconstrained problem. In our case it is straightforward to write (1) and (2) in an unconstrained form:

$$\min_w \tfrac{C}{2}\|w\|_2^2 + \tfrac{1}{n}\sum_i \max_{y \in \mathcal{Y}} \left(L(y, y_i) + f(y) - f(y_i)\right) \quad (3)$$

$$\min_w \tfrac{C}{2}\|w\|_2^2 + \tfrac{1}{n}\sum_i \max_{y \in \mathcal{Y}} L(y, y_i)\left(1 + f(y) - f(y_i)\right) \quad (4)$$

Most of the existing training algorithms for structural SVM require solving the maximization-over-labellings problems in (4) and (3):

$$\text{Margin rescaling} : \operatorname*{argmax}_{y \in \mathcal{Y}} L(y, y_i) + f(y) - f(y_i) \quad (5)$$

$$\text{Slack rescaling} : \operatorname*{argmax}_{y \in \mathcal{Y}} L(y, y_i)\left(1 + f(y) - f(y_i)\right) \quad (6)$$

To better understand the difference between margin and slack rescaling we focus on a single training instance i and define the functions: $h(y) = 1 + f(y) - f(y_i)$ and $g(y) = L(y_i, y)$. With these definitions we see that the maximization (5) for margin rescaling is $\max_{y \in \mathcal{Y}} h(y) + g(y)$, while the maximization (6) for slack rescaling is $\max_{y \in \mathcal{Y}} h(y)g(y)$. It is now obvious why margin rescaling is often easier. When the score and error functions h and g *decompose* into a sum of simpler functions, we can exploit that structure in order to solve the maximization efficiently [15, 14, 5]. In contrast, the slack rescaling score does not decompose even when both h and g do. What we show, then, is how to solve problems of the form $\max_y h(y)g(y)$, and thus the maximization (6), having access only to an oracle for additive problems of the form $\max_y h(y) + \lambda g(y)$.

That is, we assume that we have access to a procedure, referred to as the λ-*oracle*, which can efficiently solve the problem:

$$y_\lambda = \mathcal{O}(\lambda) = \operatorname*{argmax}_{y \in \mathcal{Y}} \mathcal{L}_\lambda(y) \quad (7)$$

where $\mathcal{L}_\lambda(y) = h(y) + \lambda g(y)$. This problem is just a rescaling of (5). E.g., for linear responses it is obtained by scaling the weight vector by $1/\lambda$. If we can handle margin rescaling efficiently we can most likely implement the λ-oracle efficiently. This is also the oracle used by Sarawagi and Gupta [11]. In Section 4, we show how to obtain a solution to the slack-rescaling problem (6) using such a λ-oracle. Our method can be used as a subroutine in a variety of training algorithms, and we demonstrate that it is more

scalable than previous methods. However, we also show that this approach is limited, since no procedure that only has access to a λ-oracle can guarantee the quality of its solution, no matter how much time it is allowed to run.

Therefore, we propose an alternative procedure that can access a more powerful oracle, which we call the *constrained λ-oracle*:

$$y_{\lambda,\alpha,\beta} = \mathcal{O}_c(\lambda,\alpha,\beta) = \max_{y\in\mathcal{Y},\ \alpha h(y)>g(y),\ \beta h(y)\leq g(y)} \mathcal{L}_\lambda(y), \tag{8}$$

where $\alpha, \beta \in \mathbb{R}$. This oracle is similar to the λ-oracle, but can additionally handle linear constraints on the values $h(y)$ and $g(y)$. In the sequel we show that in many interesting cases this oracle is not more computationally expensive than the basic one. For example, when the λ-oracle is implemented as a linear program (LP), the additional constraints are simply added to the LP formulation and do not complicate the problem significantly. Before presenting our algorithms for optimizing (6), we first review the training framework in the next section.

3 Optimization for Slack Rescaling

In this section we briefly survey cutting plane and stochastic gradient descent optimization for the slack rescaled objective (2) and (4). This will be helpful in understanding the difference between our approach and that of prior work on the slack rescaled objective by Sarawagi and Gupta [11].

The cutting plane algorithm was proposed for solving the structural SVM formulation in [15, 6]. This algorithm has also been used in previous work on slack rescaling optimization [11, 2]. The difficulty in optimizing (2) stems from the number of constraints, which is equal to the size of the output space $\mathcal{Y}$ (for each training instance). The cutting plane method maintains a small set of constraints and solves the optimization problem only over that set. At each iteration the active set of constraints is augmented with new violated constraints, and it can be shown that not too many such constraints need to be added for a good solution to be found [6]. The main computational bottleneck here is to find a violating constraint at each iteration, which is challenging since it requires searching over the output space for some violating labeling y.

Relying on this framework, Sarawagi and Gupta [11] use the formulation in (2) and rewrite the constraints as:

$$1 + f(y) - f(y_i) - \frac{\xi_i}{L(y,y_i)} \leq 0 \quad \forall i, y \neq y_i \tag{9}$$

In our notation, to find a violated constraint they aim at solving the problem:

$$\operatorname*{argmax}_{y\in\mathcal{Y}'} \left(h(y) - \frac{\xi_i}{g(y)} \right) \tag{10}$$

where $\mathcal{Y}' = \{y | y \in \mathcal{Y}, h(y) > 0, y \neq y_i\}$. They suggest to minimize a convex upper bound of (10) which stems from the convex conjugate function of $\frac{\xi_i}{g(y)}$:

$$\begin{aligned} &\max_{y\in\mathcal{Y}'} h(y) - \frac{\xi_i}{g(y)} = \max_{y\in\mathcal{Y}'} \min_{\lambda\geq 0} \left(h(y) + \lambda g(y) - 2\sqrt{\xi_i \lambda} \right) \\ &\leq \min_{\lambda\geq 0} \max_{y\in\mathcal{Y}'} F'(\lambda, y) = \min_{\lambda\geq 0} \max_{y\in\mathcal{Y}'} F'(\lambda, y) = \min_{\lambda\geq 0} F(\lambda) \end{aligned} \tag{11}$$

where $F(\lambda) = \max_{y\in\mathcal{Y}'} F'(\lambda, y) = \max_{y\in\mathcal{Y}'} h(y) + \lambda g(y) - 2\sqrt{\xi_i \lambda}$. Since $F(\lambda)$ is a convex function, (11) can be solved by a simple search method such as golden search over λ [11].

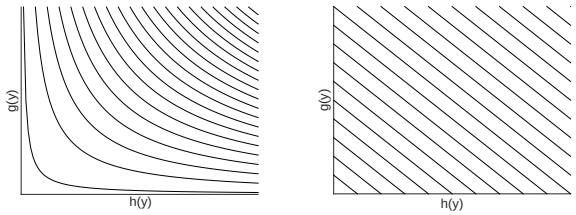
Although this approach is suitable for the cutting plane algorithm, unfortunately it cannot be easily extended to other training algorithms. In particular, $F'(\lambda, y)$ is defined in terms of slack variables ξ_i, which ties it to the constrained form (2). On the other hand, algorithms such as stochastic gradient descent (SGD) [10, 12], stochastic dual coordinate ascent (SDCA) [13], or block-coordinate Frank-Wolfe (FW) [7], all optimize the unconstrained objective form (4). These methods are typically preferable in the large scale setting, since they have very low per-iteration cost, handling a single example at a time, with the same overall iteration complexity as cutting plane methods. In contrast, the cutting plane algorithm considers the entire training set at each iteration, so the method does not scale well to large problems. Since our goal in this work is to handle large datasets, we would like to be able to use the stochastic methods mentioned above, working on the unconstrained formulation (4). The update in these algorithms requires solving the maximization problem (6), which is the goal of Section 4. Note that solving (6) also allows using a cutting plane method if desired.

4 Algorithms

In this section we present our main contribution, a framework for solving the maximization problem (6), which we write as:

$$\max_y \Phi(y) := \max_y h(y) g(y) \tag{12}$$

We describe two new algorithms to solve this problem using access to the λ-oracle, which have several advantages over previous approaches. However, we also show that any algorithm which uses only the λ-oracle cannot always recover an optimal solution. Therefore, in Section 4.5 we proposed an improved algorithm which requires access to an augmented λ-oracle that can also handle linear constraints.



(a) Φ contour. (b) λ contour.

Figure 1: Contour of the two functions considered in $\mathbb{R}^2$. Φ contour is the contour of the objective function, and λ contour is the contour used by the oracle.

4.1 Binary search

We first present a binary search algorithm similar to the one proposed by Sarawagi and Gupta [11], but with one main difference. Our algorithm can be easily used with training methods that optimize the unconstrained objective (4), and can therefore be used for SGD, SDCA and FW. The algorithm minimizes a convex upper bound on Φ without slack variable ξ_i. The algorithm is based on the following lemma (details and proofs are in Appendix A).

Lemma 1. *Let* $\bar{F}(\lambda) = \frac{1}{4}\max_{y\in\mathcal{Y}^+}\left(\frac{1}{\lambda}h(y)+\lambda g(y)\right)^2$, *then*

$$\max_{y\in\mathcal{Y}}\Phi(y) \le \min_{\lambda>0}\bar{F}(\lambda)$$

and $\bar{F}(\lambda)$ *is a convex function in* λ.

Rather than minimizing this upper bound, we next present an algorithm that aims to optimize $\Phi(y)$ in a more direct manner, using a geometrical interpretation of mapping labels into $\mathbb{R}^2$.

4.2 Geometrical Interpretation of λ-oracle search

To understand the problem better and motivate our approach, it is useful to consider the following geometrical interpretation of (12): we map each labeling y to a vector $\vec{y} = [h(y)\ g(y)] \in \mathbb{R}^2$. Let $\vec{\mathcal{Y}} = \{\vec{y} \in \mathbb{R}^2 | y \in \mathcal{Y}\}$ be the set of all mapped labels. The maximization (12) reduces to the problem: given a set of points $\vec{\mathcal{Y}} \subset \mathbb{R}^2$, maximize the product of their coordinates $\vec{y^*} = \operatorname{argmax}_{\vec{y}\in\vec{\mathcal{Y}}}[\vec{y}]_1 \cdot [\vec{y}]_2$.

The contours of our objective function $\vec{\Phi}(\vec{y}) = [\vec{y}]_1 \cdot [\vec{y}]_2$ are then hyperbolas. We would like to maximize this function by repeatedly finding points that maximize linear objectives of the form $\vec{\mathcal{L}}_\lambda(\vec{y}) = [\vec{y}]_1 + \lambda[\vec{y}]_2$, whose contours form lines in the plane (see Figure 1). An example of mapping labels into $\mathbb{R}^2$ is shown in Appendix B.

The importance of the $\mathbb{R}^2$ mapping is that each y_λ revealed by the λ-oracle shows that the optimal y^* can only reside in a small slice of the plane. See figure 2.

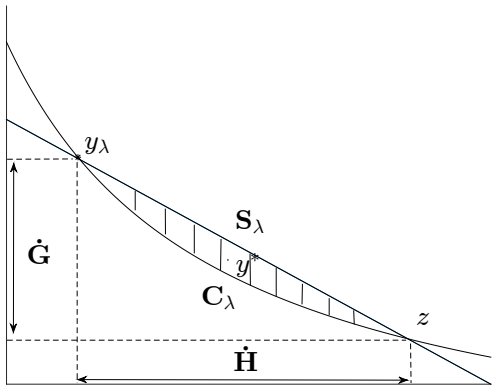


Figure 2: Geometric interpretation of the λ-oracle: $\vec{y^*}$ must reside in the marked area between the upper bound S_λ and the lower bound C_λ. It follows that $h(y^*)$ and $g(y^*)$ reside in a simple segment $\dot{H}$ and $\dot{G}$ respectively.

Lemma 2. *Let* S_λ *be a line through* $\vec{y}_\lambda$ *and* $\vec{z} = [\lambda[\vec{y}_\lambda]_2, \frac{1}{\lambda}[\vec{y}_\lambda]_1]$, *and let* $C_\lambda = \{\vec{y} \in \mathbb{R}^2 | [\vec{y}]_1 \cdot [\vec{y}]_2 = \vec{\Phi}(\vec{y}_\lambda)\}$ *be the hyperbola through* $\vec{y}_\lambda$. *Then,* $\vec{y^*}$ *is on or below line* S_λ, *and* $\vec{y^*}$ *is on or above hyperbola* C_λ.

Proof. If there exists a $\vec{y} \in \vec{\mathcal{Y}}$ which is above S_λ, it contradicts the fact that $\vec{y}_\lambda$ is the argmax point for function $\vec{\mathcal{L}}_\lambda$. And the second argument follows from $\vec{y^*}$ being the argmax label w.r.t. $\vec{\Phi}$, and the area above C_λ corresponds to points whose $\vec{\Phi}$ value is greater than $\vec{y_\lambda}$. □

It follows that $h(y^*)$ and $g(y^*)$ must each reside in a segment:

Lemma 3. *Let* $\dot{H} = [\min([\vec{y_\lambda}]_1, [\vec{z}]_1), \max([\vec{y_\lambda}]_1, [\vec{z}]_1)]$ *and* $\dot{G} = [\min([\vec{y_\lambda}]_2, [\vec{z}]_2), \max([\vec{y_\lambda}]_2, [\vec{z}]_2)]$. *Then,*

$$h(y^*) \in \dot{H}, \qquad g(y^*) \in \dot{G}$$

Proof. This follows from the fact that S_λ and C_λ intersects at two points, $\vec{y_\lambda}$ and $\vec{z}$, and the boundaries, S_λ and C_λ, are strictly decreasing functions in $\mathbb{R}^2$. □

4.3 Bisecting search

In this section, we propose a search algorithm which is based on the previous geometric interpretation. Similar to the binary search, our method also relies on the basic λ-oracle.

We first give an overview of the algorithm. We maintain a set of possible value ranges $\lambda^* = \operatorname{argmax}_{\lambda>0}\Phi(y_\lambda)$, $h(\lambda^*)$, and $g(\lambda^*)$ as L, H, and G, respectively; all initialized as $\mathbb{R}$. First, for each y_λ returned by the oracle, we take an intersection of G and H with a segment of possible values of $h(y)$ and $g(y)$, respectively, using Lemmas 2 and 3.

Second, we reduce the space L of potential λ's based on the following Lemma (proved in Appendix C).

Lemma 4. *$h(y_\lambda)$ is a non-increasing function of λ, and $g(y_\lambda)$ is a non-decreasing function of λ.*

Thus, we can discard $\{\lambda' | \lambda' > \lambda\}$ if $h(y^*_\lambda) > h(y_\lambda)$ or $\{\lambda' | \lambda' < \lambda\}$ otherwise from L. Next, we pick $\lambda \in L$ in the middle, and query y_λ. The algorithm continues until at least one of L, H, and G is empty. This *bisecting search* procedure is given in Algorithm 1.

Similar to the binary search from the previous section, this algorithm can be used with training methods like SGD and SDCA, as well as the cutting-plane algorithm. However, this approach has several advantages compared to the binary search. First, the binary search needs explicit upper and lower bounds on λ, thus it has to search the entire λ space [11]. However, the bisecting search can directly start from any λ without an initial range, and for instance, this can be used to warm-start from the optimal λ in the previous iteration. Furthermore, we point out that since the search space of h and g is also bisected, the procedure can terminate early if either of them becomes empty.

Finally, in Appendix D we propose two improvements that can be applied to either the binary search or the bisecting search. Specifically, we first provide a simple stopping criterion that can be used to terminate the search when the current solution y_{λ_t} will not further improve. Second, we show how to obtain a bound on the suboptimality of the current solution, which can give some guarantee on its quality.

So far we have used the λ-oracle as a basic subroutine in our search algorithms. Unfortunately, as we show next, this approach is limited as we cannot guarantee finding the optimal solution y^*, even with unlimited number of calls to the λ-oracle. This is somewhat distressing since with unlimited computation we can find the optimum of (6) by enumerating all y's.

4.4 Limitation of the λ-oracle

Until now, we used only the λ-oracle to search for Φ^* without directly accessing the functions h and g. We now show that this approach, searching Φ^* with only a λ-oracle, is very limited: even with an unlimited number of queries, the search cannot be exact and might return a trivial solution in the worst case (see Appendix E for proof).

Theorem 1. *Let $\hat{H} = \max_y h(y)$ and $\hat{G} = \max_y g(y)$. For any $\epsilon > 0$, there exists a problem with 3 labels such that for any $\lambda \geq 0$, $\Phi(y_\lambda) = \min_{y \in \mathcal{Y}} \Phi(y) < \epsilon$, while $\Phi(y^*) = \frac{1}{4}\hat{H}\hat{G}$.*

Theorem 1 shows that any search algorithm that can access the function only through λ-oracle, including the method of Sarawagi and Gupta [11] and both methods presented above, cannot be guaranteed to find a label optimizing $\Phi(y)$, even approximately, and even with unlimited accesses to the oracle. This problem calls for a more powerful oracle.

Algorithm 1 Bisecting search

procedure BISECTING(λ_0)
Input: Initial λ for the search $\lambda_0 \in \mathbb{R}_+$
Output: $\hat{y} \in \mathcal{Y}$.
Initialize: $H = G = L = \mathbb{R}_+, \lambda = \lambda_0, \hat{\Phi} = 0$.
  while $H \neq \emptyset$ and $G \neq \emptyset$ **do**
    $y' \leftarrow \mathcal{O}(\lambda)$
    $u \leftarrow [h(y') \ \lambda g(y')], v \leftarrow [g(y') \ \frac{1}{\lambda} h(y')]$
    $H \leftarrow H \cap \{h' | \min u \leq h' \leq \max u\}$ ▷ Update
    $G \leftarrow G \cap \{g' | \min v \leq g' \leq \max v\}$
    if $v_1 \leq v_2$ **then** ▷ Increase λ
      $L \leftarrow L \cap \{\lambda' \in \mathbb{R} | \lambda' \geq \lambda\}$
    else ▷ Decrease λ
      $L \leftarrow L \cap \{\lambda' \in \mathbb{R} | \lambda' \leq \lambda\}$
    $\lambda \leftarrow \frac{1}{2}(\min L + \max L)$
    if $h(y')g(y') \geq \hat{\Phi}$ **then**
      $\hat{y} \leftarrow y', \hat{\Phi} \leftarrow h(y')g(y')$.

4.5 Angular search with the constrained-λ-oracle

The *constrained λ-oracle* defined in (8) has two inequality constraints to restrict the search space. Using this modified oracle, we can present an algorithm that *is* guranteed to find the most violating constraint, as captured by the following theorem, proved in Appendix F:

Theorem 2. *Angular search described in Algorithm 2 finds the optimum $y^* = \operatorname{argmax}_{y \in \mathcal{Y}} \Phi(y)$ using at most $t = 2M + 1$ iteration where $M = |\mathcal{Y}|$ is the number of the labels.*

This is already an improvement over the previous methods, as at least we are guaranteed to return the actual most violating label. However, it is still disappointing since the number of iterations, and thus number of oracle accesses might actually be larger than the number of labels. This defies the whole point, since we might as well just enumerate over all M possible labels. Unfortunately, even with a constrained oracle, this is almost the best we can hope for. In fact, even if we allow additional linear constraints, we might still need M oracle accesses, as indicated by the following theorem, proved in Appendix E.

Theorem 3. *Any search algorithm accessing labels only through a λ-oracle with any number of linear constraints cannot find y^* using less than M iterations in the worst case, where $M = |\mathcal{Y}|$ is the number of labels.*

Fortunately, even though we cannot guarantee optimizing $\Phi(y)$ exactly using a small number of oracle accesses, we

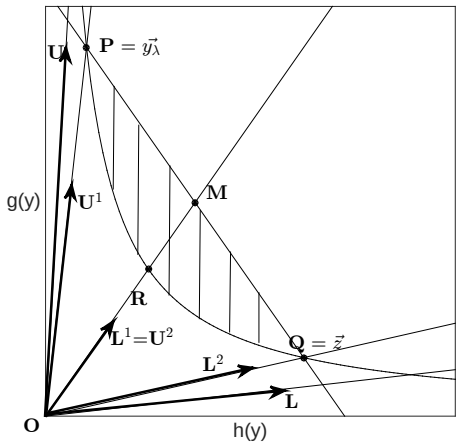


Figure 3: Split procedure.

can at least do so approximately. This can be achieved by Algorithm 2, as the next theorem states (proved in Appendix F).

Theorem 4. *In angular search, described in Algorithm 2, at iteration t,*

$$\Phi(\hat{y}^t) \geq \Phi(y^*)(v_1^{-\frac{4}{t+1}})$$

where $\hat{y}^t = \operatorname{argmax}_t y^t$ is the optimum up to t, $v_1 = \max\left\{\frac{\lambda_0}{\partial(\vec{y_1})}, \frac{\partial(\vec{y_1})}{\lambda_0}\right\}$, λ_0 is the initial λ used, and y_1 is the first label returned by the constrained λ-oracle.

We use $\partial(a) = \frac{a_2}{a_1}$ to denote the slope of a vector in $\mathbb{R}^2$.

With proper initialization, we get the following runtime guarantee:

Theorem 5. *Assuming $\Phi(y^*) > \phi$, angular search described in algorithm 2 with $\lambda_0 = \frac{\hat{G}}{\hat{H}}, \alpha_0 = \frac{\hat{G}^2}{\phi}, \beta_0 = \frac{\phi}{\hat{H}^2}$, finds an ϵ-optimal solution, $\Phi(y) \geq (1-\epsilon)\Phi(y^*)$, in T queries and $O(T)$ operations, where $T = 4\log\left(\frac{\hat{G}\hat{H}}{\phi}\right) \cdot \frac{1}{\epsilon}$, and δ-optimal solution, $\Phi(y) \geq \Phi(y^*) - \delta$, in T' queries and $O(T')$ operations, where $T' = 4\log\left(\frac{\hat{G}\hat{H}}{\phi}\right) \cdot \frac{\Phi(y^*)}{\delta}$.*

Here we give an overview of the algorithm with an illustration in Figure 3. The constrained λ-oracle restricts the search space, and this restriction can be illustrated as a linear upper bound U and a lower bound L. The search is initialized with the entire right angle: $U = [0\ \infty]$ and $L = [\infty\ 0]$, and maintains that $\vec{y^*}$ is always between U and L. The constrained λ-oracle is used with U, L and a certain λ to reduce the potential area where $\vec{y^*}$ can reside.

Algorithm 2 Angular search

procedure ANGULARSEARCH(λ_0, T)
Input: $\lambda_0 \in \mathbb{R}_+$, and maximum iteration $T \in \mathbb{R}_+$
Output: $\hat{y} \in \mathcal{Y}$.
Initialize: $\alpha_0 = \infty, \beta_0 = 0$, Empty queue $\mathcal{Q}$, $\hat{y} = \emptyset.\lambda \leftarrow \lambda_0$
  ADD($\mathcal{Q}, (\alpha, \beta, 0)$)
  while $\mathcal{Q} \neq \emptyset$ **do**
    $(\alpha, \beta, s) \leftarrow$ Dequeue($\mathcal{Q}$)
    if $\beta \neq 0$ **then**
      $\lambda \leftarrow \frac{1}{\sqrt{\alpha\beta}}$
    if $s = 0$ **then**
      $y \leftarrow \mathcal{O}_c(\lambda, \alpha, \beta)$
    else
      $y \leftarrow \underline{\mathcal{O}}_c(\lambda, \alpha, \beta)$
    if $\Phi(y) > \Phi(\hat{y})$ **then**
      $\hat{y} \leftarrow y$
    if $y \neq \emptyset$ **then**
      $z \leftarrow [h(y)\ g(y)], z' \leftarrow [\lambda g(y)\ \frac{1}{\lambda}h(y)]$
      $r \leftarrow \left[\sqrt{\lambda h(y)g(y)}\ \sqrt{\frac{1}{\lambda}h(y)g(y)}\ \right]$
      if $z_1 = z'_1$ **then**
        return y
      else if $\partial(z) > \partial(z')$ **then**
        $K^1 \leftarrow (\partial(z), \partial(r), 1)$
        $K^2 \leftarrow (\partial(r), \partial(z'), 0)$
      else
        $K^1 \leftarrow (\partial(z'), \partial(r), 1)$
        $K^2 \leftarrow (\partial(r), \partial(z), 0)$
      ADD($\mathcal{Q}, K^1$) .ADD($\mathcal{Q}, K^2$)
    $t \leftarrow t + 1$
    if $t = T$ **then** ▷ maximum iteration reached
      return $\hat{y}$

Specifically, the search space is reduced using an angle defined by $U = \overline{OP}$ and $L = \overline{OQ}$. In the next iteration, the constrained λ-oracle is invoked with $U^1 = \overline{OP}$ and $L^1 = \overline{OM}$, and also with $U^2 = \overline{OM}$ and $L^2 = \overline{OQ}$. Intuitively, each such query shrinks the search space, and as the search space shrinks, the suboptimaly bound improves. This process is continued until the remaining search space is empty. The angular search algorithm defines the optimal λ and values to be passed to the constrained λ-oracle.

In Algorithm 2 each angle is dequeued, split, and enqueued recursively. Each angle maintains its upper bound from the previous iterations and stops splitting itself and terminates if it is ensured that there exists no label with larger Φ value within the angle. When the oracle reveals a label with $\Phi(y_\lambda) = c$, we can safely discard all area corresponding to $\{\vec{y}|\Phi(\vec{y}) \leq c\}$. This works as a global constraint which shrinks the search space. Therefore, acquiring a label with high Φ value in the early stages facilitate

	Angular	Bisecting	Sarawagi
Yeast (N=160)			
Success	22.4%	16.5%	16.4%
Queries per search	3.8	10.3	43.2
Average time (ms)	4.7	3.6	18.5
RCV1 (N=160)			
Success	25.6%	18.2%	18%
Queries per search	4.8	12.7	49
Average time (ms)	4.4	5.2	20.9

Table 1: Comparison of the search algorithm.

Yeast

	Acc	Label loss	MiF1	MaF1
Slack	.54	.205	.661	.651
Margin	.545	.204	.666	.654

RCV1

	Acc	Label loss	MiF1	MaF1
Slack	.676	.023	.755	.747
Margin	.662	.023	.753	.742

Table 2: Results on Multi-label Dataset with Markov Random Field.

convergence. Thus, it is suggested to use a priority queue, and dequeue the angle with the highest upper bound on Φ. A similar strategy is to have a *label cache*, the subset of previous most violated labels, denoted as $\mathcal{C}$. With the label cache, we can discard a large part of the search space $\{\vec{y}|\Phi(\vec{y}) \leq \max_{y' \in \mathcal{C}} \Phi(\vec{y'})\}$ immediately. Algorithm 2 also uses the constrained λ-oracle to avoid returning previously found labels. Finally, for λ_0, we suggest to use $\lambda_0 = \frac{\hat{H}}{\hat{G}}$, with $\hat{H}$ calculated from the current weights w.

See Appendix G for an illustration of the angular search.

5 Experiments

In this section we compare the behavior of various search algorithms on standard benchmark datasets, and its effect on the learning algorithm. Specifically, we show that Angular search with SGD is not only much faster than the other alternatives, but also results in an accurate predictor which outperforms both margin rescaling and other slack rescaling methods.

Unlike the simple structure used in [2], we apply our approach to complicated structures. Specifically, we experiment with multi-label dataset modeled by a Markov Random Field with pairwise potentials as in [5]. Since the inference of margin rescaling is NP-hard in this case, we rely on linear programming relaxation to compute the λ-oracle. Note that this complicates the problem, since the number of labels becomes even larger with additional fractional solutions. Also notice that all of our results above apply with mild modifications to this harder setting. Two standard benchmark multi-label datasets, Yeast [4] (14 labels) and RCV1 [8], are tested. For RCV1 we reduce the data to the 50 most frequent labels. For angular search, we stop the search whenever $\Phi(\hat{y}) > 0.999 \cdot \Phi(y^*)$ holds, to avoid numerical issues.

5.1 Comparison of the search algorithms

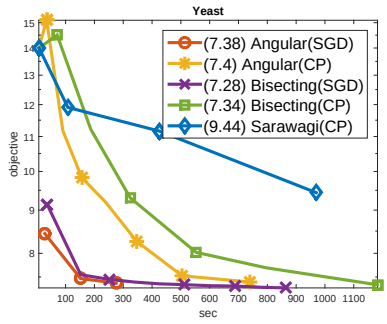
We first focus on the search method and run cutting-plain optimization, where at each iteration we call all three algorithms to find the most violating label: Angular search, Bisecting search, and Sarawagi and Gupta's search (Sarawagi) [11]. The cutting-plane algorithm calls the search procedures to find the most violating label $\hat{y}$, and adds it to the active set if the violation is larger than some margin ϵ, i.e., $\Delta(\hat{y}, y_i)(1 + f(\hat{y}) - f(y_i)) > \xi_i + \epsilon$. We use the result of Angular search for the actual update and repeat. Table 1 compares the performance of the search in terms of the time spend, the number of oracle queries, and the success rate of finding the most violating label. Success is measured by the percentage in which the search algorithm finds such a violating label. As expected by Theorem 1, Bisecting and Sarawagi's search miss the violating label in cases where Angular search successfully finds one. For RCV1 dataset, not only is Angular search more accurate, but it also uses about 2.6 times less oracle queries than Bisecting and 10.1 times less queries than Sarawagi's search. As for runtime, Angular search is 1.18 times faster than Bisecting search, and 4.7 times faster than Sarawagi's algorithm.

In order to understand the effect of the search procedure on the overall performance, we next compare combinations of learning and search algorithms. In figure 4 we compare convergence rate and accuracy for the different combinations. These show that Angular search with SGD converges much faster than other schemes. Additional plots showing convergence w.r.t. the number of queries and iterations can be found in Appendix I.

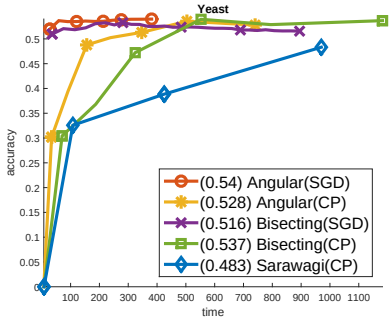
Table 2 shows a comparison of predictive performance for the multi-label datasets. In this case margin and slack rescaling perform similarly, with a slight advantage for slack rescaling on RCV1.

5.2 Hierarchical Multi-label Classification

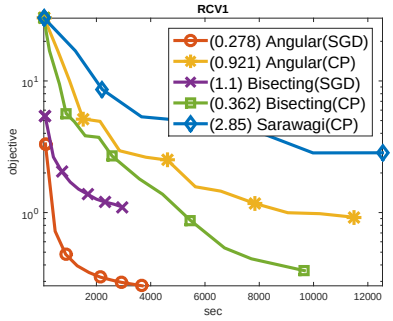
We further run experiments on a hierarchical multi-label classification problem [3]. In hierarchical multi-label classification, each label y is a leaf node in a given graph, and it shares ancestor nodes. It can be described as a graphical model where a potential of a multi-label $\mathcal{Y} = \{y_1, \ldots, y_k\}$ is the sum of all potentials of its ancestors, i.e., $\Phi(Y) = \sum_{n' \in \bigcup_{n \in Y} Anc(n)} \Phi(n)$. We extracted 1500 instances with dimensionality 17944 and a graph structure of 156 nodes



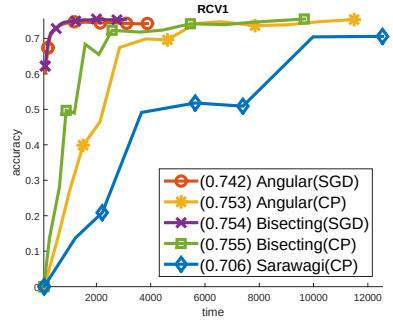
(a) Yeast Convergence



(b) Yeast Accuracy



(c) RCV1 Convergence



(d) RCV1 Accuracy

Figure 4: Convergence and accuracy of various learning and search algorithms.

with 123 labels from SWIKI-2011. SWIKI-2011 is a multi-label dataset of Wikipedia pages with a DAG structure from LSHTC competition.[1] We used 750 instances as training set, 250 instances as holdout set, and 500 instances as test set. The hamming distance of label vectors including inner nodes is used as label loss. Unfortunately, the cutting-plane approach failed to scale up to the large problem size, however, using our approach we show that slack rescaling in such large label structure is tractable and outperforms margin rescaling. See Table 3.

	Acc	Label loss	MiF1	MaF1
Slack	.3798	.0105	.3917	.3880
Margin	.3327	.0110	.3394	.3378

Table 3: Result on hierarchical multi-label dataset

6 Summary

As we saw in our experiments, and has also been previously noted, slack rescaling is often beneficial compared to margin rescaling in terms of predictive performance. However, the margin-rescaled argmax (5) is often much easier computationally due to its additive form. Here we show how an oracle for solving an argmax of the form (5), or perhaps a slightly modified form (the constrained-λ oracle), is sufficient for also obtaining exact solutions to the slack-rescaling argmax (6). This allows us to train slack-rescaled SVMs using SGD, obtaining better predictive performance than using margin rescaling. Prior work in this direction provided only approximations [11] or handled only specific models and losses [2], and more significantly, was specific to cutting-plane optimization. In this work we provide a generic method relying on a simple explicitly specified oracle that is guaranteed to be exact and efficient even when the number of labels is infinite. Our approach allows using SGD and is thus more suitable for large scale problems.

References

[1] Bakir, G. H., Hofmann, T., Schölkopf, B., Smola, A. J., Taskar, B., and Vishwanathan, S. V. N. (2007). *Predicting Structured Data*. The MIT Press.

[2] Bauer, A., Gornitz, N., Biegler, F., Muller, K.-R., and Kloft, M. (2014). Efficient algorithms for exact inference in sequence labeling svms. *Neural Networks and Learning Systems, IEEE Transactions on*, 25(5):870–881.

[3] Cai, L. and Hofmann, T. (2004). Hierarchical document categorization with support vector machines. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 78–87. ACM.

[1] http://lshtc.iit.demokritos.gr/

[4] Elisseeff, A. and Weston, J. (2001). A kernel method for multi-labelled classification. In *Advances in neural information processing systems*, pages 681–687.

[5] Finley, T. and Joachims, T. (2008). Training structural svms when exact inference is intractable. In *Proceedings of the 25th international conference on Machine learning*, pages 304–311. ACM.

[6] Joachims, T., Finley, T., and Yu, C.-N. J. (2009). Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59.

[7] Lacoste-Julien, S., Jaggi, M., Schmidt, M., and Pletscher, P. (2013). Block-coordinate frank-wolfe optimization for structural svms. In *ICML 2013 International Conference on Machine Learning*, pages 53–61.

[8] Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. (2004). Rcv1: A new benchmark collection for text categorization research. *The Journal of Machine Learning Research*, 5:361–397.

[9] Nghia, N. D., Chinh, D. D., and Duong, P. C. (1995). Minimizing the product of two discrete convex functions. *ACTA MATHEMATICA VIETNAMICA*, 20(2):265–267.

[10] Ratliff, N., Bagnell, J. A. D., and Zinkevich, M. (2007). (Online) subgradient methods for structured prediction. In *AISTATS*.

[11] Sarawagi, S. and Gupta, R. (2008). Accurate max-margin training for structured output spaces. In *Proceedings of the 25th international conference on Machine learning*, pages 888–895. ACM.

[12] Shalev-Shwartz, S., Singer, Y., Srebro, N., and Cotter, A. (2011). Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30.

[13] Shalev-Shwartz, S. and Zhang, T. (2013). Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. *Mathematical Programming*, pages 1–41.

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

[15] Tsochantaridis, I., Hofmann, T., Joachims, T., and Altun, Y. (2004). Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, page 104. ACM.