Generalized Higher-Order Dependency Parsing with Cube Pruning

Hao Zhang Ryan McDonald
Google, Inc.
{haozhang,ryanmcd}@google.com

Abstract

State-of-the-art graph-based parsers use features over higher-order dependencies that rely on decoding algorithms that are slow and difficult to generalize. On the other hand, transition-based dependency parsers can easily utilize such features without increasing the linear complexity of the shift-reduce system beyond a constant. In this paper, we attempt to address this imbalance for graph-based parsing by generalizing the Eisner (1996) algorithm to handle arbitrary features over higher-order dependencies. The generalization is at the cost of asymptotic efficiency. To account for this, cube pruning for decoding is utilized (Chiang, 2007). For the first time, label tuple and structural features such as valencies can be scored efficiently with third-order features in a graph-based parser. Our parser achieves the state-of-art unlabeled accuracy of 93.06% and labeled accuracy of 91.86% on the standard test set for English, at a faster speed than a reimplementation of the third-order model of Koo et al. (2010).

1 Introduction

The trade-off between rich features and exact decoding in dependency parsing has been well documented (McDonald and Nivre, 2007; Nivre and McDonald, 2008). Graph-based parsers typically trade-off rich feature scope for exact (or near exact) decoding, whereas transition-based parsers make the opposite trade-off. Recent research on both parsing paradigms has attempted to address this.

In the transition-based parsing literature, the focus has been on increasing the search space of the system at decoding time, as expanding the feature scope is often trivial and in most cases only leads to a constant-time increase in parser complexity. The most common approach is to use beam search (Duan et al., 2007; Johansson and Nugues, 2007; Titov and Henderson, 2007; Zhang and Clark, 2008; Zhang and Nivre, 2011), but more principled dynamic programming solutions have been proposed (Huang and Sagae, 2010). In all cases inference remains approximate, though a larger search space is explored.

In the graph-based parsing literature, the main thrust of research has been on extending the Eisner chart-parsing algorithm (Eisner, 1996) to incorporate higher-order features (McDonald and Pereira, 2006; Carreras, 2007; Koo and Collins, 2010). A similar line of research investigated the use of integer linear programming (ILP) formulations of parsing (Riedel and Clarke, 2006; Martins et al., 2009; Martins et al., 2010). Both solutions allow for exact inference with higher-order features, but typically at a high cost in terms of efficiency. Furthermore, specialized algorithms are required that deeply exploit the structural properties of the given model. Upgrading a parser to score new types of higher-order dependencies thus requires significant changes to the underlying decoding algorithm. This is in stark contrast to transition-based systems, which simply require the definition of new feature extractors.

In this paper, we abandon exact search in graph-based parsing in favor of freedom in feature scope. We propose a parsing algorithm that keeps the backbone Eisner chart-parsing algorithm for first-order parsing unchanged. Incorporating higher-order features only involves changing the scoring function of

potential parses in each chart cell by expanding the signature of each chart item to include all the non-local context required to compute features. The core chart-parsing algorithm remains the same regardless of which features are incorporated. To control complexity we use cube pruning (Chiang, 2007) with the beam size k in each cell. Furthermore, dynamic programming in the style of Huang and Sagae (2010) can be done by merging k-best items that are equivalent in scoring. Thus, our method is an application of integrated decoding with a language model in MT (Chiang, 2007) to dependency parsing, which has previously been applied to constituent parsing (Huang, 2008). However, unlike Huang, we only have one decoding pass and a single trained model, while Huang's constituent parser maintains a separate generative base model from a following discriminative re-ranking model. We draw connections to related work in Section 6.

Our chart-based approximate search algorithm allows for features on dependencies of an arbitrary order — as well as over non-local structural properties of the parse trees — to be scored at will. In this paper, we use first to third-order features of greater varieties than Koo and Collins (2010). Additionally, we look at higher-order dependency arc-label features, which is novel to graph-based parsing, though commonly exploited in transition-based parsing (Zhang and Nivre, 2011). This is because adding label tuple features would introduce a large constant factor of $O(|L|^3)$, where $|L|$ is the size of the label set L, into the complexity for exact third-order parsing. In our formulation, only the top-ranked labelled arcs would survive in each cell. As a result, label features can be scored without combinatorial explosion. In addition, we explore the use of valency features counting how many modifiers a word can have on its left and right side. In the past, only re-rankers on k-best lists of parses produced by a simpler model use such features due to the difficulty of incorporating them into search (Hall, 2007).

The final parser with all these features is both accurate and fast. In standard experiments for English, the unlabeled attachment score (UAS) is 93.06%, and the labeled attachment score (LAS) is 91.86%. The UAS score is state-of-art. The speed of our parser is 220 tokens per second, which is over 4 times faster than an exact third-order parser that attains UAS of 92.81% and comparable to the state-of-the-art transition-based system of Zhang and Nivre (2011) that employs beam search.

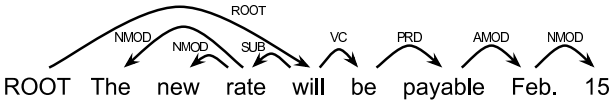

Figure 1: Example Sentence.

2 Graph-based Dependency Parsing

Dependency parsers produce directed relationships between *head* words and their syntactic *modifiers*. Each word modifies exactly one head, but can have any number of modifiers itself. The *root* of a sentence is a designated special symbol which all words in the sentence directly or indirectly modify. Thus, the dependency graph for a sentence is constrained to be a directed tree. The directed syntactic relationships, aka dependency arcs or dependencies for short, can often be labeled to indicate their syntactic role. Figure 1 gives an example dependency tree.

For a sentence $x = x_1 \dots x_n$, dependency parsing is the search for the set of head-modifier dependency arcs y^* such that $y^* = \operatorname{argmax}_{y \in \mathcal{Y}(x)} f(x, y)$, where f is a scoring function. As mentioned before, y^* must represent a directed tree. $|\mathcal{Y}(x)|$ is then the set of valid dependency trees for x and grows exponentially with respect to its length $|x|$. We further define L as the set of possible arc labels and use the notation $(i \xrightarrow{l} j) \in y$ to indicate that there is a dependency from head word x_i to modifier x_j with label l in dependency tree y.

In practice, $f(x, y)$ is factorized into scoring functions on parts of (x, y). For example, in first-order dependency parsing (McDonald et al., 2005), $f(x, y)$ is factored by the individual arcs:

$$y^* = \operatorname*{argmax}_{y \in \mathcal{Y}(x)} f(x, y) = \operatorname*{argmax}_{y \in \mathcal{Y}(x)} \sum_{(i \xrightarrow{l} j) \in y} f(i \xrightarrow{l} j)$$

The factorization of dependency structures into arcs enables an efficient dynamic programming algorithm with running time $O(|x|^3)$ (Eisner, 1996), for the large family of projective dependency structures.

Figure 2 shows the parsing logic for the Eisner algorithm. It has two types of dynamic programming states: *complete items* and *incomplete items*.

Complete items correspond to half-constituents, and are represented as triangles graphically. Incomplete items correspond to dependency arcs, and are represented as trapezoids. The Eisner algorithm is the basis for the more specialized variants of higher-order projective dependency parsing.

Second-order sibling models (McDonald and Pereira, 2006) score adjacent arcs with a common head. In order to score them efficiently, a new state corresponding to modifier pairs was introduced to the chart-parsing algorithm. Due to the careful factorization, the asymptotic complexity of the revised algorithm remains $O(|x|^3)$. The resulting scoring function is:

$$y^* = \underset{y \in \mathcal{Y}(x)}{\operatorname{argmax}} \sum_{(i \xrightarrow{l} j, i \xrightarrow{l'} k) \in y} f(i \xrightarrow{l} j, i \xrightarrow{l'} k)$$

where $(i \xrightarrow{l} j, i \xrightarrow{l'} k) \in y$ indicates two adjacent head-modifier relationships in dependency tree y, one from x_i to x_j with label l and another from x_i to x_k with label l'. Words x_j and x_k are commonly referred to as *siblings*. In order to maintain cubic parsing complexity, adjacent dependencies are scored only if the modifiers occur on the same side in the sentence relative to the head.

Second-order grandchild models (Carreras, 2007) score adjacent arcs in length-two head-modifier chains. For example, if word x_i modifies word x_j with label l, but itself has a dependency to modifier x_k with label l', then we would add a scoring function $f(j \xrightarrow{l} i \xrightarrow{l'} k)$. These are called *grandchild* models as they can score dependencies between a word and its modifier's modifiers, i.e., x_k is the grandchild of x_j in the above example. The states in the Eisner algorithm need to be augmented with the indices to the outermost modifiers in order to score the outermost grandchildren. The resulting algorithm becomes $O(|x|^4)$.

Finally, third-order models (Koo and Collins, 2010) score arc triples such as three adjacent sibling modifiers, called *tri-siblings*, or structures looking at both horizontal contexts and vertical contexts, e.g., *grand-siblings* that score a word, its modifier and its adjacent grandchildren. To accommodate the scorers for these sub-graphs, even more specialized dynamic programming states were introduced. The Koo and Collins (2010) factorization enables the scoring of certain types of third-order dependencies with $O(|x|^4)$ decoder run-time complexity.

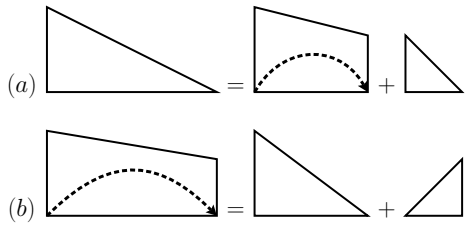

Figure 2: Structures and rules for parsing first-order models with the (Eisner, 1996) algorithm. This shows only the construction of right-pointing dependencies and not the symmetric case of left-pointing dependencies.

Each of these higher-order parsing algorithms makes a clever factorization for the specific model in consideration to keep complexity as low as possible. However, this results in a loss of generality.

3 Generalizing Eisner's Algorithm

In this section, we generalize the Eisner algorithm without introducing new parsing rules. The generalization is straight-forward: expand the dynamic programming state to incorporate feature histories. This is done on top of the two distinct chart items in the $O(|x|^3)$ Eisner chart-parsing algorithm (Figure 2). The advantage of this approach is that it maintains the simplicity of the original Eisner algorithm. Unfortunately, it can increase the run-time complexity of the algorithm substantially, but we will employ cube pruning to regain tractability. Because our higher-order dependency parsing algorithm is based the Eisner algorithm, it is currently limited to produce projective trees only.

3.1 Arbitrary n-th-order dependency parsing

We start with the simplest case of sibling models. If we want to score sibling arcs, at rule (b) in Figure 2, we can see that the complete item lying between the head and the modifier (the middle of the three items) does not contain information about the outermost modifier of the head, which is the previous dependency constructed and the sibling to the modifier of the dependency currently being constructed. This fact suggests that, in order to score modifier bigrams, the complete item states should be augmented by the outermost modifier. We can augment the chart items with such information, which

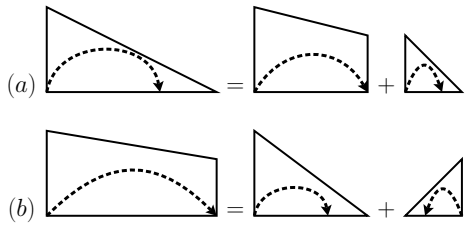

Figure 3: Structures and rules for parsing models based on modifier bigrams, with a generalized (Eisner, 1996) algorithm. Here the dashed arrows indicate additional information stored in each chart-cell. Specifically the previous modifier in complete chart items.

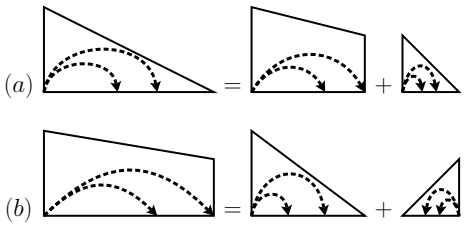

Figure 4: Structures and rules for parsing models based on modifier trigrams in horizontal contexts, with a generalized (Eisner, 1996) algorithm. Here the dashed arrows indicate the previous two modifiers to the head in each chart item.

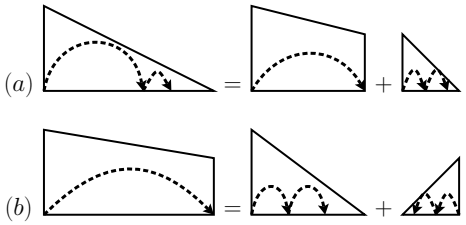

Figure 5: Structures and rules for parsing models based on modifier trigrams in vertical contexts, with a generalized (Eisner, 1996) algorithm. Here the dashed arrows indicate the modifier to the head and the modifier's modifier, forming a modifier chain of length two.

is shown in Figure 3. It refines the complete items by storing the previously constructed dependency to the outermost modifiers. Note that now the signature of the complete items is not simply the end-point indexes, but contains the index of the outer modifier.

Using this chart item augmentation it is now possible to score both first-order arcs as well as second-order sibling arcs. In fact, by symmetry, the new dynamic program can also score the leftmost and rightmost grandchildren of a head-modifier pair, in rule (a) and rule (b) respectively. By counting the number of free variables in each parsing rule, we see that the parsing complexity is $O(|x|^5)$, which is higher than both McDonald and Pereira (2006) and Carreras (2007). The added complexity comes from the fact that it is now possible to score a third-order dependency consisting of the head, the modifier, the sibling, and the outermost grandchild jointly.

We can go further to augment the complete and incomplete states with more parsing history. Figure 4 shows one possible next step of generalization. We generalize the states to keep track of the latest two modifiers of the head. As a result, it becomes possible to score tri-siblings involving three adjacent modifiers and grand-siblings involving two outermost grandchildren – both of which comprise the third-order Model 2 of Koo and Collins (2010) – plus potentially any additional interactions of these roles. Figure 5 shows another possible generalization. We keep modifier chains up to length two in the complete states. The added history enables the computation of features for great-grandchildren relationships: $(h \xrightarrow{l} m \xrightarrow{l'} gc \xrightarrow{l''} ggc)$.

In general, we can augment the complete and incomplete states with n variables representing the possible parsing histories and loop over the cross product of the histories in the innermost loop of Eisner algorithm. The cardinality of the cross product is $|x|^n \cdot |x|^n$. Thus, the complexity of the algorithm augmented by n variables is $O(|x|^3 \cdot |x|^{2n}) = O(|x|^{3+2n})$, where $n \geq 0$. Note that this complexity is for unlabeled parsing. A factor of $|L|$ for all or a subset of the encoded arcs must be multiplied in when predicting labeled parse structures.

3.2 History-based dependency parsing

The previous n modifiers, either horizontal or vertical, is a potential signature of parsing history. We can put arbitrary signatures of parsing history into the chart items so that when we score a new item, we can draw the distinguishing power of features based on an arbitrarily deep history. For example, consider the *position* of a modifier, which is the position in which it occurs amongst its siblings relative to the location of the head. We can store the position of the last modifier into both chart states. In complete states, this signature tells us the position of the outermost modifier, which is the valency of the head in the left or right half-constituent.

In the extreme case, we can use full subtrees as histories, although the cardinality of the set of histories would quickly become exponential, especially when one considers label ambiguity. Regardless, the high complexity associated with this generalization, even for second or third-order models, requires us to appeal to approximate search algorithms.

3.3 Advantage of the generalization

The complexity analysis earlier in this section reveals the advantage of such a generalization scheme. It factorizes a dynamic programming state for dependency parsing into two parts: 1) the structural state, which consists of the boundaries of incomplete and complete chart items, and accounts for the $O(|x|^3)$ term in the analysis, and 2) the feature history, which is a signature of the internal content of a sub-parse and accounts for the $O(|x|^{2n})$ term. The rules of the deductive parsing system – the Eisner algorithm – stay the same as long as the structural representation is unchanged. To generalize the parser to handle richer features, one can simply enrich the feature signature and the scoring function without changing the structural state. A natural grouping of states follows where all sub-parses sharing the same chart boundaries are grouped together. This grouping will enable the cube pruning in Section 4 for approximate search.

There is another advantage of keeping the Eisner parsing logic unchanged: derivations one-to-one correspond to dependency parse trees. Augmenting the complete and incomplete states does not introduce spurious ambiguity. This grouping view is useful for proving this point. Introducing higher order features in each chart item will cause sub-derivations to be re-ranked only. As a result, the final Viterbi parse can differ from the one from the standard Eisners algorithm. But the one-to-one correspondence still holds.

4 Approximate Search with Cube Pruning

In machine translation decoding, an n-gram language model can be incorporated into a translation model by augmenting the dynamic programming states for the translation model with the boundary $n-1$ words on the target side. The complexity for exact search involves a factor of $|x|^{4n-4}$ in the hierarchical phrase-based model of Chiang (2007), where $|x|$ is the input sentence length. The standard technique is to force a beam size k on each translation state so that the possible combinations of language model histories is bounded by k^2. Furthermore, if the list of k language model states are sorted from the lowest cost to the highest cost, we can assume the best combinations will still be among the combinations of the top items from each list, although the incorporation of n-gram features breaks the monotonic property of the underlying semi-ring.

Cube pruning is based on this approximation (Chiang, 2007). It starts with the combination of the top items in the lists to be combined. At each step, it puts the neighbors of the current best combination, which consists of going one position down in one of the k-best lists, into a priority queue. The algorithm stops when k items have been popped off from the queue. At the final step, it sorts the popped items since they can be out-of-order. It reduces the combination complexity from $O(k^2)$ to $O(k \cdot \log(k))$.

Our history-augmented parsing is analogous to MT decoding. The possible higher-order histories can similarly be limited to at most k in each complete or incomplete item. The core loop of the generalized algorithm which has a complexity of $O(|x|^{2n})$ can similarly be reduced to $O(k \cdot \log(k))$. Therefore, the whole parsing algorithm remains $O(|x|^3)$ regardless how deep we look into parsing history. Figure 6 illustrates the computation. We apply rule (b) to combine two lists of augmented complete items and keep the combinations with the highest model scores. With cube pruning, we only explore cells at $(0,0)$, $(0,1)$, $(1,0)$, $(2,0)$, and $(1,1)$, without the need to evaluate scoring functions for the remaining cells in the table. Similar computation happens with rule (a).

In this example cube pruning does find the highest scoring combination, i.e., cell $(1,1)$. However, note that the scores are not monotonic in the order in which we search these cells as non-local features are used to score the combinations. Thus, cube pruning may not find the highest scoring combination. This approximation is at the heart of cube pruning.

4.1 Recombination

The significance of using feature signatures is that when two combinations result in a state with the

identical feature signature the one with the highest score survives. This is the core principle of dynamic programming. We call it *recombination*. It denotes the same meaning as *state-merging* in Huang and Sagae (2010) for transition-based parsers.

In cube pruning, with recombination, the k-best items in each chart cell are locally optimal (in the pruned search space) over all sub-trees with an equivalent state for future combinations. The cube pruning algorithm without recombination degenerates to a recursive k-best re-scoring algorithm since each of the k-best items would be unique by itself as a sub-tree. It should be noted that by working on a chart (or a forest, equivalently) the algorithm is already applying recombination at a coarser level.

In machine translation, due to its large search space and the abstract nature of an n-gram language model, it is more common to see many sub-trees with the same language model feature signature, making recombination crucial (Chiang, 2007). In constituent parser reranking (Huang, 2008), recombination is less likely to happen since the reranking features capture peculiarities of local tree structures. For dependency parsing, we hypothesize that the higher-order features are more similar to the n-gram language model features in MT as they tend to be common features among many sub-trees. But as the feature set becomes richer, recombination tends to have a smaller effect. We will discuss the empirical results on recombination in Section 5.4.

5 Experiments

We define the scoring function $f(x, y)$ as a linear classifier between a vector of features and a corresponding weight vector, i.e., $f(x, y) = w \cdot \phi(x, y)$. The feature function ϕ decomposes with respect to scoring function f. We train the weights to optimize the first-best structure. We use the max-loss variant of the margin infused relaxed algorithm (MIRA) (Crammer et al., 2006) with a hamming-loss margin as is common in the dependency parsing literature (Martins et al., 2009; Martins et al., 2010). MIRA only requires a first-best decoding algorithm, which in our case is the approximate chart-based parsing algorithms defined in Sections 3 and 4. Because our decoding algorithm is approximate, this may lead to invalid updates given to the optimizer (Huang and Fayong, 2012). However, we found that ignoring or modifying such updates led to negligible differences in practice. In all our experiments, we train MIRA for 8 epochs and use a beam of $k = 5$ during decoding. Both these values were determined on the English development data.

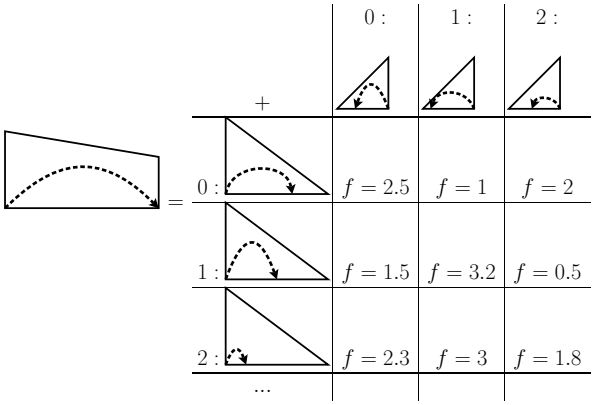

Figure 6: Combining two lists of complete items with cube pruning.

5.1 Features

The feature templates we use are drawn from the past work on graph-based parsing and transition-based parsing. The base templates for the higher-order dependencies are close to Koo and Collins (2010), with the major exception that our features include label-tuple information. The basic features include identities, part of speech tags, and labels of the words in dependency structures. These atomic features are conjoined with the directions of arcs to create composite n-gram features. The higher-order dependency features can be categorized into the following sub-groups, where we use h to indicate the head, m the modifier, s the modifier's sibling and gc a grandchild word in a dependency part.

- (labeled) modifier features: $(h \xrightarrow{l} m)$
- (labeled) sibling features: $(h \xrightarrow{l} m, h \xrightarrow{l'} s)$
- (labeled) outermost grandchild features: $(h \xrightarrow{l} m \xrightarrow{l'} gc)$
- (labeled) tri-sibling features: $(h \xrightarrow{l} m, h \xrightarrow{l'} s, h \xrightarrow{l''} s_2)$
- (labeled) grand-sibling features: $(h \xrightarrow{l} m \xrightarrow{l'} gc, h \xrightarrow{l} m \xrightarrow{l''} gc_2)$,

- (labeled) sibling and grandchild conjoined features: $(h \xrightarrow{l} m, h \xrightarrow{l'} s, m \xrightarrow{l''} gc)$

The general history features include valencies of words conjoined with the directions of the dominating arcs. The positions of the modifiers are also conjoined with the higher-order dependency features in the previous list.

The features that are new compared to Koo and Collins (2010) are the label tuple features, the sibling and grandchild conjoined features, and the valency features. We determine this feature set based on experiments on the development data for English. In Section 5.3 we examine the impact of these new features on parser performance.

5.2 Main Results

Our first set of results are on English dependencies. We used the Penn WSJ Treebank converted to dependencies with Penn2Malt[1] conversion software specifying Yamada and Matsumoto head rules and Malt label set. We used the standard splits of this data: sections 2-21 for training; section 22 for validation; and section 23 for evaluation. We evaluated our parsers using standard labeled accuracy scores (LAS) and unlabeled accuracy scores (UAS) excluding punctuation. We report run-times in tokens per second. Part-of-speech tags are predicted as input using a linear-chain CRF.

Results are given in Table 1. We compare our method to a state-of-the-art graph-based parser (Koo and Collins, 2010) as well as a state-of-the-art transition-based parser that uses a beam (Zhang and Nivre, 2011) and the dynamic programming transition-based parser of Huang and Sagae (2010). Additionally, we compare to our own implementation of exact first to third-order graph-based parsing and the transition-based system of Zhang and Nivre (2011) with varying beam sizes.

There are a number of points to make. First, approximate decoding with rich features and cube pruning gives state-of-the-art labeled and unlabeled parsing accuracies relative to previously reported results. This includes the best graph-based parsing results of Koo and Collins (2010), which has near identical performance, as well as the best beam-based and dynamic-programming-based transition parsers (Huang and Sagae, 2010; Zhang and Nivre, 2011). Second, at a similar toks/sec parser speed, our method achieves better performance than the transition-based model of Zhang and Nivre (2011) with a beam of 256. Finally, compared to an implementation of an exact third-order parser – which provides us with an apples-to-apples comparison in terms of features and runtime – approximate decoding with cube pruning is both more accurate and while being 4-5 times as fast. It is more accurate as it can easily incorporate more complex features and it is faster since its asymptotic complexity is lower. We should point out that our third-order reimplementation is a purely unlabeled parser as we do not have an implementation of an exact labeled third-order parser. This likely under estimates its accuracy, but also significantly overestimates its speed.

Parser	*UAS*	*LAS*	*Toks/Sec*
Huang and Sagae (2010)	92.1-	-	-
Zhang and Nivre (2011)	92.9-	91.8-	-
Zhang and Nivre (reimpl.) (beam=64)	92.73	91.67	760
Zhang and Nivre (reimpl.) (beam=256)	92.75	91.71	190
Koo and Collins (2010)	93.04	-	-
1^{st}-order exact (reimpl.)	91.80	90.50	2070
2^{nd}-order exact (reimpl.)	92.40	91.12	1110
3^{rd}-order exact (reimpl.)	92.81	-†	50
this paper	93.06	91.86	220

Table 1: Comparing this work in terms of parsing accuracy compared to state-of-the-art baselines on the English test data. We also report results for a re-implementation of exact first to third-order graph-based parsing and a re-implementation of Zhang and Nivre (2011) in order to compare parser speed. †Our exact third-order implementation currently only supports unlabeled parsing.

Next, we looked at the impact of our system on non-English treebanks. Specifically we focused on two sets of data. The first is the Chinese Treebank converted to dependencies. Here we use the identical training/validation/evaluation splits and experimental set-up as Zhang and Nivre (2011). Additionally, we evaluate our system on eight other languages from the CoNLL 2006/2007 shared-task (Buchholz and Marsi, 2006; Nivre et al., 2007). We selected the following four data sets since they are primarily projective treebanks (<1.0% non-projective arcs): Bulgarian and Spanish from CoNLL 2006 as well as Catalan and Italian from CoNLL 2007. Currently our method is restricted to predicting strictly projective trees as it

[1] http://w3.msi.vxu.se/∼nivre/research/Penn2Malt.html

uses the Eisner chart parsing algorithm as its backbone. We also report results from four additional CoNLL data sets reported in Rush and Petrov (2012) in order to directly compare accuracy. These are German, Japanese, Portuguese and Swedish. For all data sets we measure UAS and LAS excluding punctuation and use gold tags as input to the parser as is standard for these data sets.

Results are given in Table 2. Here we compare to our re-implementations of Zhang and Nivre (2011), exact first to third-order parsing and Rush and Petrov (2012) for the data sets in which they reported results. We again see that approximate decoding with rich features and cube pruning has higher accuracy than transition-based parsing with a large beam. In particular, for the ZH-CTB data set, our system is currently the best reported result. Furthermore, our system returns comparable accuracy with exact third-order parsing, while being significantly faster and more flexible.

5.3 Ablation studies

In this section, we analyze the contributions from each of the feature groups. Each row in Table 3 uses a super set of features than the previous row. All systems use our proposed generalized higher-order parser with cube-pruning. I.e., they are all using the Eisner chart-parsing algorithm with expanded feature signatures. The only difference between systems is the set of features used. This allows us to see the improvement from additional features.

The first row uses no higher-order features. It is equivalent to the first-order model from Table 1. The only difference is that it uses the k-best algorithm to find the first-best, so it has additional overhead compared to the standard Viterbi algorithm. Each of the following rows gets a higher accuracy than its previous row by adding more higher order features. Putting in the sibling and grandchild conjoined features and the valency features yields a further improvement over the approximation of Koo and Collins (2010). Thus, the addition of new higher-order features, including valency, extra third-order, and label tuple features, results in increased accuracy. However, this is not without cost as the run-time in terms of tokens/sec decreases (300 to 220). But this decrease is not asymptotic, as it would be if one were to exactly search over our final model with these additional features, e.g., valency would at least multiply an additional $O(n)$ factor.

Higher-order Features	*UAS*	*LAS*	*Toks/Sec*
none	91.74	90.46	1510
McDonald (2006) features + labels	92.48	91.25	860
Carreras (2007) features + labels	92.85	91.66	540
Koo (2010) features + labels	92.92	91.75	300
all features	93.06	91.86	220

Table 3: Generalized higher-order parsing with cube pruning using different feature sets.

Beam	*Recombination*	*UAS*	*LAS*	*Toks/Sec*
2	no	92.86	91.63	280
2	yes	92.89	91.65	260
5	no	93.05	91.85	240
5	yes	93.06	91.86	230
10	yes	93.05	91.85	140

Table 4: Showing the effect of better search on accuracy and speed on the English test data with a fixed model.

5.4 Impact of Search Errors

Since our decoding algorithm is not exact, it could return sub-optimal outputs under the current model. We analyze the effect of search errors on accuracies in Table 4. We vary the beam size at each cell and switch the option for signature-based recombination to make search better or worse to see how much impact it has on the final accuracy.

The results indicate that a relatively small per-cell beam is good enough. Going from a beam of 2 to 5 increases accuracy notably, but going to a larger beam size has little effect but at a cost in terms of efficiency. This suggests that most of the parser ambiguity is represented in the top-5 feature signatures at each chart cell. Furthermore, recombination does help slightly, but more so at smaller beam sizes.

If we keep the beam size constant but enlarge the feature scope from second-order to third-order, one would expect more search errors to occur. We measured this empirically by computing the number of sentences where the gold tree had a higher model score than the predicted tree in the English evaluation data. Indeed, larger feature scopes do lead to more search errors, but the absolute number of search errors is usually quite small – there are only 19 search errors using second-order features and 32 search errors using third-order plus valency features out of 2416 English test sentences. Part of the reason for this is that there are only 12 la-

Language	Zhang and Nivre (reimpl.) (beam=64)	Zhang and Nivre (reimpl.) (beam=256)	Rush and Petrov‡	1^{st}-order exact (reimpl.)	2^{nd}-order exact (reimpl.)	3^{rd}-order exact (reimpl.)	this paper
BG-CONLL	92.22 / 87.87	92.28 / 87.91	91.9- / -	91.98 / 87.13	93.02 / 88.13	92.96 / -	**93.08** / **88.23**
CA-CONLL	93.76 / 87.74	93.83 / 87.85		92.83 / 86.22	93.45 / 87.19	**94.07** / -	94.00 / **88.08**
DE-CONLL	89.18 / 86.50	88.94 / 86.58	90.8- / -	89.28 / 86.06	90.87 / 87.72	91.29 / -	**91.35** / **88.42**
ES-CONLL	86.64 / 83.25	86.62 / 83.11		85.35 / 81.53	86.80 / 82.91	87.26 / -	**87.48** / **84.05**
IT-CONLL	85.51 / 81.12	85.45 / 81.10		84.98 / 80.23	85.46 / 80.66	86.49 / -	**86.54** / **82.15**
JA-CONLL	92.70 / 91.03	92.76 / 91.09	92.3- / -	93.00 / 91.03	93.20 / 91.25	**93.36** / -	93.24 / **91.45**
PT-CONLL	91.32 / 86.98	91.28 / 86.88	91.5- / -	90.36 / 85.77	91.36 / 87.22	91.66 / -	**91.69** / **87.70**
SV-CONLL	90.84 / 85.30	91.00 / **85.42**	90.1- / -	89.32 / 82.06	90.50 / 83.01	90.32 / -	**91.44** / 84.58
ZH-CTB	86.04 / 84.48†	86.14 / 84.57		84.38 / 82.62	86.63 / 84.95	86.77 / -	**86.87** / **85.19**
AVG	89.80 / 86.03	89.81 / 86.06		89.05 / 84.74	90.14 / 85.89	90.46 / -	**90.63** / **86.65**

Table 2: UAS/LAS for experiments on non-English treebanks. Numbers in bold are the highest scoring system. Zhang and Nivre is a reimplementation of Zhang and Nivre (2011) with beams of size 64 and 256. Rush and Petrov are the UAS results reported in Rush and Petrov (2012). N^{th}-order exact are implementations of exact 1st-3rd order dependency parsing. †For reference, Zhang and Nivre (2011) report 86.0/84.4, which is previously the best result reported on this data set. ‡It should be noted that Rush and Petrov (2012) do not jointly optimize labeled and unlabeled dependency structure, which we found to often help. This, plus extra features, accounts for the differences in UAS.

bels in the Penn2Malt label set, which results in little non-structural ambiguity. In contrast, Stanford-style dependencies contain a much larger set of labels (50) with more fine-grained syntactic distinctions (De Marneffe et al., 2006). Training and testing a model using this dependency representation[2] increases the number of search errors of the full model to 126 out 2416 sentences. But that is still only 5% of all sentences and significantly smaller when measured per dependency.

6 Related Work

As mentioned in the introduction, there has been numerous studies on trying to reconcile the rich-features versus exact decoding trade-off in dependency parsing. In the transition-based parsing literature this has included the use of beam search to increase the search space (Duan et al., 2007; Johansson and Nugues, 2007; Titov and Henderson, 2007; Zhang and Clark, 2008; Zhang and Nivre, 2011). Huang and Sagae (2010) took a more principled approach proposing a method combining shift-reduce parsing with dynamic programming. They showed how feature signatures can be compiled into dynamic programming states and how best-first search can be used to find the optimal transition sequence. However, when the feature scope becomes large, then the state-space and resulting search space can be either intractable or simply non-practical to explore. Thus, they resort to an approximate beam search that still exploring an exponentially-larger space than greedy or beam-search transition-based systems. One can view the contribution in this paper as being the complement of the work of Huang and Sagae (2010) for graph-based systems. Our approach also uses approximate decoding in order to exploit arbitrary feature scope, while still exploring an exponentially-large search space. The primary difference is how the system is parameterized, over dependency sub-graphs or transitions. Another critical difference is that a chart-based algorithm, though still subject to search errors, is less likely to be hindered by an error made at one word position because it searches over many parallel alternatives in a bottom-up search as opposed to a left-to-right pass.

In the graph-based parsing literature, exact parsing algorithms for higher-order features have been studied extensively (McDonald and Pereira, 2006; Carreras, 2007; Koo and Collins, 2010), but at a high computational cost as increasing the order of a model typically results in an asymptotic increase in

[2]This model gets 90.4/92.8 LAS/UAS which is comparable to the UAS of 92.7 reported by Rush and Petrov (2012).

running time. ILP formulations of parsing (Riedel and Clarke, 2006; Martins et al., 2009; Martins et al., 2010) also allow for exact inference with higher-order features, but again at a high computational cost as ILP's have, in the worst-case, exponential run-time with respect to the sentence length. Studies that have abandoned exact inference have focused on sampling (Nakagawa, 2007), belief propagation (Smith and Eisner, 2008), Lagrangian relaxation (Koo et al., 2010; Martins et al., 2011), and more recently structured prediction cascades (Weiss and Taskar, 2010; Rush and Petrov, 2012). However, these approximations themselves are often computationally expensive, requiring multiple decoding/sampling stages in order to produce an output. All the methods above, both exact and approximate, require specialized algorithms for every new feature that is beyond the scope of the previous factorization. In our method, the same parsing algorithm can be utilized (Eisner's + cube pruning) just with slight different feature signatures.

Our proposed parsing model draws heavily on the work of Huang (2008). Huang introduced the idea of "forest rescoring", which uses cube pruning to enable the incorporation of non-local features into a constituency parsing model providing state-of-the art performance. This paper is the extension of such ideas to dependency parsing, also giving state-of-the-art results. An important difference between our formulation and forest rescoring is that we only have one decoding pass and a single trained model, while forest rescoring, as formulated by Huang (2008), separates a generative base model from a following discriminative re-ranking model. Hence, our formulation is more akin to the one pass decoding algorithm of Chiang (2007) for integrated decoding with a language model in machine translation. This also distinguishes it from previous work on dependency parse re-ranking (Hall, 2007) as we are not re-ranking/re-scoring the output of a base model but using a single decoding algorithm and learned model at training and testing.

This work is largely orthogonal to other attempts to speed up chart parsing algorithms. This includes work on coarse-to-fine parsing (Charniak and Johnson, 2005; Petrov and Klein, 2007; Rush and Petrov, 2012), chart-cell closing and pruning (Roark and Hollingshead, 2008; Roark and Hollingshead, 2009), and dynamic beam-width prediction (Bodenstab et al., 2011). Of particular note, Rush and Petrov (2012) report run-times far better than our cube pruning system. At the heart of their system is a linear time vine-parsing stage that prunes most of the search space before higher-order parsing. This effectively makes their final system linear time in practice as the higher order models have far fewer parts to consider. One could easily use the same first-pass pruner in our cube-pruning framework.

In our study we use cube pruning only for decoding and rely on inference-based learning algorithms to train model parameters. Gimpel and Smith (2009) extended cube pruning concepts to partition-function and marginal calculations, which would enable the training of probabilistic graphical models.

Finally, due to its use of the Eisner chart-parsing algorithm as a backbone, our model is fundamentally limited to predicting projective dependency structures. Investigating extensions of this work to the non-projective case is an area of future study. Work on defining bottom-up chart-parsing algorithms for non-projective dependency trees could potentially serve as a mechanism to solving this problem (Gómez-Rodríguez et al., 2009; Kuhlmann and Satta, 2009; Gómez-Rodríguez et al., 2010).

7 Conclusion

In this paper we presented a method for generalized higher-order dependency parsing. The method works by augmenting the dynamic programming signatures of the Eisner chart-parsing algorithm and then controlling complexity via cube pruning. The resulting system has the flexibility to incorporate arbitrary feature history while still exploring an exponential search space efficiently. Empirical results show that the system gives state-of-the-art accuracies across numerous data sets while still maintaining practical parsing speeds – as much as 4-5 times faster than exact third-order decoding.

Acknowledgments: We would like to thank Sasha Rush and Slav Petrov for help modifying their hypergraph parsing code. We would also like to thank the parsing team at Google for providing interesting discussions and new ideas while we conducted this work, as well as comments on earlier drafts of the paper.

References

N. Bodenstab, A. Dunlop, K. Hall, and B. Roark. 2011. Beam-width prediction for efficient context-free parsing. In *Proc. ACL*.

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

X. Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proc. of the CoNLL Shared Task Session of EMNLP-CoNLL*.

E. Charniak and M. Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proc. ACL*.

D. Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2).

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

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

X. Duan, J. Zhao, and B. Xu. 2007. Probabilistic parsing action models for multi-lingual dependency parsing. In *Proc. of EMNLP-CoNLL*.

J. Eisner. 1996. Three new probabilistic models for dependency parsing: an exploration. In *Proc. of COLING*.

K. Gimpel and N.A. Smith. 2009. Cube summing, approximate inference with non-local features, and dynamic programming without semirings. In *Proc. EACL*.

C. Gómez-Rodríguez, M. Kuhlmann, G. Satta, and D. Weir. 2009. Optimal reduction of rule length in linear context-free rewriting systems. In *Proc. NAACL*.

C. Gómez-Rodríguez, M. Kuhlmann, and G. Satta. 2010. Efficient parsing of well-nested linear context-free rewriting systems. In *Proc. NAACL*.

K. Hall. 2007. K-best spanning tree parsing. In *Proc. of ACL*.

L. Huang and S. Fayong. 2012. Structured perceptron with inexact search. In *Proc. of NAACL*.

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

L. Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proc. of ACL*.

R. Johansson and P. Nugues. 2007. Incremental dependency parsing using online learning. In *Proc. of EMNLP-CoNLL*.

T. Koo and M. Collins. 2010. Efficient third-order dependency parsers. In *Proc. of ACL*.

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

M. Kuhlmann and G. Satta. 2009. Treebank grammar techniques for non-projective dependency parsing. In *Proc. EACL*.

A. F. T. Martins, N. Smith, and E. P. Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proc. of ACL*.

A. F. T. Martins, N. Smith, E. P. Xing, P. M. Q. Aguiar, and M. A. T. Figueiredo. 2010. Turbo parsers: Dependency parsing by approximate variational inference. In *Proc. of EMNLP*.

A. F. T. Martins, N. Smith, M. A. T. Figueiredo, and P. M. Q. Aguiar. 2011. Dual decomposition with many overlapping components. In *Proc of EMNLP*.

R. McDonald and J. Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proc. of EMNLP-CoNLL*.

R. McDonald and F. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proc. of EACL*.

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

T. Nakagawa. 2007. Multilingual dependency parsing using global features. In *Proc. of EMNLP-CoNLL*.

J. Nivre and R. McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proc. of ACL*.

J. Nivre, J. Hall, S. Kübler, R. McDonald, J. Nilsson, S. Riedel, and D. Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proc. of EMNLP-CoNLL*.

S. Petrov and D. Klein. 2007. Improved inference for unlexicalized parsing. In *Proc. NAACL*.

S. Riedel and J. Clarke. 2006. Incremental integer linear programming for non-projective dependency parsing. In *Proc. of EMNLP*.

B. Roark and K. Hollingshead. 2008. Classifying chart cells for quadratic complexity context-free inference. In *Proc. COLING*.

B. Roark and K. Hollingshead. 2009. Linear complexity context-free parsing pipelines via chart constraints. In *Proce. NAACL*.

A. Rush and S. Petrov. 2012. Efficient multi-pass dependency pruning with vine parsing. In *Proc. of NAACL*.

D. Smith and J. Eisner. 2008. Dependency parsing by belief propagation. In *Proc. of EMNLP*.

I. Titov and J. Henderson. 2007. Fast and robust multilingual dependency parsing with a generative latent variable model. In *Proc. of EMNLP-CoNLL*.

D. Weiss and B. Taskar. 2010. Structured prediction cascades. In *Proc. of AISTATS*.

Y. Zhang and S. Clark. 2008. A Tale of Two Parsers: Investigating and Combining Graph-based and Transition-based Dependency Parsing. In *Proc. of EMNLP*.

Y. Zhang and J. Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proc. of ACL-HLT*, volume 2.