Vine Pruning for Efficient Multi-Pass Dependency Parsing

Alexander M. Rush*
MIT CSAIL
Cambridge, MA 02139, USA
srush@csail.mit.edu

Slav Petrov
Google
New York, NY 10027, USA
slav@google.com

Abstract

Coarse-to-fine inference has been shown to be a robust approximate method for improving the efficiency of structured prediction models while preserving their accuracy. We propose a multi-pass coarse-to-fine architecture for dependency parsing using linear-time vine pruning and structured prediction cascades. Our first-, second-, and third-order models achieve accuracies comparable to those of their unpruned counterparts, while exploring only a fraction of the search space. We observe speed-ups of up to two orders of magnitude compared to exhaustive search. Our pruned third-order model is twice as fast as an unpruned first-order model and also compares favorably to a state-of-the-art transition-based parser for multiple languages.

1 Introduction

Coarse-to-fine inference has been extensively used to speed up structured prediction models. The general idea is simple: use a coarse model where inference is cheap to prune the search space for more complex models. In this work, we present a multi-pass coarse-to-fine architecture for graph-based dependency parsing. We start with a linear-time vine pruning pass and build up to higher-order models, achieving speed-ups of two orders of magnitude while maintaining state-of-the-art accuracies.

In constituency parsing, exhaustive inference for all but the simplest grammars tends to be prohibitively slow. Consequently, most high-accuracy constituency parsers routinely employ a coarse grammar to prune dynamic programming chart cells of the final grammar of interest (Charniak et al., 2006; Carreras et al., 2008; Petrov, 2009). While there are no strong theoretical guarantees for these approaches,[1] in practice one can obtain significant speed improvements with minimal loss in accuracy. This benefit comes primarily from reducing the large grammar constant $|G|$ that can dominate the runtime of the cubic-time CKY inference algorithm. Dependency parsers on the other hand do not have a multiplicative grammar factor $|G|$, and until recently were considered efficient enough for exhaustive inference. However, the increased model complexity of a third-order parser forced Koo and Collins (2010) to prune with a first-order model in order to make inference practical. While fairly effective, all these approaches are limited by the fact that inference in the coarse model remains cubic in the sentence length. The desire to parse vast amounts of text necessitates more efficient dependency parsing algorithms.

We thus propose a multi-pass coarse-to-fine approach where the initial pass is a linear-time sweep, which tries to resolve local ambiguities, but leaves arcs beyond a fixed length b unspecified (Section 3). The dynamic program is a form of *vine parsing* (Eisner and Smith, 2005), which we use to compute parse max-marginals, rather than for finding the 1-best parse tree. To reduce pruning errors, the parameters of the vine parser (and all subsequent pruning models) are trained using the *structured prediction cascades* of Weiss and Taskar (2010) to optimize for pruning efficiency, and not for 1-best prediction (Section 4). Despite a limited scope of $b = 3$, the

* Research conducted at Google.

[1] This is in contrast to optimality preserving methods such as A* search, which typically do not provide sufficient speed-ups (Pauls and Klein, 2009).

vine pruning pass is able to preserve $>98\%$ of the correct arcs, while ruling out $\sim 86\%$ of all possible arcs. Subsequent i-th order passes introduce larger scope features, while further constraining the search space. In Section 5 we present experiments in multiple languages. Our coarse-to-fine first-, second-, and third-order parsers preserve the accuracy of the unpruned models, but are faster by up to two orders of magnitude. Our pruned third-order model is faster than an unpruned first-order model, and compares favorably in speed to the state-of-the-art transition-based parser of Zhang and Nivre (2011).

It is worth noting the relationship to greedy transition-based dependency parsers that are also linear-time (Nivre et al., 2004) or quadratic-time (Yamada and Matsumoto, 2003). It is their success that motivates building explicitly trained, linear-time pruning models. However, while a greedy solution for arc-standard transition-based parsers can be computed in linear-time, Kuhlmann et al. (2011) recently showed that computing exact solutions or (max-)marginals has time complexity $O(n^4)$, making these models inappropriate for coarse-to-fine style pruning. As an alternative, Roark and Hollingshead (2008) and Bergsma and Cherry (2010) present approaches where individual classifiers are used to prune chart cells. Such approaches have the drawback that pruning decisions are made locally and therefore can rule out all valid structures, despite explicitly evaluating $O(n^2)$ chart cells. In contrast, we make pruning decisions based on global parse max-marginals using a vine pruning pass, which is linear in the sentence length, but nonetheless guarantees to preserve a valid parse structure.

2 Motivation & Overview

The goal of this work is fast, high-order, graph-based dependency parsing. Previous work on constituency parsing demonstrates that performing several passes with increasingly more complex models results in faster inference (Charniak et al., 2006; Petrov and Klein, 2007). The same technique applies to dependency parsing with a cascade of models of increasing order; however, this strategy is limited by the speed of the simplest model. The algorithm for first-order dependency parsing (Eisner, 2000) already requires $O(n^3)$ time, which Lee

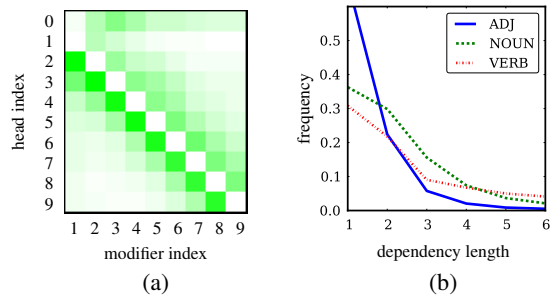

Figure 1: (a) Heat map indicating how likely a particular head position is for each modifier position. Greener/darker is likelier. (b) Arc length frequency for three common modifier tags. Both charts are computed from all sentences in Section 22 of the PTB.

(2002) shows is a practical lower bound for parsing of context-free grammars. This bound implies that it is unlikely that there can be an exhaustive parsing algorithm that is asymptotically faster than the standard approach.

We thus need to leverage domain knowledge to obtain faster parsing algorithms. It is well-known that natural language is fairly linear, and most head-modifier dependencies tend to be short. This property is exploited by transition-based dependency parsers (Yamada and Matsumoto, 2003; Nivre et al., 2004) and empirically demonstrated in Figure 1. The heat map on the left shows that most of the probability mass of modifiers is concentrated among nearby words, corresponding to a diagonal band in the matrix representation. On the right we show the frequency of arc lengths for different modifier part-of-speech tags. As one can expect, almost all arcs involving adjectives (ADJ) are very short (length 3 or less), but even arcs involving verbs and nouns are often short. This structure suggests that it may be possible to disambiguate most dependencies by considering only the "banded" portion of the sentence.

We exploit this linear structure by employing a variant of vine parsing (Eisner and Smith, 2005). Vine parsing is a dependency parsing algorithm that considers only close words as modifiers. Because of this assumption it runs in linear time.[2] Of course, any parse tree with hard limits on dependency lengths will contain major parse errors. We therefore use the vine parser only for pruning and

[2] Vine parsers are as expressive as finite-state automata. This allows them to circumvent the cubic-time bound.

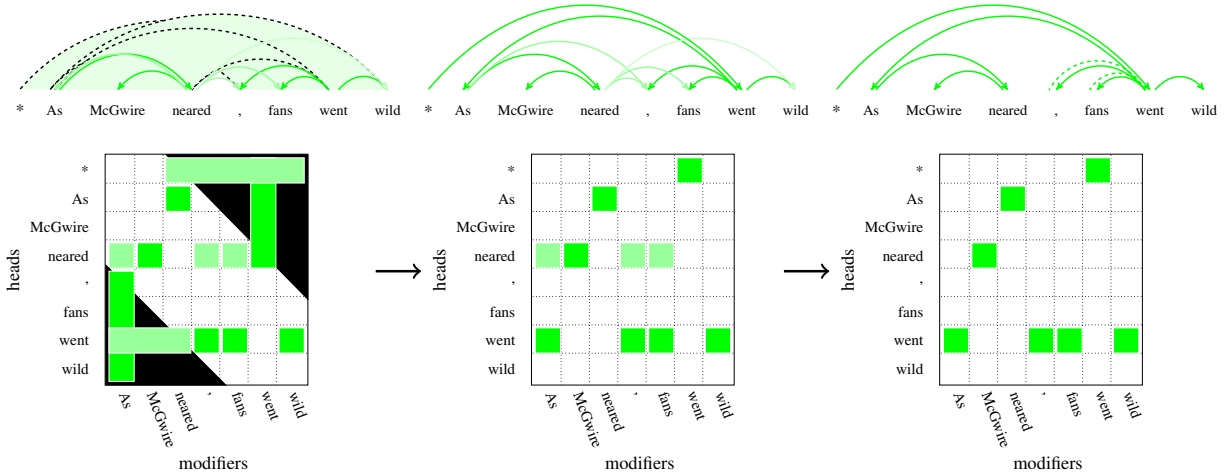

Figure 2: Multi-pass pruning with a vine, first-order, and second-order model shown as dependencies and filtered index sets after each pass. Darker cells have higher max-marginal values, while empty cells represent pruned arcs.

augment it to allow arcs to remain unspecified (by including so called *outer* arcs). The vine parser can thereby eliminate a possibly quadratic number of arcs, while having the flexibility to defer some decisions and preserve ambiguity to be resolved by later passes. In Figure 2 for example, the vine pass correctly determined the head-word of *McGwire* as *neared*, limited the head-word candidates for *fans* to *neared* and *went*, and decided that the head-word for *went* falls outside the band by proposing an outer arc. A subsequent first-order pass needs to score only a small fraction of all possible arcs and can be used to further restrict the search space for the following higher-order passes.

3 Graph-Based Dependency Parsing

Graph-based dependency parsing models factor all valid parse trees for a given sentence into smaller units, which can be scored independently. For instance, in a first-order factorization, the units are just dependency arcs. We represent these units by an index set $\mathcal{I}$ and use binary vectors $\mathcal{Y} \subset \{0,1\}^{|\mathcal{I}|}$ to specify a parse tree $y \in \mathcal{Y}$ such that $y(i) = 1$ iff the index i exists in the tree. The index sets of higher-order models can be constructed out of the index sets of lower-order models, thus forming a hierarchy that we will exploit in our coarse-to-fine cascade.

The inference problem is to find the 1-best parse tree $\arg\max_{y \in \mathcal{Y}} y \cdot w$, where $w \in R^{|\mathcal{I}|}$ is a weight vector that assigns a score to each index i (we discuss how w is learned in Section 4). A generalization of the 1-best inference problem is to find the max-marginal score for each index i. Max-marginals are given by the function $M : \mathcal{I} \to \mathcal{Y}$ defined as $M(i; \mathcal{Y}, w) = \arg\max_{y \in \mathcal{Y}: y(i)=1} y \cdot w$. For first-order parsing, this corresponds to the best parse utilizing a given dependency arc. Clearly there are exponentially many possible parse tree structures, but fortunately there exist well-known dynamic programming algorithms for searching over all possible structures. We review these below, starting with the first-order factorization for ease of exposition.

Throughout the paper we make use of some basic mathematical notation. We write $[c]$ for the enumeration $\{1, \ldots, c\}$ and $[c]_a$ for $\{a, \ldots, c\}$. We use $\mathbf{1}[c]$ for the indicator function, equal to 1 if condition c is true and 0 otherwise. Finally we use $[c]_+ = \max\{0, c\}$ for the positive part of c.

3.1 First-Order Parsing

The simplest way to index a dependency parse structure is by the individual arcs of the parse tree. This model is known as first-order or arc-factored. For a sentence of length n the index set is:

$$\mathcal{I}_1 = \{(h, m) : h \in [n]_0, m \in [n]\}$$

Each dependency tree has $y(h, m) = 1$ iff it includes an arc from head h to modifier m. We follow common practice and use position 0 as the pseudo-root ($*$) of the sentence. The full set $\mathcal{I}_1$ has cardinality $|\mathcal{I}_1| = O(n^2)$.

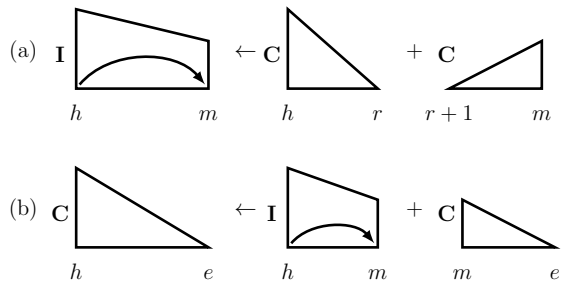

Figure 3: Parsing rules for first-order dependency parsing. The complete items C are represented by triangles and the incomplete items I are represented by trapezoids. Symmetric left-facing versions are also included.

The first-order bilexical parsing algorithm of Eisner (2000) can be used to find the best parse tree and max-marginals. The algorithm defines a dynamic program over two types of items: *incomplete* items $I(h, m)$ that denote the span between a modifier m and its head h, and *complete* items $C(h, e)$ that contain a full subtree spanning from the head h and to the word e on one side. The algorithm builds larger items by applying the composition rules shown in Figure 3. Rule 3(a) builds an incomplete item $I(h, m)$ by attaching m as a modifier to h. This rule has the effect that $y(h, m) = 1$ in the final parse. Rule 3(b) completes item $I(h, m)$ by attaching item $C(m, e)$. The existence of $I(h, m)$ implies that m modifies h, so this rule enforces that the constituents of m are also constituents of h.

We can find the best derivation for each item by adapting the standard CKY parsing algorithm to these rules. Since both rule types contain three variables that can range over the entire sentence ($h, m, e \in [n]_0$), the bottom-up, inside dynamic programming algorithm requires $O(n^3)$ time. Furthermore, we can find max-marginals with an additional top-down outside pass also requiring cubic time. To speed up search, we need to filter indices from $\mathcal{I}_1$ and reduce possible applications of Rule 3(a).

3.2 Higher-Order Parsing

Higher-order models generalize the index set by using siblings s (modifiers that previously attached to a head word) and grandparents g (head words above the current head word). For compactness, we use g_1 for the head word and s_{k+1} for the modifier and parameterize the index set to capture arbitrary higher-order decisions in both directions:

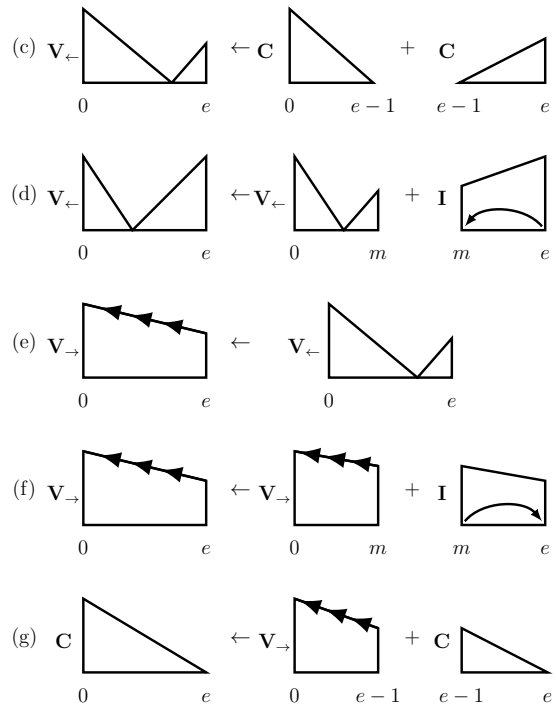

Figure 4: Additional rules for vine parsing. Vine left ($V_{\leftarrow}$) items are pictured as right-facing triangles and vine right ($V_{\rightarrow}$) items are marked trapezoids. Each new item is anchored at the root and grows to the right.

$$\mathcal{I}_{k,l} = \{(g, s) : g \in [n]_0^{l+1}, s \in [n]^{k+1}\}$$

where $k + 1$ is the sibling order, $l + 1$ is the parent order, and $k + l + 1$ is the model order. The canonical second-order model uses $\mathcal{I}_{1,0}$, which has a cardinality of $O(n^3)$. Although there are several possibilities for higher-order models, we use $\mathcal{I}_{1,1}$ as our third-order model. Generally, the parsing index set has cardinality $|\mathcal{I}_{k,l}| = O(n^{2+k+l})$. Inference in higher-order models uses variants of the dynamic program for first-order parsing, and we refer to previous work for the full set of rules. For second-order models with index set $\mathcal{I}_{1,0}$, parsing can be done in $O(n^3)$ time (McDonald and Pereira, 2006) and for third-order models in $O(n^4)$ time (Koo and Collins, 2010). Even though second-order parsing has the same asymptotic time complexity as first-order parsing, inference is significantly slower due to the cost of scoring the larger index set.

We aim to prune the index set, by mapping each higher-order index down to a set of small set indices

that can be pruned using a coarse pruning model. For example, to use a first-order model for pruning, we would map the higher-order index to the individual indices for its arc, grandparents, and siblings:

$$\begin{aligned} p_{k,l\to 1}(g,s) &= \{(g_1, s_j) : j \in [k+1]\} \\ &\cup \{(g_{j+1}, g_j) : j \in [l]\} \end{aligned}$$

The first-order pruning model can then be used to score these indices, and to produce a filtered index set $F(\mathcal{I}_1)$ by removing low-scoring indices (see Section 4). We retain only the higher-order indices that are supported by the filtered index set:

$$\{(g,s) \in \mathcal{I}_{k,l} : p_{k,l\to 1}(g,s) \subset F(\mathcal{I}_1)\}$$

3.3 Vine Parsing

To further reduce the cost of parsing and produce faster pruning models, we need a model with less structure than the first-order model. A natural choice, following Section 2, is to only consider "short" arcs:

$$\mathcal{S} = \{(h,m) \in \mathcal{I}_1 \quad : \quad |h-m| \leq b\}$$

where b is a small constant. This constraint reduces the size of the set to $|\mathcal{S}| = O(nb)$.

Clearly, this index set is severely limited; it is necessary to have some long arcs for even short sentences. We therefore augment the index set to include *outer* arcs:

$$\begin{aligned} \mathcal{I}_0 = \mathcal{S} \quad &\cup \quad \{(d,m) : d \in \{\leftarrow, \rightarrow\}, m \in [n]\} \\ &\cup \quad \{(h,d) : h \in [n]_0, d \in \{\leftarrow, \rightarrow\}\} \end{aligned}$$

The first set lets modifiers choose an outer headword and the second set lets head words accept outer modifiers, and both sets distinguish the direction of the arc. Figure 5 shows a right outer arc. The size of $\mathcal{I}_0$ is linear in the sentence length. To parse the index set $\mathcal{I}_0$, we can modify the parse rules in Figure 3 to enforce additional length constraints ($|h-e| \leq b$ for $I(h,e)$ and $|h-m| \leq b$ for $C(h,m)$). This way, only indices in $\mathcal{S}$ are explored. Unfortunately, this is not sufficient since the constraints also prevent the algorithm from producing a full derivation, since no item can expand beyond length b.

Eisner and Smith (2005) therefore introduce vine parsing, which includes two new items, *vine left*, $V_{\leftarrow}(e)$, and *vine right*, $V_{\rightarrow}(e)$. Unlike the previous items, these new items are left-anchored at the root and grow only towards the right. The items $V_{\leftarrow}(e)$ and $V_{\rightarrow}(e)$ encode the fact that a word e has not taken a close (within b) head word to its left or right. We incorporate these items by adding the five new parsing rules shown in Figure 4.

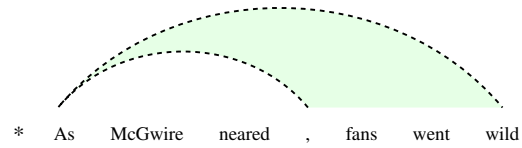

Figure 5: An outer arc $(1, \rightarrow)$ from the word "As" to possible right modifiers.

The major addition is Rule 4(e) which converts a vine left item $V_{\leftarrow}(e)$ to a vine right item $V_{\rightarrow}(e)$. This implies that word e has no close head to either side, and the parse has outer head arcs, $y(\leftarrow, e) = 1$ or $y(\rightarrow, e) = 1$. The other rules are structural and dictate creation and extension of vine items. Rules 4(c) and 4(d) create vine left items from items that cannot find a head word to their left. Rules 4(f) and 4(g) extend and finish vine right items. Rules 4(d) and 4(f) each leave a head word incomplete, so they may set $y(e, \leftarrow) = 1$ or $y(m, \rightarrow) = 1$ respectively. Note that for all the new parse rules, $e \in [n]_0$ and $m \in \{e-b \ldots n\}$, so parsing time of this so called vine parsing algorithm is linear in the sentence length $O(nb^2)$.

Alone, vine parsing is a poor model of syntax - it does not even score most dependency pairs. However, it can act as a pruning model for other parsers. We prune a first-order model by mapping first-order indices to indices in $\mathcal{I}_0$.

$$p_{1\to 0}(h,m) = \begin{cases} \{(h,m)\} & \text{if } |h-m| \leq b \\ \{(\rightarrow, m), (h, \rightarrow)\} & \text{if } h < m \\ \{(\leftarrow, m), (h, \leftarrow)\} & \text{if } h > m \end{cases}$$

The remaining first-order indices are then given by:

$$\{(h,m) \in \mathcal{I}_1 : p_{1\to 0}(h,m) \subset F(\mathcal{I}_0)\}$$

Figure 2 depicts a coarse-to-fine cascade, incorporating vine and first-order pruning passes and finishing with a higher-order parse model.

4 Training Methods

Our coarse-to-fine parsing architecture consists of multiple pruning passes followed by a final pass of 1-best parsing. The training objective for the pruning models comes from the prediction cascade framework of Weiss and Taskar (2010), which explicitly trades off pruning efficiency versus accuracy. The models used in the final pass on the other hand are trained for 1-best prediction.

4.1 Max-Marginal Filtering

At each pass of coarse-to-fine pruning, we apply an index filter function F to trim the index set:

$$F(\mathcal{I}) = \{i \in \mathcal{I} : f(i) = 1\}$$

Several types of filters have been proposed in the literature, with most work in coarse-to-fine parsing focusing on predicates that threshold the posterior probabilities. In structured prediction cascades, we use a non-probabilistic filter, based on the max-marginal value of the index:

$$f(i; \mathcal{Y}, w) = \mathbf{1}[\ M(i; \mathcal{Y}, w) \cdot w < t_\alpha(\mathcal{Y}, w)\ ]$$

where $t_\alpha(\mathcal{Y}, w)$ is a sentence-specific threshold value. To counteract the fact that the max-marginals are not normalized, the threshold $t_\alpha(\mathcal{Y}, w)$ is set as a convex combination of the 1-best parse score and the average max-marginal value:

$$\begin{aligned} t_\alpha(\mathcal{Y}, w) &= \alpha \max_{y \in \mathcal{Y}} (y \cdot w) \\ &+ (1-\alpha) \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} M(i; \mathcal{Y}, w) \cdot w \end{aligned}$$

where the model-specific parameter $0 \leq \alpha \leq 1$ is the tradeoff between $\alpha = 1$, pruning all indices i not in the best parse, and $\alpha = 0$, pruning all indices with max-marginal value below the mean.

The threshold function has the important property that for any parse y, if $y \cdot w \geq t_\alpha(\mathcal{Y}, w)$ then $y(i) = 1$ implies $f(i) = 0$, i.e. if the parse score is above the threshold, then none of its indices will be pruned.

4.2 Filter Loss Training

The aim of our pruning models is to filter as many indices as possible without losing the gold parse. In structured prediction cascades, we incorporate this pruning goal into our training objective.

Let y be the gold output for a sentence. We define *filter loss* to be an indicator of whether any i with $y(i) = 1$ is filtered:

$$\Delta(y, \mathcal{Y}, w) = \mathbf{1}[\exists i \in y, M(i; \mathcal{Y}, w) \cdot w < t_\alpha(\mathcal{Y}, w)]$$

During training we minimize the expected filter loss using a standard structured SVM setup (Tsochantaridis et al., 2006). First we form a convex, continuous upper-bound of our loss function:

$$\begin{aligned} \Delta(y, \mathcal{Y}, w) &\leq \mathbf{1}[y \cdot w < t_\alpha(\mathcal{Y}, w)] \\ &\leq [1 - y \cdot w + t_\alpha(\mathcal{Y}, w)]_+ \end{aligned}$$

where the first inequality comes from the properties of max-marginals and the second is the standard hinge-loss upper-bound on an indicator.

Now assume that we have a corpus of P training sentences. Let the sequence $(y^{(1)}, \ldots, y^{(P)})$ be the gold parses for each sentences and the sequence $(\mathcal{Y}^{(1)}, \ldots, \mathcal{Y}^{(P)})$ be the set of possible output structures. We can form the regularized risk minimization for this upper bound of filter loss:

$$\min_w \lambda \|w\|^2 + \frac{1}{P} \sum_{p=1}^{P} [1 - y^{(p)} \cdot w + t_\alpha(\mathcal{Y}^{(p)}, w)]_+$$

This objective is convex and non-differentiable, due to the max inside t. We optimize using stochastic subgradient descent (Shalev-Shwartz et al., 2007). The stochastic subgradient at example p, $H(w, p)$ is 0 if $y^{(p)} - 1 \geq t_\alpha(\mathcal{Y}, w)$ otherwise,

$$\begin{aligned} H(w, p) &= \frac{2\lambda w}{P} - y^{(p)} + \alpha \arg \max_{y \in \mathcal{Y}^{(p)}} y \cdot w \\ &+ (1-\alpha) \frac{1}{|\mathcal{I}^{(p)}|} \sum_{i \in \mathcal{I}^{(p)}} M(i; \mathcal{Y}^{(p)}, w) \end{aligned}$$

Each step of the algorithm has an update of the form:

$$w_k = w_{k-1} - \eta_k H(w, p)$$

where η is an appropriate update rate for subgradient convergence. If $\alpha = 1$ the objective is identical to structured SVM with 0/1 hinge loss. For other values of α, the subgradient includes a term from the features of all max-marginal structures at each index. These feature counts can be computed using dynamic programming.

Setup	First-order				Second-order				Third-order			
	Speed	PE	Oracle	UAS	Speed	PE	Oracle	UAS	Speed	PE	Oracle	UAS
NoPrune	1.00	0.00	100	91.4	0.32	0.00	100	92.7	0.01	0.00	100	93.3
LengthDictionary	1.94	43.9	99.9	91.5	0.76	43.9	99.9	92.8	0.05	43.9	99.9	93.3
LocalShort	3.08	76.6	99.1	91.4	1.71	76.4	99.1	92.6	0.31	77.5	99.0	93.1
Local	4.59	89.9	98.8	91.5	2.88	83.2	99.5	92.6	1.41	89.5	98.8	93.1
FirstOnly	3.10	95.5	95.9	91.5	2.83	92.5	98.4	92.6	1.61	92.2	98.5	93.1
FirstAndSecond	-				-				1.80	97.6	97.7	93.1
VinePosterior	3.92	94.6	96.5	91.5	3.66	93.2	97.7	92.6	1.67	96.5	97.9	93.1
VineCascade	5.24	95.0	95.7	91.5	3.99	91.8	98.7	92.6	2.22	97.8	97.4	93.1
	k=8				k=16				k=64			
ZhangNivre	4.32	-	-	92.4	2.39	-	-	92.5	0.64	-	-	92.7

Table 1: Results comparing pruning methods on PTB Section 22. Oracle is the max achievable UAS after pruning. Pruning efficiency (PE) is the percentage of non-gold first-order dependency arcs pruned. Speed is parsing time relative to the unpruned first-order model (around 2000 tokens/sec). UAS is the unlabeled attachment score of the final parses.

4.3 1-Best Training

For the final pass, we want to train the model for 1-best output. Several different learning methods are available for structured prediction models including structured perceptron (Collins, 2002), max-margin models (Taskar et al., 2003), and log-linear models (Lafferty et al., 2001). In this work, we use the margin infused relaxed algorithm (MIRA) (Crammer and Singer, 2003; Crammer et al., 2006) with a hamming-loss margin. MIRA is an online algorithm with similar benefits as structured perceptron in terms of simplicity and fast training time. In practice, we found that MIRA with hamming-loss margin gives a performance improvement over structured perceptron and structured SVM.

5 Parsing Experiments

To empirically demonstrate the effectiveness of our approach, we compare our vine pruning cascade with a wide range of common pruning methods on the Penn WSJ Treebank (PTB) (Marcus et al., 1993). We then also show that vine pruning is effective across a variety of different languages.

For English, we convert the PTB constituency trees to dependencies using the Stanford dependency framework (De Marneffe et al., 2006). We then train on the standard PTB split with sections 2-21 as training, section 22 as validation, and section 23 as test. Results are similar using the Yamada and Matsumoto (2003) conversion. We additionally selected six languages from the CoNLL-X shared task (Buchholz and Marsi, 2006) that cover a number of different language families: Bulgarian, Chinese, Japanese, German, Portuguese, and Swedish. We use the standard CoNLL-X training/test split and tune parameters with cross-validation.

All experiments use unlabeled dependencies for training and test. Accuracy is reported as unlabeled attachment score (UAS), the percentage of tokens with the correct head word. For English, UAS ignores punctuation tokens and the test set uses predicted POS tags. For the other languages we follow the CoNLL-X setup and include punctuation in UAS and use gold POS tags on the set set. Speed-ups are given in terms of time relative to a highly optimized C++ implementation. Our unpruned first-order baseline can process roughly two thousand tokens a second and is comparable in speed to the greedy shift-reduce parser of Nivre et al. (2004).

5.1 Models

Our parsers perform multiple passes over each sentence. In each pass we first construct a (pruned) hypergraph (Klein and Manning, 2005) and then perform feature computation and inference. We choose the highest α that produces a pruning error of no more than 0.2 on the validation set (typically $\alpha \approx 0.6$) to filter indices for subsequent rounds (similar to Weiss and Taskar (2010)). We compare a variety of pruning models:

LengthDictionary a deterministic pruning method that eliminates all arcs longer than the maximum length observed for each

head-modifier POS pair.

LOCAL an unstructured arc classifier that chooses indices from $\mathcal{I}_1$ directly without enforcing parse constraints. Similar to the quadratic-time filter from Bergsma and Cherry (2010).

LOCALSHORT an unstructured arc classifier that chooses indices from $\mathcal{I}_0$ directly without enforcing parse constraints. Similar to the linear-time filter from Bergsma and Cherry (2010).

FIRSTONLY a structured first-order model trained with filter loss for pruning.

FIRSTANDSECOND a structured cascade with first- and second-order pruning models.

VINECASCADE the full cascade with vine, first- and second-order pruning models.

VINEPOSTERIOR the vine parsing cascade trained as a CRF with L-BFGS (Nocedal and Wright, 1999) and using posterior probabilities for filtering instead of max-marginals.

ZHANGNIVRE an unlabeled reimplementation of the linear-time, k-best, transition-based parser of Zhang and Nivre (2011). This parser uses composite features up to third-order with a greedy decoding algorithm. The reimplementation is about twice as fast as their reported speed, but scores slightly lower.

We found LENGTHDICTIONARY pruning to give significant speed-ups in all settings and therefore always use it as an initial pass. The maximum number of passes in a cascade is five: dictionary, vine, first-, and second-order pruning, and a final third-order 1-best pass.[3] We tune the pruning thresholds for each round and each cascade separately. This is because we might be willing to do a more aggressive vine pruning pass if the final model is a first-order model, since these two models tend to often agree.

5.2 Features

For the non-pruning models, we use a standard set of features proposed in the discriminative graph-based dependency parsing literature (McDonald et al., 2005; Carreras, 2007; Koo and Collins, 2010).

[3]For the first-order parser, we found it beneficial to employ a reduced feature first-order pruner before the final model, i.e. the cascade has four rounds: dictionary, vine, first-order pruning, and first-order 1-best.

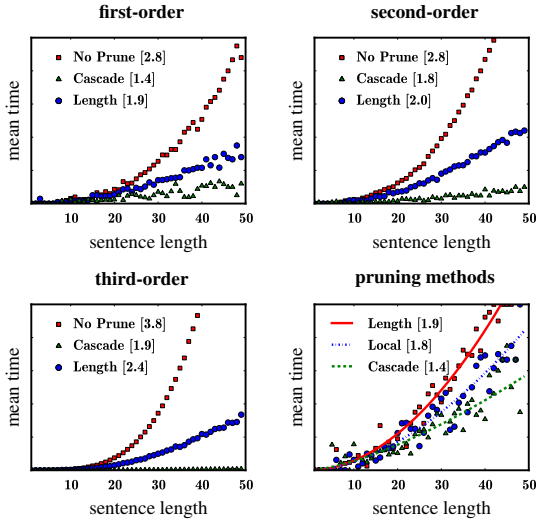

Figure 6: Mean parsing speed by sentence length for first-, second-, and third-order parsers as well as different pruning methods for first-order parsing. $[b]$ indicates the empirical complexity obtained from fitting ax^b.

Included are lexical features, part-of-speech features, features on in-between tokens, as well as feature conjunctions, surrounding part-of-speech tags, and back-off features. In addition, we replicate each part-of-speech (POS) feature with an additional feature using coarse POS representations (Petrov et al., 2012). Our baseline parsing models replicate and, for some experiments, surpass previous best results.

The first- and second-order pruning models have the same structure, but for efficiency use only the basic features from McDonald et al. (2005). As feature computation is quite costly, future work may investigate whether this set can be reduced further. VINEPRUNE and LOCALSHORT use the same feature sets for short arcs. Outer arcs have features of the unary head or modifier token, as well as features for the POS tag bordering the cutoff and the direction of the arc.

5.3 Results

A comparison between the pruning methods is shown in Table 1. The table gives relative speed-ups, compared to the unpruned first-order baseline, as well as accuracy, pruning efficiency, and oracle scores. Note particularly that the third-order cascade is twice as fast as an unpruned first-order model and >200 times faster than the unpruned third-order baseline. The comparison with poste-

Round	1-Best Model		
	First	Second	Third
Vine	37%	27%	16%
First	63%	30%	17%
Second	-	43%	18%
Third	-	-	49%

Table 2: Relative speed of pruning models in a multi-pass cascade. Note that the 1-best models use richer features than the corresponding pruning models.

rior pruning is less pronounced. Filter loss training is faster than VINEPOSTERIOR for first- and third-order parsing, but the two models have similar second-order speeds. It is also noteworthy that oracle scores are consistently high even after multiple pruning rounds: the oracle score of our third-order model for example is 97.4%.

Vine pruning is particularly effective. The vine pass is faster than both LOCAL and FIRSTONLY and prunes more effectively than LOCALSHORT. Vine pruning benefits from having a fast, linear-time model, but still maintaining enough structure for pruning. While our pruning approach does not provide any asymptotic guarantees, Figure 6 shows that in practice our multi-pass parser scales well even for long sentences: Our first-order cascade scales almost linearly with the sentence length, while the third-order cascade scales better than quadratic. Table 2 shows that the final pass dominates the computational cost, while each of the pruning passes takes up roughly the same amount of time.

Our second- and third-order cascades also significantly outperform ZHANGNIVRE. The transition-based model with $k = 8$ is very efficient and effective, but increasing the k-best list size scales much worse than employing multi-pass pruning. We also note that while direct speed comparison are difficult, our parser is significantly faster than the published results for other high accuracy parsers, e.g. Huang and Sagae (2010) and Koo et al. (2010).

Table 3 shows our results across a subset of the CoNLL-X datasets, focusing on languages that differ greatly in structure. The unpruned models perform well across datasets, scoring comparably to the top results from the CoNLL-X competition. We see speed increases for our cascades with almost no loss in accuracy across all languages, even for languages with fairly free word order like German. This is encouraging and suggests that the outer arcs of the vine-pruning model are able to cope with languages that are not as linear as English.

Setup		First-order		Second-order		Third-order	
		Speed	UAS	Speed	UAS	Speed	UAS
BG	B	1.90	90.7	0.67	92.0	0.05	92.1
	V	6.17	90.5	5.30	91.6	1.99	91.9
DE	B	1.40	89.2	0.48	90.3	0.02	90.8
	V	4.72	89.0	3.54	90.1	1.44	90.8
JA	B	1.77	92.0	0.58	92.1	0.04	92.4
	V	8.14	91.7	8.64	92.0	4.30	92.3
PT	B	0.89	90.1	0.28	91.2	0.01	91.7
	V	3.98	90.0	3.45	90.9	1.45	91.5
SW	B	1.37	88.5	0.45	89.7	0.01	90.4
	V	6.35	88.3	6.25	89.4	2.66	90.1
ZH	B	7.32	89.5	3.30	90.5	0.67	90.8
	V	7.45	89.3	6.71	90.3	3.90	90.9
EN	B	1.0	91.2	0.33	92.4	0.01	93.0
	V	5.24	91.0	3.92	92.2	2.23	92.7

Table 3: Speed and accuracy results for the vine pruning cascade across various languages. B is the unpruned baseline model, and V is the vine pruning cascade. The first section of the table gives results for the CoNLL-X test datasets for Bulgarian (BG), German (DE), Japanese (JA), Portuguese (PT), Swedish (SW), and Chinese (ZH). The second section gives the result for the English (EN) test set, PTB Section 23.

6 Conclusion

We presented a multi-pass architecture for dependency parsing that leverages vine parsing and structured prediction cascades. The resulting 200-fold speed-up leads to a third-order model that is twice as fast as an unpruned first-order model for a variety of languages, and that also compares favorably to a state-of-the-art transition-based parser. Possible future work includes experiments using cascades to explore much higher-order models.

Acknowledgments

We would like to thank the members of the Google NLP Parsing Team for comments, suggestions, bugfixes and help in general: Ryan McDonald, Hao Zhang, Michael Ringgaard, Terry Koo, Keith Hall, Kuzman Ganchev and Yoav Goldberg. We would also like to thank Andre Martins for showing that MIRA with hamming-loss margin performs better than other 1-best training algorithms.

References

S. Bergsma and C. Cherry. 2010. Fast and accurate arc filtering for dependency parsing. In *Proc. of COLING*, pages 53–61.

S. Buchholz and E. Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *CoNLL*.

X. Carreras, M. Collins, and T. Koo. 2008. Tag, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proc. of CoNLL*, pages 9–16.

X. Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proc. of CoNLL Shared Task Session of EMNLP-CoNLL*, volume 7, pages 957–961.

E. Charniak, M. Johnson, M. Elsner, J. Austerweil, D. Ellis, I. Haxton, C. Hill, R. Shrivaths, J. Moore, M. Pozar, et al. 2006. Multilevel coarse-to-fine PCFG parsing. In *Proc. of NAACL/HLT*, pages 168–175.

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

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

K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. 2006. Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 7:551–585.

M.C. De Marneffe, B. MacCartney, and C.D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proc. of LREC*, volume 6, pages 449–454.

J. Eisner and N.A. Smith. 2005. Parsing with soft and hard constraints on dependency length. In *Proc. of IWPT*, pages 30–41.

J. Eisner. 2000. Bilexical grammars and their cubic-time parsing algorithms. *Advances in Probabilistic and Other Parsing Technologies*, pages 29–62.

L. Huang and K. Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proc. of ACL*, pages 1077–1086.

D. Klein and C.D. Manning. 2005. Parsing and hypergraphs. *New developments in parsing technology*, pages 351–372.

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

T. Koo, A.M. Rush, M. Collins, T. Jaakkola, and D. Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *Proc. of EMNLP*, pages 1288–1298.

M. Kuhlmann, C. Gómez-Rodríguez, and G. Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *Proc. of ACL/HLT*, pages 673–682.

J. Lafferty, A. McCallum, and F.C.N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of ICML*, pages 282–289.

L. Lee. 2002. Fast context-free grammar parsing requires fast boolean matrix multiplication. *Journal of the ACM*, 49(1):1–15.

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

R. McDonald and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proc. of EACL*, volume 6, pages 81–88.

R. McDonald, K. Crammer, and F. Pereira. 2005. Online large-margin training of dependency parsers. In *Proc. of ACL*, pages 91–98.

J. Nivre, J. Hall, and J. Nilsson. 2004. Memory-based dependency parsing. In *Proc. of CoNLL*, pages 49–56.

J. Nocedal and S. J. Wright. 1999. *Numerical Optimization*. Springer.

A. Pauls and D. Klein. 2009. Hierarchical search for parsing. In *Proc. of NAACL/HLT*, pages 557–565.

S. Petrov and D. Klein. 2007. Improved inference for unlexicalized parsing. In *Proc. of NAACL/HLT*, pages 404–411.

S. Petrov, D. Das, and R. McDonald. 2012. A universal part-of-speech tagset. In *LREC*.

S. Petrov. 2009. *Coarse-to-Fine Natural Language Processing*. Ph.D. thesis, University of California at Bekeley, Berkeley, CA, USA.

B. Roark and K. Hollingshead. 2008. Classifying chart cells for quadratic complexity context-free inference. In *Proc. of COLING*, pages 745–751.

S. Shalev-Shwartz, Y. Singer, and N. Srebro. 2007. Pegasos: Primal estimated sub-gradient solver for svm. In *Proc. of ICML*, pages 807–814.

B. Taskar, C. Guestrin, and D. Koller. 2003. Max-margin markov networks. *Advances in neural information processing systems*, 16:25–32.

I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. 2006. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6(2):1453.

D. Weiss and B. Taskar. 2010. Structured prediction cascades. In *Proc. of AISTATS*, volume 1284, pages 916–923.

H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proc. of IWPT*, volume 3, pages 195–206.

Y. Zhang and J. Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proc. of ACL*, pages 188–193.