The Structured Weighted Violations Perceptron Algorithm

Rotem Dror and **Roi Reichart**
Faculty of Industrial Engineering and Management, Technion, IIT
{rtmdrr@campus|roiri@ie}.technion.ac.il

Abstract

We present the *Structured Weighted Violations Perceptron (SWVP)* algorithm, a new structured prediction algorithm that generalizes the Collins Structured Perceptron (CSP, (Collins, 2002)). Unlike CSP, the update rule of SWVP explicitly exploits the internal structure of the predicted labels. We prove the convergence of SWVP for linearly separable training sets, provide mistake and generalization bounds, and show that in the general case these bounds are tighter than those of the CSP special case. In synthetic data experiments with data drawn from an HMM, various variants of SWVP substantially outperform its CSP special case. SWVP also provides encouraging initial dependency parsing results.

1 Introduction

The structured perceptron ((Collins, 2002), henceforth denoted CSP) is a prominent training algorithm for structured prediction models in NLP, due to its effective parameter estimation and simple implementation. It has been utilized in numerous NLP applications including word segmentation and POS tagging (Zhang and Clark, 2008), dependency parsing (Koo and Collins, 2010; Goldberg and Elhadad, 2010; Martins et al., 2013), semantic parsing (Zettlemoyer and Collins, 2007) and information extraction (Hoffmann et al., 2011; Reichart and Barzilay, 2012), if to name just a few.

Like some training algorithms in structured prediction (e.g. structured SVM (Taskar et al., 2004; Tsochantaridis et al., 2005), MIRA (Crammer and Singer, 2003) and LaSo (Daumé III and Marcu, 2005)), CSP considers in its update rule the difference between *complete* predicted and gold standard labels (Sec. 2). Unlike others (e.g. factored MIRA (McDonald et al., 2005b; McDonald et al., 2005a) and dual-loss based methods (Meshi et al., 2010)) it does not exploit the structure of the predicted label. This may result in valuable information being lost.

Consider, for example, the gold and predicted dependency trees of Figure 1. The substantial difference between the trees may be mostly due to the difference in roots (*are* and *worse*, respectively). Parameter update w.r.t this mistake may thus be more useful than an update w.r.t the complete trees.

In this work we present a new perceptron algorithm with an update rule that exploits the structure of a predicted label when it differs from the gold label (Section 3). Our algorithm is called *The Structured Weighted Violations Perceptron (SWVP)* as its update rule is based on a weighted sum of updates w.r.t *violating assignments* and *non-violating assignments*: assignments to the input example, derived from the predicted label, that score higher (for violations) and lower (for non-violations) than the gold standard label according to the current model.

Our concept of *violating assignment* is based on Huang et al. (2012) that presented a variant of the CSP algorithm where the argmax inference problem is replaced with a violation finding function. Their update rule, however, is identical to that of the CSP algorithm. Importantly, although CSP and the above variant do not exploit the internal structure of the predicted label, they are special cases of SWVP.

In Section 4 we prove that for a linearly separable training set, SWVP converges to a linear separator of the data under certain conditions on the parameters of the algorithm, that are respected by the CSP special case. We further prove mistake and generalization bounds for SWVP, and show that in the general case the SWVP bounds are tighter than the CSP's.

In Section 5 we show that SWVP allows *aggressive* updates, that exploit only violating assignments derived from the predicted label, and more *balanced* updates, that exploit both violating and non-violating assignments. In experiments with synthetic data generated by an HMM, we demonstrate that various SWVP variants substantially outperform CSP training. We also provide initial encouraging dependency parsing results, indicating the potential of SWVP for real world NLP applications.

2 The Collins Structured Perceptron

In structured prediction the task is to find a mapping $f : \mathcal{X} \rightarrow \mathcal{Y}$, where $y \in \mathcal{Y}$ is a structured object rather than a scalar, and a feature mapping $\phi(x, y) : \mathcal{X} \times \mathcal{Y}(x) \rightarrow \mathbb{R}^d$ is given. In this work we denote $\mathcal{Y}(x) = \{y' | y' \in {D_Y}^{L_x}\}$, where L_x, a scalar, is the size of the allowed output sequence for an input x and D_Y is the domain of y'_i for every $i \in \{1, \ldots L_x\}$. [1] Our results, however, hold for the general case of an output space with variable size vectors as well.

The CSP algorithm (Algorithm 1) aims to learn a parameter (or weight) vector $\mathbf{w} \in \mathbb{R}^d$, that separates the training data, i.e. for each training example (x, y) it holds that: $y = \arg\max_{y' \in \mathcal{Y}(x)} \mathbf{w} \cdot \phi(x, y')$. To find such a vector the algorithm iterates over the training set examples and solves the above inference ($argmax$) problem. If the inferred label y^* differs from the gold label y the update $\mathbf{w} = \mathbf{w} + \Delta\phi(x, y, y^*)$ is performed. For linearly separable training data (see definition 4), CSP is proved to converge to a vector $\mathbf{w}$ separating the training data.

Collins and Roark (2004) and Huang et al. (2012) expanded the CSP algorithm by proposing various alternatives to the argmax *inference* problem which is often intractable in structured prediction problems (e.g. in high-order graph-based dependency parsing (McDonald and Pereira, 2006)). The basic idea is replacing the argmax problem with the search for a *violation*: an output label that the model scores higher

[1] In the general case L_x is a set of output sizes, which may be finite or infinite (as in constituency parsing (Collins, 1997)).

Algorithm 1 The Structured Perceptron (CSP)

```
   Input: data $D = \{x^i, y^i\}_{i=1}^n$, feature mapping $\phi$
   Output: parameter vector $\mathbf{w} \in \mathbb{R}^d$
   Define: $\Delta\phi(x, y, z) \triangleq \phi(x, y) - \phi(x, z)$
1: Initialize $\mathbf{w} = 0$.
2: repeat
3:   for each $(x^i, y^i) \in D$ do
4:     $y^* = \arg\max_{y' \in \mathcal{Y}(x^i)} \mathbf{w} \cdot \phi(x^i, y')$
5:     if $y^* \neq y^i$ then
6:       $\mathbf{w} = \mathbf{w} + \Delta\phi(x^i, y^i, y^*)$
7:     end if
8:   end for
9: until Convergence
```

than the gold standard label. The update rule in these CSP variants is, however, identical to the CSP's. We, in contrast, propose a novel update rule that exploits the internal structure of the model's prediction regardless of the way this prediction is generated.

3 The Structured Weighted Violations Perceptron (SWVP)

SWVP exploits the internal structure of a predicted label $y^* \neq y$ for a training example $(x, y) \in D$, by updating the weight vector with respect to substructures of y^*. We start by presenting the fundamental concepts at the basis of our algorithm.

3.1 Basic Concepts

Sub-structure Sets We start with two fundamental definitions: **(1)** An individual *sub-structure* of a structured object (or label) $y \in {D_Y}^{L_x}$, denoted with J, is defined to be a subset of indexes $J \subseteq [L_x]$;[2] and **(2)** A *set of substructures* for a training example (x, y), denoted with JJ_x, is defined as $JJ_x \subseteq 2^{[L_x]}$.

Mixed Assignment We next define the concept of a *mixed assignment*:

Definition 1. *For a training pair (x, y) and a predicted label $y^* \in \mathcal{Y}(x)$, $y^* \neq y$, a **mixed assignment** (MA) vector denoted as $m^J(y^*, y)$ is defined with respect to $J \in JJ_x$ as follows:*

$$m_k^J(y^*, y) = \begin{cases} y_k^* & k \in J \\ y_k & else \end{cases}$$

That is, a mixed assignment is a new label, derived from the predicted label y^*, that is identical to y^* in all indexes in J and to y otherwise. For simplicity we denote $m^J(y^*, y) = m^J$ when the reference y^* and y labels are clear from the context.

[2] We use the notation $[n] = \{1, 2, \ldots n\}$.

Consider, for example, the trees of Figure 1, assuming that the top tree is y, the middle tree is y^* and $J = [2, 5]$.[3] In the $m^J(y^*, y)$ (bottom) tree the heads of all the words are identical to those of the top tree, except for the heads of *mistakes* and of *then*.

Violation The next central concept is that of a violation, originally presented by Huang et al. (2012):

Definition 2. *A triple (x, y, y^*) is said to be a* ***violation*** *with respect to a training example (x, y) and a parameter vector* **w** *if for $y^* \in \mathcal{Y}(x)$ it holds that $y^* \neq y$ and $\mathbf{w} \cdot \Delta\phi(x, y, y^*) \leq 0$.*

The SWVP algorithm distinguishes between MAs that are violations, and ones that are not. For a triplet (x, y, y^*) and a set of substructures $JJ_x \subseteq 2^{[L_x]}$ we provide the following notations:

$$I(y^*, y, JJ_x)^v \ = \{J \in JJ_x | m^J \neq y, \mathbf{w} \cdot \Delta\phi(x, y, m^J) \leq 0\}$$

$$I(y^*, y, JJ_x)^{nv} = \{J \in JJ_x | m^J \neq y, \mathbf{w} \cdot \Delta\phi(x, y, m^J) > 0\}$$

This notation divides the set of substructures into two subsets, one consisting of the substructures that yield violating MAs and one consisting of the substructures that yield non-violating MAs. Here again when the reference label y^* and the set JJ_x are known we denote: $I(y^*, y, JJ_x)^v = I^v$, $I(y^*, y, JJ_x)^{nv} = I^{nv}$ and $I = I^v \cup I^{nv}$.

Weighted Violations The key idea of SWVP is the exploitation of the internal structure of the predicted label in the update rule. For this aim at each iteration we define the set of substructures, JJ_x, and then, for each $J \in JJ_x$, update the parameter vector, **w**, with respect to the mixed assignments, MA^J's. This is a more flexible setup compared to CSP, as we can update with respect to the predicted output (if it is a violation, as is promised if inference is performed via argmax), if we wish to do so, as well as with respect to other mixed assignments.

Naturally, not all mixed assignments are equally important for the update rule. Hence, we weigh the different updates using a weight vector γ. This paper therefore extends the observation of Huang et al. (2012) that perceptron parameter update can be performed w.r.t violations (Section 2), by showing that **w** can actually be updated w.r.t linear combinations of mixed assignments, under certain conditions on the selected weights.

[3]We index the dependency tree words from 1 onwards.

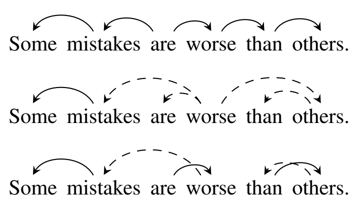
Some mistakes are worse than others.
Some mistakes are worse than others.
Some mistakes are worse than others.

Figure 1: Example parse trees: gold tree (y, top), predicted tree (y^*, middle) with arcs differing from the gold's marked with a dashed line, and $m^J(y^*, y)$ for $J = [2, 5]$ (bottom tree).

3.2 Algorithm

With these definitions we can present the SWVP algorithm (Algorithm 2). SWVP is in fact a family of algorithms differing with respect to two decisions that can be made at each pass over each training example (x, y): the choice of the set JJ_x and the implementation of the SETGAMMA function.

SWVP is very similar to CSP except for in the update rule. Like in CSP, the algorithm iterates over the training data examples and for each example it first predicts a label according to the current parameter vector **w** (inference is discussed in Section 4.2, property 2). The main difference from CSP is in the update rule (lines 6-12). Here, for each substructure in the substructure set, $J \in JJ_x$, the algorithm generates a mixed assignment m^J (lines 7-9). Then, **w** is updated with a weighted sum of the mixed assignments (line 11), unlike in CSP where the update is held w.r.t the predicted assignment only.

The $\gamma(m^J)$ weights assigned to each of the $\Delta\phi(x, y, m^J)$ updates are defined by a SETGAMMA function (line 10). Intuitively, a $\gamma(m^J)$ weight should be higher the more the mixed assignment is assumed to convey useful information that can guide the update of **w** in the right direction. In Section 4 we detail the conditions on SETGAMMA under which SWVP converges, and in Section 5 we describe various SETGAMMA implementations.

Going back to the example of Figure 1, one would assume (Sec. 1) that the head word prediction for *worse* is pivotal to the substantial difference between the two top trees (UAS of 0.2). CSP does not directly exploit this observation as it only updates its parameter vector with respect to the differences between complete assignments: $\mathbf{w} = \mathbf{w} + \Delta\phi(x, y, z)$.

In contrast, SWVP can exploit this observation in various ways. For example, it can generate a mixed assignment for each of the erroneous arcs where all other words are assigned their correct arc (according to the gold tree) except for that specific arc which is kept as in the bottom tree. Then, higher weights can be assigned to errors that seem more central than others. We elaborate on this in the next two sections.

Algorithm 2 The Structured Weighted Violations Perceptron

```
    Input: data $D = \{x^i, y^i\}_{i=1}^n$, feature mapping $\phi$
    Output: parameter vector $\mathbf{w} \in \mathbb{R}^d$
    Define: $\Delta\phi(x, y, z) \triangleq \phi(x, y) - \phi(x, z)$
 1: Initialize $\mathbf{w} = 0$.
 2: repeat
 3:   for each $(x^i, y^i) \in D$ do
 4:     $y^* = \arg\max_{y' \in \mathcal{Y}(x^i)} \mathbf{w} \cdot \phi(x^i, y')$
 5:     if $y^* \neq y^i$ then
 6:       Define: $JJ_{x^i} \subseteq 2^{[L_{x^i}]}$
 7:       for $J \in JJ_{x^i}$ do
 8:         Define: $m^J$ s.t. $m_k^J = \begin{cases} y_k^* & k \in J \\ y_k^i & else \end{cases}$
 9:       end for
10:       $\gamma = \textsc{SetGamma}()$
11:       $\mathbf{w} = \mathbf{w} + \sum_{J \in I^v \cup I^{nv}} \gamma(m^J)\Delta\phi(x^i, y^i, m^J)$
12:     end if
13:   end for
14: until Convergence
```

4 Theory

We start this section with the convergence conditions on the γ vector which weighs the mixed assignment updates in the SWVP update rule (line 11). Then, using these conditions, we describe the relation between the SWVP and the CSP algorithms. After that, we prove the convergence of SWVP and analyse the derived properties of the algorithm.

γ Selection Conditions Our main observation in this section is that SWVP converges under two conditions: **(a)** the training set D is linearly separable; and **(b)** for any parameter vector $\mathbf{w}$ achievable by the algorithm, there exists $(x, y) \in D$ with $JJ_x \subseteq 2^{[L_x]}$, such that for the predicted output $y^* \neq y$, SetGamma returns a γ weight vector that respects the γ selection conditions defined as follows:

Definition 3. *The γ **selection conditions** for the SWVP algorithm are ($I = I^v \cup I^{nv}$):*

$$(1) \sum_{J \in I} \gamma(m^J) = 1. \quad \gamma(m^J) \geq 0, \quad \forall J \in I.$$

$$(2) \quad \boldsymbol{w} \cdot \sum_{J \in I} \gamma(m^J)\Delta\phi(x^i, y^i, m^J) \leq 0.$$

With this definition we are ready to prove the following property.

SWVP Generalizes the CSP Algorithm We now show that the CSP algorithm is a special case of SWVP. CSP can be derived from SWVP when taking: $JJ_x = \{[L_x]\}$, and $\gamma(m^{[L_x]}) = 1$ for every $(x, y) \in D$. With these parameters, the γ selection conditions hold for every $\mathbf{w}$ and y^*. Condition (1) holds trivially as there is only one γ coefficient and it is equal to 1. Condition (2) holds as $y^* = m^{[L_x]}$ and hence $I = \{[L_x]\}$ and $w \cdot \sum_{J \in I} \Delta\phi(x, y, m^J) \leq 0$.

4.1 Convergence for Linearly Separable Data

Here we give the theorem regarding the convergence of the SWVP in the separable case. We first define:

Definition 4. *A data set $D = \{x^i, y^i\}_{i=1}^n$ is **linearly separable with margin** $\delta > 0$ if there exists some vector $\boldsymbol{u}$ with $\|\boldsymbol{u}\|_2 = 1$ such that for all i:*

$$\boldsymbol{u} \cdot \Delta\phi(x^i, y^i, z) \geq \delta, \forall z \in \mathcal{Y}(x^i).$$

Definition 5. *The **radius** of a data set $D = \{x^i, y^i\}_{i=1}^n$ is the minimal scalar R s.t for all i:*

$$\|\Delta\phi(x^i, y^i, z)\| \leq R, \forall z \in \mathcal{Y}(x^i).$$

We next extend these definitions:

Definition 6. *Given a data set $D = \{x^i, y^i\}_{i=1}^n$ and a set $JJ = \{JJ_{x^i} \subseteq 2^{[L_{x^i}]} | (x^i, y^i) \in D\}$, D is **linearly separable w.r.t** JJ, **with margin** $\delta^{JJ} > 0$ if there exists a vector $\boldsymbol{u}$ with $\|\boldsymbol{u}\|_2 = 1$ such that: $\boldsymbol{u} \cdot \Delta\phi(x^i, y^i, m^J(z, y^i)) \geq \delta^{JJ}$ for all $i, z \in \mathcal{Y}(x^i), J \in JJ_{x^i}$.*

Definition 7. *The **mixed assignment radius w.r.t** JJ of a data set $D = \{x^i, y^i\}_{i=1}^n$ is a constant R^{JJ} s.t for all i it holds that:*

$$\|\Delta\phi(x^i, y^i, m^J(z, y^i))\| \leq R^{JJ}, \forall z \in \mathcal{Y}(x^i), J \in JJ_{x^i}.$$

With these definitions we can make the following observation (proof in A):

Observation 1. For linearly separable data D and a set JJ, every unit vector $\mathbf{u}$ that separates the data with margin δ, also separates the data with respect to mixed assignments with JJ, with margin $\delta^{JJ} \geq \delta$. Likewise, it holds that $R^{JJ} \leq R$.

We can now state our convergence theorem. While the proof of this theorem resembles that of the CSP (Collins, 2002), unlike the CSP proof the SWVP proof relies on the γ *selection conditions* presented above and on the *Jensen inequality*.

Theorem 1. *For any dataset D, linearly separable with respect to JJ with margin $\delta^{JJ} > 0$, the SWVP algorithm terminates after $t \leq \frac{(R^{JJ})^2}{(\delta^{JJ})^2}$ steps, where R^{JJ} is the mixed assignment radius of D w.r.t. JJ.*

Proof. Let $\mathbf{w}^t$ be the weight vector before the t^{th} update, thus $\mathbf{w}^1 = 0$. Suppose the t^{th} update occurs on example (x, y), i.e. for the predicted output y^* it holds that $y^* \neq y$. We will bound $\|\mathbf{w}^{t+1}\|^2$ from both sides.
First, it follows from the update rule of the algorithm that: $\mathbf{w}^{t+1} = \mathbf{w}^t + \sum_{J \in I^v \cup I^{nv}} \gamma(m^J)\Delta\phi(x, y, m^J)$.
For simplicity, in this proof we will use the notation $I^v \cup I^{nv} = I$. Hence, multiplying each side of the equation by $\mathbf{u}$ yields:

$$\begin{aligned}\mathbf{u} \cdot \mathbf{w}^{t+1} &= \mathbf{u} \cdot \mathbf{w}^t + \mathbf{u} \cdot \sum_{J \in I} \gamma(m^J)\Delta\phi(x, y, m^J)\\ &= \mathbf{u} \cdot \mathbf{w}^t + \sum_{J \in I} \gamma(m^J)\mathbf{u} \cdot \Delta\phi(x, y, m^J)\\ &\geq \mathbf{u} \cdot \mathbf{w}^t + \sum_{J \in I} \gamma(m^J)\delta^{JJ} \text{ (margin property)}\\ &\geq \mathbf{u} \cdot \mathbf{w}^t + \delta^{JJ} \geq \ldots \geq t\delta^{JJ}.\end{aligned}$$

The last inequality holds because $\sum_{J \in I} \gamma(m^J) = 1$. From this we get that $\|w^{t+1}\|^2 \geq (\delta^{JJ})^2 t^2$ since $\|u\|$=1. Second,

$$\begin{aligned}\|\mathbf{w}^{t+1}\|^2 &= \|\mathbf{w}^t + \sum_{J \in I} \gamma(m^J)\Delta\phi(x, y, m^J)\|^2\\ &= \|\mathbf{w}^t\|^2 + \|\sum_{J \in I} \gamma(m^J)\Delta\Phi(x, y, m^J)\|^2\\ &\quad + 2\mathbf{w}^t \cdot \sum_{J \in I} \gamma(m^J)\Delta\Phi(x, y, m^J).\end{aligned}$$

From γ selection condition (2) we get that:

$$\begin{aligned}\|\mathbf{w}^{t+1}\|^2 &\leq \|\mathbf{w}^t\|^2 + \|\sum_{J \in I} \gamma(m^J)\Delta\Phi(x, y, m^J)\|^2\\ &\leq \|\mathbf{w}^t\|^2 + \sum_{J \in I} \gamma(m^J)\|\Delta\Phi(x, y, m^J)\|^2\\ &\leq \|\mathbf{w}^t\|^2 + (R^{JJ})^2. \text{ (radius property)}\end{aligned}$$

The inequality one before the last results from the *Jensen inequality* which holds due to **(a)** γ selection condition (1); and **(b)** the squared norm function being convex. From this we finally get:

$$\|\mathbf{w}^{t+1}\|^2 \leq \|\mathbf{w}^t\|^2 + (R^{JJ})^2 \leq \ldots \leq t(R^{JJ})^2.$$

Combining the two steps we get:

$$(\delta^{JJ})^2 t^2 \leq \|\mathbf{w}^{t+1}\|^2 \leq t(R^{JJ})^2.$$

From this it is easy to derive the upper bound in the theorem: $t \leq \frac{(R^{JJ})^2}{(\delta^{JJ})^2}$. □

4.2 Convergence Properties

We next point on three properties of the SWVP algorithm, derived from its convergence proof:

Property 1 (tighter iterations bound) The convergence proof of CSP (Collins, 2002) is given for a vector $\mathbf{u}$ that linearly separates the data, with margin δ and for a data radius R. Following observation 1, it holds that in our case, $\mathbf{u}$ also linearly separates the data with respect to mixed assignments with a set JJ and with margin $\delta^{JJ} \geq \delta$. Together with the definition of $R^{JJ} \leq R$ we get that: $\frac{(R^{JJ})^2}{(\delta^{JJ})^2} \leq \frac{R^2}{\delta^2}$. This means that the bound on the number of updates made by SWVP is tighter than the bound of CSP.

Property 2 (inference) From the γ selection conditions it holds that any label from which at least one violating MA can be derived through JJ_x is suitable for an update. This is because in such a case we can choose, for example, a SETGAMMA function that assigns the weight of 1 to that MA, and the weight of 0 to all other MAs.

Algorithm 2 employs the $argmax$ inference function, following the basic reasoning that it is a good choice to base the parameter update on. Importantly, if the inference function is argmax and the algorithm performs an update ($y^* \neq y$), this means that y^*, the output of the argmax function, is a violating MA by definition. However, it is obvious that solving the inference problem and the optimal γ assignment problems jointly may result in more informed parameter ($\mathbf{w}$) updates. We leave a deeper investigation of this issue to future research.

Property 3 (dynamic updates) The γ selection conditions paragraph states two conditions ((a) and (b)) under which the convergence proof holds. While it is trivial for SETGAMMA to generate a γ vector that respects condition (a), if there is a parameter vector **w'** achievable by the algorithm for which SETGAMMA cannot generate γ that respects condition (b), SWVP gets stuck when reaching **w'**.

This problem can be solved with *dynamic updates*. A deep look into the convergence proof reveals that the set JJ_x and the SETGAMMA function can actually differ between iterations. While this will change the bound on the number of iterations, it will not change the fact that the algorithm converges if the data is linearly separable. This makes SWVP highly flexible as it can always back off to the CSP setup of $JJ_x = \{[L_x]\}$, and $\forall(x,y) \in D : \gamma(m^{[L_x]}) = 1$, update its parameters and continue with its original JJ and SETGAMMA when this option becomes feasible. If this does not happen, the algorithm can continue till convergence with the CSP setup.

4.3 Mistake and Generalization Bounds

The following bounds are proved: the number of updates in the separable case (see Theorem 1); the number of mistakes in the non-separable case (see Appendix B); and the probability to misclassify an unseen example (see supplementary material). It can be shown that in the general case these bounds are tighter than those of the CSP special case. We next discuss variants of SWVP.

5 Passive Aggressive SWVP

Here we present types of update rules that can be implemented within SWVP. Such rule types are defined by: (a) the selection of γ, which should respect the γ *selection conditions* (see Definition 3) and (b) the selection of $JJ = \{JJ_x \subseteq 2^{[L_x]} | (x,y) \in D\}$, the substructure sets for the training examples.

γ Selection A first approach we consider is the *aggressive approach*[4] where only mixed assignments that are violations $\{m^J : J \in I^v\}$ are exploited (i.e. for all $J \in I^{nv}, \gamma(m^J) = 0$). Note, that in this case condition (2) of the γ *selection conditions* trivially holds as: $\mathbf{w} \cdot \sum\limits_{J \in I^v} \gamma(m^J) \Delta\phi(x,y,m^J) \leq 0$. The only remaining requirement is that condition (1) also holds, i.e. that $\sum_{J \in I^v} \gamma(m^J) = 1$.

The opposite, *passive approach*, exploits only non-violating MA's $\{m^J : J \in I^{nv}\}$. However, such γ assignments do not respect γ selection condition (2), as they yield: $\mathbf{w} \cdot \sum_{J \in I^{nv}} \gamma(m^J) \Delta\phi(x,y,m^J) \leq 0$ which holds if and only if for every $J \in I^{nv}$, $\gamma(m^J) = 0$ that in turn contradicts condition (1).

Finally, we can take a *balanced approach* which gives a positive γ coefficient for at least one violating MA and at least one positive γ coefficient for a non-violating MA. This approach is allowed by SWVP as long as both γ selection conditions hold.

[4]We borrow the term *passive-aggressive* from (Crammer et al., 2006), despite the substantial difference between the works.

We implemented two weighting methods, both based on the concept of margin:

(1) Weighted Margin (WM): $\gamma(m^J) = \frac{|\mathbf{w} \cdot \Delta\phi(x,y,m^J)|^\beta}{\sum\limits_{J' \in JJ_x} |\mathbf{w} \cdot \Delta\phi(x,y,m^{J'})|^\beta}$

(2) Weighted Margin Rank (WMR): $\gamma(m^J) = \left(\frac{|JJ_x| - r}{|JJ_x|}\right)^\beta$. where r is the rank of $|\mathbf{w} \cdot \Delta\phi(x,y,m^J(y^*,y))|$ among the $|\mathbf{w} \cdot \Delta\phi(x,y,m^{J'}(y^*,y))|$ values for $J' \in JJ_x$.

Both schemes were implemented twice, within a balanced approach (denoted as B) and an aggressive approach (denoted as A).[5] The aggressive schemes respect both γ selection conditions. The balanced schemes, however, respect the first condition but not necessarily the second. Since all models that employ the balanced weighting schemes converged after at most 10 iterations, we did not impose this condition (which we could do by, e.g., excluding terms for $J \in I^{nv}$ till condition (2) holds).

JJ Selection Another choice that strongly affects the updates made by SWVP is that of JJ. A choice of $JJ_x = 2^{[L_x]}$, for every $(x,y) \in D$ results in an update rule which considers all possible mixing assignments derived from the predicted label y^* and the gold label y. Such an update rule, however, requires computing a sum over an exponential number of terms (2^{L_x}) and is therefore highly inefficient.

Among the wide range of alternative approaches, in this paper we exploit *single difference* mixed assignments. In this approach we define: $JJ = \{JJ_x = \{\{1\},\{2\},\dots\{L_x\}\} | (x,y) \in D\}$. For a training pair $(x,y) \in D$, a predicted label y^* and $J = \{j\} \in JJ_x$, we will have:

$$m^J_k(y^*,y) = \begin{cases} y_k & k \neq j \\ y^*_k & k = j \end{cases}$$

Under this approach for the pair $(x,y) \in D$ only L_x terms are summed in the SWVP update rule. We leave a further investigation of JJ selection approaches to future research.

6 Experiments

Synthetic Data We experiment with synthetic data generated by a linear-chain, first-

[5]For the aggressive approach the equations for schemes (1) and (2) are changed such that JJ_x is replaced with $I(y^*,y,JJ_x)^v$.

order Hidden Markov Model (HMM, (Rabiner and Juang, 1986)). Our learning algorithm is a liner-chain conditional random field (CRF, (Lafferty et al., 2001)): $P(y|x) = \frac{1}{Z(x)} \prod_{i=1:L_x} exp(w \cdot \phi(y_{i-1}, y_i, x))$ (where $Z(x)$ is a normalization factor) with binary indicator features $\{x_i, y_i, y_{i-1}, (x_i, y_i), (y_i, y_{i-1}), (x_i, y_i, y_{i-1})\}$ for the triplet (y_i, y_{i-1}, x).

A dataset is generated by iteratively sampling K items, each is sampled as follows. We first sample a hidden state, y_1, from a uniform prior distribution. Then, iteratively, for $i = 1, 2, \dots, L_x$ we sample an observed state from the emission probability and (for $i < L_x$) a hidden state from the transition probability. We experimented in 3 setups. In each setup we generated 10 datasets that were subsequently divided to a 7000 items training set, a 2000 items development set and a 1000 items test set. In all datasets, for each item, we set $L_x = 8$. We experiment in three conditions: (1) simple(++), learnable(+++), (2) simple(++), learnable(++) and (3) simple(+), learnable(+).[6]

For each dataset (3 setups, 10 datasets per setup) we train variants of the SWVP algorithm differing in the γ selection strategy (WM or WMR, Section 5), being aggressive (A) or passive (B), and in their β parameter ($\beta = \{0.5, 1, \dots, 5\}$). Training is done on the training subset and the best performing variant on the development subset is applied to the test subset. For CSP no development set is employed as there is no hyper-parameter to tune. We report averaged accuracy (fraction of observed states for which the model successfully predicts the hidden state value) across the test sets, together with the standard deviation.

Dependency Parsing We also report initial dependency parsing results. We implemented our algorithms within the TurboParser (Martins et al., 2013). That is, every other aspect of the parser: feature set, probabilistic pruning algorithm, inference algorithm etc., is kept fixed but training is performed with SWVP. We compare our results to the parser performance with CSP training (which comes with the standard implementation of the parser).

We experiment with the datasets of the CoNLL 2007 shared task on multilingual dependency parsing (Nilsson et al., 2007), for a total of 9 languages. We followed the standard train/test split of these dataset. For SWVP, we randomly sampled 1000 sentences from each training set to serve as development sets and tuned the parameters as in the synthetic data experiments. CSP is trained on the training set and applied to the test set without any development set involved. We report the Unlabeled Attachment Score (UAS) for each language and model.

7 Results

Synthetic Data Table 1 presents our results. In all three setups an SWVP algorithm is superior. Averaged accuracy differences between the best performing algorithms and CSP are: 3.72 (B-WMR, (simple(++), learnable(+++))), 5.29 (B-WM, (simple(++), learnable(++))) and 5.18 (A-WM, (simple(+), learnable(+))). In all setups SWVP outperforms CSP in terms of averaged performance (except from B-WMR for (simple(+), learnable(+))). Moreover, the weighted models are more stable than CSP, as indicated by the lower standard deviation of their accuracy scores. Finally, for the more simple and learnable datasets the SWVP models outperform CSP in the majority of cases (7-10/10).

We measure generalization from development to test data in two ways. First, for each SWVP algorithm we count the number of times its β parameter results in an algorithm that outperforms the CSP on the development set but not on the test set (not shown in the table). Of the 120 comparisons reported in the table (4 SWVP models, 3 setups, 10 comparisons per model/setup combination) this happened once (A-MV, (simple(++), learnable(+++)).

Second, we count the number of times the best development set value of the β hyper-parameter is also the best value on the test set, or the test set accuracy with the best development set β is at most 0.5% lower than that with the best test set β. The *Gener-*

[6]Denoting $D_x = [C_x]$, $D_y = [C_y]$, and a permutation of a vector v with $perm(v)$, the parameters of the different setups are: (1) simple(++), learnable(+++): $C_x = 5$, $C_y = 3$, $P(y'|y) = perm(0.7, 0.2, 0.1)$, $P(x|y) = perm(0.75, 0.1, 0.05, 0.05, 0.05)$. (2) simple(++), learnable(++): $C_x = 5$, $C_y = 3$, $P(y'|y) = perm(0.5, 0.3, 0.2)$, $P(x|y) = perm(0.6, 0.15, 0.1, 0.1, 0.05)$. (3) simple(+), learnable(+): $C_x = 20$, $C_y = 7$, $P(y'|y) = perm(0.7, 0.2, 0.1, 0, \dots, 0))$, $P(x|y) = perm(0.4, 0.2, 0.1, 0.1, 0.1, 0, \dots, 0)$.

	simple(++), learnable(+++)			simple(++), learnable(++)			simple(+), learnable(+)		
Model	Acc. (std)	# Wins	Gener.	Acc. (std)	# Wins	Gener.	Acc. (std)	# Wins	Gener.
B-WM	75.47(3.05)	**9/10**	10/10	**63.18 (1.32)**	**9/10**	10/10	28.48 (1.9)	5/10	10/10
B-WMR	**75.96 (2.42)**	8/10	10/10	63.02 (2.49)	9/10	10/10	24.31 (5.2)	4/10	10/10
A-WM	74.18 (2.16)	7/10	10/10	61.65 (2.30)	**9/10**	10/10	**30.45 (1.0)**	**6/10**	10/10
A-WMR	75.17 (3.07)	7/10	10/10	61.02 (1.93)	8/10	10/10	25.8 (3.18)	2/10	10/10
CSP	72.24 (3.45)	NA	NA	57.89 (2.85)	NA	NA	25.27(8.55)	NA	NA

Table 1: Overall Synthetic Data Results. *A-* and *B-* denote an aggressive and a balanced approaches, respectively. Acc. (std) is the average and the standard deviation of the accuracy across 10 test sets. # Wins is the number of test sets on which the SWVP algorithm outperforms CSP. Gener. is the number of times the best β hyper-parameter value on the development set is also the best value on the test set, or the test set accuracy with the best development set β is at most 0.5% lower than that with the best test set β.

	First Order				Second Order			
Language	CSP	B-WM	Top B-WM	Test B-WM	CSP	B-WM	Top B-WM	Test B-WM
English	86.34	86.4	**86.7**	86.7	**88.02**	87.82	87.82	87.92
Chinese	84.60	84.5	**85.04**	85.05	86.82	86.69	**86.83**	87.02
Arabic	79.09	79.17	**79.21**	79.21	76.07	75.94	**76.09**	76.09
Greek	**80.41**	80.20	80.28	80.28	80.31	**80.40**	**80.40**	80.61
Italian	84.63	84.64	**84.74**	84.70	84.03	84.08	**84.15**	84.28
Turkish	**83.05**	82.89	82.89	82.89	83.02	**83.04**	**83.04**	83.31
Basque	79.47	**79.54**	**79.54**	79.54	80.52	80.57	**80.63**	80.64
Catalan	**88.51**	88.46	88.50	88.5	88.71	**88.81**	**88.81**	88.82
Hungarian	**80.17**	80.07	80.07	80.21	**80.61**	80.45	80.45	80.55
Average	83.69	83.65	**83.77**	83.79	83.12	83.08	**83.13**	83.35

Table 2: First and second order dependency parsing UAS results for CSP trained models, as well as for models trained with SWVP with a balanced γ selection (B) and with a weighted margin (WM) strategy. For explanation of the B-WM, Top B-WM, and Test B-WM see text. For each language and parsing order we highlight the best result in bold font, but this do not include results from Test B-WM as it is provided only as an upper bound on the performance of SWVP.

alization column of the table shows that this has not happened in all of the 120 runs of SWVP.

Dependency Parsing Results are given in Table 2. For the SWVP trained models we report three numbers: (a) B-WM is the standard setup where the β hyper parameter is tuned on the development data; (b) For Top B-WM we first selected the models with a UAS score within 0.1% of the best development data result, and of these we report the UAS of the model that performs best on the test set; and (c) Test B-WM reports results when β is tuned on the test set. This measure provides an upper bound on SWVP with our simplistic JJ (Section 5).

Our results indicate the potential of SWVP. Despite our simple JJ set, Top B-WM and Test B-WM improve over CSP in 5/9 and 6/9 cases in first order parsing, respectively, and in 7/9 cases in second order parsing. In the latter case, Test B-WM improves the UAS over CSP in 0.22% on average across languages. Unfortunately, SWVP still does not generalize well from train to test data as indicated, e.g., by the modest improvements B-WM achieves over CSP in only 5 of 9 languages in second order parsing.

8 Conclusions

We presented the Structured Weighted Violations Perceptron (SWVP) algorithm, a generalization of the Structured Perceptron (CSP) algorithm that explicitly exploits the internal structure of the predicted label in its update rule. We proved the convergence of the algorithm for linearly separable training sets under certain conditions on its parameters, and provided generalization and mistake bounds.

In experiments we explored only very simple configurations of the SWVP parameters - γ and JJ. Nevertheless, several of our SWVP variants outperformed the CSP special case in synthetic data experiments. In dependency parsing experiments, SWVP demonstrated some improvements over CSP, but these do not generalize well. While we find these results somewhat encouraging, they emphasize the need to explore the much more flexible γ and JJ selection strategies allowed by SWVP (Sec. 4.2). In future work we will hence develop γ and JJ selection algorithms, where selection is ideally performed jointly with inference (property 2, Sec. 4.2), to make SWVP practically useful in NLP applications.

A Proof Observation 1.

Proof. For every training example $(x, y) \in D$, it holds that: $\cup_{z\in\mathcal{Y}(x)} m^J(z, y) \subseteq \mathcal{Y}(x)$. As **u** separates the data with margin δ, it holds that:

$$\mathbf{u} \cdot \Delta\phi(x, y, m^J(z, y)) \geq \delta^{JJ_x}, \quad \forall z \in \mathcal{Y}(x), J \in JJ_x.$$
$$\mathbf{u} \cdot \Delta\phi(x, y, z) \geq \delta, \quad \forall z \in \mathcal{Y}(x).$$

Therefore also $\delta^{JJ_x} \geq \delta$. As the last inequality holds for every $(x, y) \in D$ we get that $\delta^{JJ} = \min_{(x,y)\in D} \delta^{JJ_x} \geq \delta$.
From the same considerations it holds that $R^{JJ} \leq R$. This is because R^{JJ} is the radius of a subset of the dataset with radius R (proper subset if $\exists(x, y) \in D, [L_x] \notin JJ_x$, non-proper subset otherwise). □

B Mistake Bound - Non Separable Case

Here we provide a mistake bound for the algorithm in the non-separable case. We start with the following definition and observation:

Definition 8. *Given an example $(x^i, y^i) \in D$, for a $\boldsymbol{u}, \delta$ pair define:*

$$r^i = \boldsymbol{u} \cdot \phi(x^i, y^i) - \max_{z\in\mathcal{Y}(x^i)} \boldsymbol{u} \cdot \phi(x^i, z)$$
$$\epsilon_i = \max\{0, \delta - r^i\}$$
$$r^{i^{JJ}} = \boldsymbol{u} \cdot \phi(x^i, y^i) - \max_{z\in\mathcal{Y}(x^i), J\in JJ_{x^i}} \boldsymbol{u} \cdot \phi(x^i, m^J(z, y^i))$$

Finally define: $D_{\boldsymbol{u},\delta} = \sqrt{\sum_{i=1}^{n} \epsilon_i^2}$

Observation 2. For all i: $r^i \leq r^{i^{JJ}}$.
Observation 2 easily follows from Definition 8. Following this observation we denote: $r^{diff} = \min_i\{r^{i^{JJ}} - r^i\} \geq 0$ and present the next theorem:

Theorem 2. *For any training sequence D, for the **first** pass over the training set of the CSP and the SWVP algorithms respectively, it holds that:*

$$\#mistakes - CSP \leq \min_{\boldsymbol{u}:\|\boldsymbol{u}\|=1,\delta>0} \frac{(R + D_{\boldsymbol{u},\delta})^2}{\delta^2}.$$
$$\#mistakes - SWVP \leq \min_{\boldsymbol{u}:\|\boldsymbol{u}\|=1,\delta>0} \frac{(R^{JJ} + D_{\boldsymbol{u},\delta})^2}{(\delta + r^{diff})^2}.$$

As $R^{JJ} \leq R$ (Observation 1) and $r^{diff} \geq 0$, we get a tighter bound for SWVP. The proof for #mistakes-CSP is given at (Collins, 2002). The proof for #mistakes-SWVP is given below.

Proof. We transform the representation $\phi(x, y) \in \mathbb{R}^d$ into a new representation $\psi(x, y) \in \mathbb{R}^{d+n}$ as follows: for $i = 1, ..., d : \psi_i(x, y) = \phi_i(x, y)$, for $j = 1, ..., n : \psi_{d+j}(x, y) = \Delta$ if $(x, y) = (x^j, y^j)$ and 0 otherwise, where $\Delta > 0$ is a parameter.
Given a $\mathbf{u}, \delta$ pair define $\mathbf{v} \in \mathbb{R}^{d+n}$ as follows: for $i = 1, ..., d : \mathbf{v}_i = \mathbf{u}_i$, for $j = 1, ..., n : \mathbf{v}_{d+j} = \frac{\epsilon_j}{\Delta}$.
Under these definitions we have:

$$\mathbf{v} \cdot \psi(x^i, y^i) - \mathbf{v} \cdot \psi(x^i, z) \geq \delta, \quad \forall i, z \in \mathcal{Y}(x^i).$$

For every $i, z \in \mathcal{Y}(x^i), J \in JJ_{x^i}$:

$$\mathbf{v} \cdot \psi(x^i, y^i) - \mathbf{v} \cdot \psi(x^i, m^J(z, y^i)) \geq \delta + r^{diff}.$$
$$\|\psi(x^i, y^i) - \psi(x^i, m^J(z, y^i))\|^2 \leq (R^{JJ})^2 + \Delta^2.$$

Last, we have,

$$\|\mathbf{v}\|^2 = \|\mathbf{u}\|^2 + \sum_{i=1}^{n} \frac{\epsilon_i^2}{\Delta^2} = 1 + \frac{D_{\mathbf{u},\delta}^2}{\Delta^2}.$$

We get that the vector $\frac{\mathbf{v}}{\|\mathbf{v}\|}$ linearly separates the data with respect to single decision assignments with margin $\frac{\delta}{\sqrt{1+\frac{D_{U,\delta}^2}{\Delta^2}}}$. Likewise, $\frac{\mathbf{v}}{\|\mathbf{v}\|}$ linearly separates the data with respect to mixed assignments with JJ, with margin $\frac{\delta+r^{diff}}{\sqrt{1+\frac{D_{\mathbf{u},\delta}}{\Delta^2}}}$. Notice that the **first** pass of SWVP with representation Ψ is identical to the first pass with representation Φ because the parameter weight for the additional features affects only a single example of the training data and do not affect the classification of test examples. By theorem 1 this means that the **first** pass of SWVP with representation Ψ makes at most $\frac{((R^{JJ})^2+\Delta^2)}{(\delta+r^{diff})^2} \cdot \left(1 + \frac{D_{\mathbf{u},\delta}^2}{\Delta^2}\right)$.
We minimize this w.r.t Δ, which gives: $\Delta = \sqrt{R^{JJ} D_{\mathbf{u},\delta}}$, and obtain the result guaranteed in the theorem. □

Acknowledgments

The second author was partly supported by a research grant from the GIF Young Scientists' Program (No. I-2388-407.6/2015): Syntactic Parsing in Context.

References

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proc. of ACL*.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proc. of ACL*, pages 16–23.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proc. of EMNLP*, pages 1–8.

Koby Crammer and Yoram Singer. 2003. Ultraconservative online algorithms for multiclass problems. *The Journal of Machine Learning Research*, 3:951–991.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 7:551–585.

Hal Daumé III and Daniel Marcu. 2005. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proc. of ICML*, pages 169–176.

Yoav Freund and Robert E Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296.

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Proc. of NAACL-HLT 2010*, pages 742–750.

Raphael Hoffmann, Congle Zhang, Xiao Ling, Luke Zettlemoyer, and Daniel S Weld. 2011. Knowledge-based weak supervision for information extraction of overlapping relations. In *Proc. of ACL*.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proc. of NAACL-HLT*, pages 142–151.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proc. of ACL*, pages 1–11.

John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of ICML*.

André FT Martins, Miguel Almeida, and Noah A Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Prc. of ACL short papers*, pages 617–622.

Ryan T McDonald and Fernando CN Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proc. of EACL*.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005a. Online large-margin training of dependency parsers. In *Proc. of ACL*, pages 91–98.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proc. of EMNLP-HLT*, pages 523–530.

Ofer Meshi, David Sontag, Tommi Jaakkola, and Amir Globerson. 2010. Learning efficiently with approximate inference via dual losses. In *Proc. of ICML*.

Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The conll 2007 shared task on dependency parsing. In *Proceedings of the CoNLL shared task session of EMNLP-CoNLL*, pages 915–932. sn.

Lawrence Rabiner and Biing-Hwang Juang. 1986. An introduction to hidden markov models. *ASSP Magazine, IEEE*, 3(1):4–16.

Roi Reichart and Regina Barzilay. 2012. Multi event extraction guided by global constraints. In *Proc. of NAACL-HLT 2012*, pages 70–79.

Ben Taskar, Carlos Guestrin, and Daphne Koller. 2004. Max-margin markov networks. In *Proc. of NIPS*.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. 2005. Large margin methods for structured and interdependent output variables. In *Journal of Machine Learning Research*, pages 1453–1484.

Luke S Zettlemoyer and Michael Collins. 2007. Online learning of relaxed ccg grammars for parsing to logical form. In *Proc. of EMNLP-CoNLL*, pages 678–687.

Yue Zhang and Stephen Clark. 2008. Joint word segmentation and pos tagging using a single perceptron. In *proc. of ACL*, pages 888–896.