Efficient Third-order Dependency Parsers

Terry Koo and Michael Collins
MIT CSAIL, Cambridge, MA, 02139, USA
{maestro,mcollins}@csail.mit.edu

Abstract

We present algorithms for higher-order dependency parsing that are "third-order" in the sense that they can evaluate substructures containing three dependencies, and "efficient" in the sense that they require only $O(n^4)$ time. Importantly, our new parsers can utilize both sibling-style and grandchild-style interactions. We evaluate our parsers on the Penn Treebank and Prague Dependency Treebank, achieving unlabeled attachment scores of 93.04% and 87.38%, respectively.

1 Introduction

Dependency grammar has proven to be a very useful syntactic formalism, due in no small part to the development of efficient parsing algorithms (Eisner, 2000; McDonald et al., 2005b; McDonald and Pereira, 2006; Carreras, 2007), which can be leveraged for a wide variety of learning methods, such as feature-rich discriminative models (Lafferty et al., 2001; Collins, 2002; Taskar et al., 2003). These parsing algorithms share an important characteristic: they *factor* dependency trees into sets of *parts* that have limited interactions. By exploiting the additional constraints arising from the factorization, maximizations or summations over the set of possible dependency trees can be performed efficiently and exactly.

A crucial limitation of factored parsing algorithms is that the associated parts are typically quite small, losing much of the contextual information within the dependency tree. For the purposes of improving parsing performance, it is desirable to increase the size and variety of the parts used by the factorization.[1] At the same time, the need for more expressive factorizations must be balanced against any resulting increase in the computational cost of the parsing algorithm. Consequently, recent work in dependency parsing has been restricted to applications of second-order parsers, the most powerful of which (Carreras, 2007) requires $O(n^4)$ time and $O(n^3)$ space, while being limited to second-order parts.

In this paper, we present new third-order parsing algorithms that increase both the size and variety of the parts participating in the factorization, while simultaneously maintaining computational requirements of $O(n^4)$ time and $O(n^3)$ space. We evaluate our parsers on the Penn WSJ Treebank (Marcus et al., 1993) and Prague Dependency Treebank (Hajič et al., 2001), achieving unlabeled attachment scores of 93.04% and 87.38%. In summary, we make three main contributions:

1. Efficient new third-order parsing algorithms.
2. Empirical evaluations of these parsers.
3. A free distribution of our implementation.[2]

The remainder of this paper is divided as follows: Sections 2 and 3 give background, Sections 4 and 5 describe our new parsing algorithms, Section 6 discusses related work, Section 7 presents our experimental results, and Section 8 concludes.

2 Dependency parsing

In dependency grammar, syntactic relationships are represented as head-modifier dependencies: directed arcs between a *head*, which is the more "essential" word in the relationship, and a *modifier*, which supplements the meaning of the head. For example, Figure 1 contains a dependency between the verb "report" (the head) and its object "sales" (the modifier). A complete analysis of a sentence is given by a dependency tree: a set of dependencies that forms a rooted, directed tree spanning the words of the sentence. Every dependency tree is rooted at a special "*" token, allowing the

[1] For examples of how performance varies with the degree of the parser's factorization see, e.g., McDonald and Pereira (2006, Tables 1 and 2), Carreras (2007, Table 2), Koo et al. (2008, Tables 2 and 4), or Suzuki et al. (2009, Tables 3–6).

[2] http://groups.csail.mit.edu/nlp/dpo3/

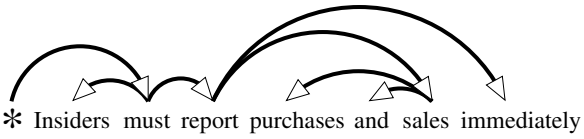


Figure 1: An example dependency structure.

selection of the sentential head to be modeled as if it were a dependency.

For a sentence $\boldsymbol{x}$, we define dependency parsing as a search for the highest-scoring analysis of $\boldsymbol{x}$:

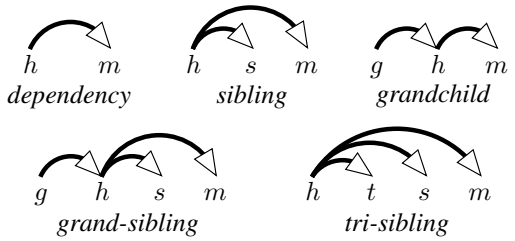
$$y^*(\boldsymbol{x}) = \operatorname*{argmax}_{y \in \mathcal{Y}(\boldsymbol{x})} \textsc{Score}(\boldsymbol{x}, y) \quad (1)$$

Here, $\mathcal{Y}(\boldsymbol{x})$ is the set of all trees compatible with $\boldsymbol{x}$ and $\textsc{Score}(\boldsymbol{x}, y)$ evaluates the event that tree y is the analysis of sentence $\boldsymbol{x}$. Since the cardinality of $\mathcal{Y}(\boldsymbol{x})$ grows exponentially with the length of the sentence, directly solving Eq. 1 is impractical. A common strategy, and one which forms the focus of this paper, is to *factor* each dependency tree into small *parts*, which can be scored in isolation. Factored parsing can be formalized as follows:

$$\textsc{Score}(\boldsymbol{x}, y) = \sum_{p \in y} \textsc{ScorePart}(\boldsymbol{x}, p)$$

That is, we treat the dependency tree y as a set of parts p, each of which makes a separate contribution to the score of y. For certain factorizations, efficient parsing algorithms exist for solving Eq. 1.

We define the *order* of a part according to the number of dependencies it contains, with analogous terminology for factorizations and parsing algorithms. In the remainder of this paper, we focus on factorizations utilizing the following parts:



Specifically, Sections 4.1, 4.2, and 4.3 describe parsers that, respectively, factor trees into grandchild parts, grand-sibling parts, and a mixture of grand-sibling and tri-sibling parts.

3 Existing parsing algorithms

Our new third-order dependency parsers build on ideas from existing parsing algorithms. In this section, we provide background on two relevant parsers from previous work.

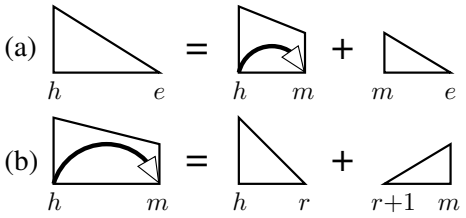


Figure 2: The dynamic-programming structures and derivations of the Eisner (2000) algorithm. Complete spans are depicted as triangles and incomplete spans as trapezoids. For brevity, we elide the symmetric right-headed versions.

3.1 First-order factorization

The first type of parser we describe uses a "first-order" factorization, which decomposes a dependency tree into its individual dependencies. Eisner (2000) introduced a widely-used dynamic-programming algorithm for first-order parsing; as it is the basis for many parsers, including our new algorithms, we summarize its design here.

The Eisner (2000) algorithm is based on two interrelated types of dynamic-programming structures: *complete* spans, which consist of a head-word and its descendents on one side, and *incomplete* spans, which consist of a dependency and the region between the head and modifier.

Formally, we denote a complete span as $C_{h,e}$ where h and e are the indices of the span's head-word and endpoint. An incomplete span is denoted as $I_{h,m}$ where h and m are the index of the head and modifier of a dependency. Intuitively, a complete span represents a "half-constituent" headed by h, whereas an incomplete span is only a partial half-constituent, since the constituent can be extended by adding more modifiers to m.

Each type of span is created by recursively combining two smaller, adjacent spans; the constructions are specified graphically in Figure 2. An incomplete span is constructed from a pair of complete spans, indicating the division of the range $[h, m]$ into constituents headed by h and m. A complete span is created by "completing" an incomplete span with the other half of m's constituent. The point of concatenation in each construction—m in Figure 2(a) or r in Figure 2(b)—is the *split point*, a free index that must be enumerated to find the optimal construction.

In order to parse a sentence $\boldsymbol{x}$, it suffices to find optimal constructions for all complete and incomplete spans defined on $\boldsymbol{x}$. This can be

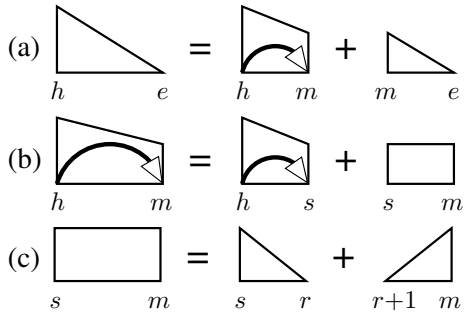


Figure 3: The dynamic-programming structures and derivations of the second-order sibling parser; sibling spans are depicted as boxes. For brevity, we elide the right-headed versions.

accomplished by adapting standard chart-parsing techniques (Cocke and Schwartz, 1970; Younger, 1967; Kasami, 1965) to the recursive derivations defined in Figure 2. Since each derivation is defined by two fixed indices (the boundaries of the span) and a third free index (the split point), the parsing algorithm requires $O(n^3)$ time and $O(n^2)$ space (Eisner, 1996; McAllester, 1999).

3.2 Second-order sibling factorization

As remarked by Eisner (1996) and McDonald and Pereira (2006), it is possible to rearrange the dynamic-programming structures to conform to an improved factorization that decomposes each tree into *sibling* parts—pairs of dependencies with a shared head. Specifically, a sibling part consists of a triple of indices (h, m, s) where (h, m) and (h, s) are dependencies, and where s and m are successive modifiers to the same side of h.

In order to parse this factorization, the second-order parser introduces a third type of dynamic-programming structure: *sibling* spans, which represent the region between successive modifiers of some head. Formally, we denote a sibling span as $S_{s,m}$ where s and m are a pair of modifiers involved in a sibling relationship. Modified versions of sibling spans will play an important role in the new parsing algorithms described in Section 4.

Figure 3 provides a graphical specification of the second-order parsing algorithm. Note that incomplete spans are constructed in a new way: the second-order parser combines a smaller incomplete span, representing the next-innermost dependency, with a sibling span that covers the region between the two modifiers. Sibling parts (h, m, s) can thus be obtained from Figure 3(b). Despite the use of second-order parts, each derivation is still defined by a span and split point, so the parser requires $O(n^3)$ time and $O(n^2)$ space.

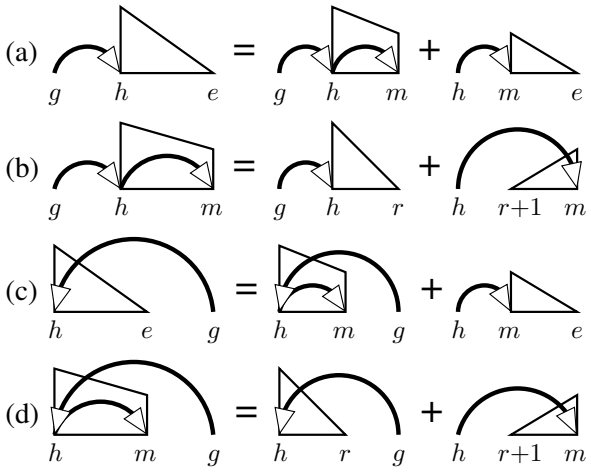


Figure 4: The dynamic-programming structures and derivations of Model 0. For brevity, we elide the right-headed versions. Note that (c) and (d) differ from (a) and (b) only in the position of g.

4 New third-order parsing algorithms

In this section we describe our new third-order dependency parsing algorithms. Our overall method is characterized by the augmentation of each span with a "grandparent" index: an index external to the span whose role will be made clear below. This section presents three parsing algorithms based on this idea: Model 0, a second-order parser, and Models 1 and 2, which are third-order parsers.

4.1 Model 0: all grandchildren

The first parser, Model 0, factors each dependency tree into a set of *grandchild* parts—pairs of dependencies connected head-to-tail. Specifically, a grandchild part is a triple of indices (g, h, m) where (g, h) and (h, m) are dependencies.[3]

In order to parse this factorization, we augment both complete and incomplete spans with grandparent indices; for brevity, we refer to these augmented structures as *g-spans*. Formally, we denote a complete g-span as $C^g_{h,e}$, where $C_{h,e}$ is a normal complete span and g is an index lying outside the range $[h, e]$, with the implication that (g, h) is a dependency. Incomplete g-spans are defined analogously and are denoted as $I^g_{h,m}$.

Figure 4 depicts complete and incomplete g-spans and provides a graphical specification of the

[3]The Carreras (2007) parser also uses grandchild parts but only in restricted cases; see Section 6 for details.

OPTIMIZEALLSPANS($\boldsymbol{x}$)

1. $\forall g, i \quad C^g_{i,i} = 0$ ◁ *base case*
2. **for** $w = 1 \ldots (n-1)$ ◁ *span width*
3. **for** $i = 1 \ldots (n-w)$ ◁ *span start index*
4. $j = i + w$ ◁ *span end index*
5. **for** $g < i$ **or** $g > j$ ◁ *grandparent index*
6. $I^g_{i,j} = \max_{i \le r < j} \{C^g_{i,r} + C^i_{j,r+1}\} +$ SCOREG($\boldsymbol{x}, g, i, j$)
7. $I^g_{j,i} = \max_{i \le r < j} \{C^g_{j,r+1} + C^j_{i,r}\} +$ SCOREG($\boldsymbol{x}, g, j, i$)
8. $C^g_{i,j} = \max_{i < m \le j} \{I^g_{i,m} + C^i_{m,j}\}$
9. $C^g_{j,i} = \max_{i \le m < j} \{I^g_{j,m} + C^j_{m,i}\}$
10. **endfor**
11. **endfor**
12. **endfor**

Figure 5: A bottom-up chart parser for Model 0. SCOREG is the scoring function for grandchild parts. We use the g-span identities as shorthand for their chart entries (e.g., $I^g_{i,j}$ refers to the entry containing the maximum score of that g-span).

Model 0 dynamic-programming algorithm. The algorithm resembles the first-order parser, except that every recursive construction must also set the grandparent indices of the smaller g-spans; fortunately, this can be done deterministically in all cases. For example, Figure 4(a) depicts the decomposition of $C^g_{h,e}$ into an incomplete half and a complete half. The grandparent of the incomplete half is copied from $C^g_{h,e}$ while the grandparent of the complete half is set to h, the head of m as defined by the construction. Clearly, grandchild parts (g, h, m) can be read off of the incomplete g-spans in Figure 4(b,d). Moreover, since each derivation copies the grandparent index g into successively smaller g-spans, grandchild parts will be produced for *all* grandchildren of g.

Model 0 can be parsed by adapting standard top-down or bottom-up chart parsing techniques. For concreteness, Figure 5 provides a pseudocode sketch of a bottom-up chart parser for Model 0; although the sketch omits many details, it suffices for the purposes of illustration. The algorithm progresses from small widths to large in the usual manner, but after defining the endpoints (i, j) there is an additional loop that enumerates all possible grandparents. Since each derivation is defined by three fixed indices (the g-span) and one free index (the split point), the complexity of the algorithm is $O(n^4)$ time and $O(n^3)$ space.

Note that the grandparent indices cause each g-span to have non-contiguous structure. For example, in Figure 4(a) the words between g and h will be controlled by some other g-span. Due to these discontinuities, the correctness of the Model 0 dynamic-programming algorithm may not be immediately obvious. While a full proof of correctness is beyond the scope of this paper, we note that each structure on the right-hand side of Figure 4 lies completely within the structure on the left-hand side. This nesting of structures implies, in turn, that the usual properties required to ensure the correctness of dynamic programming hold.

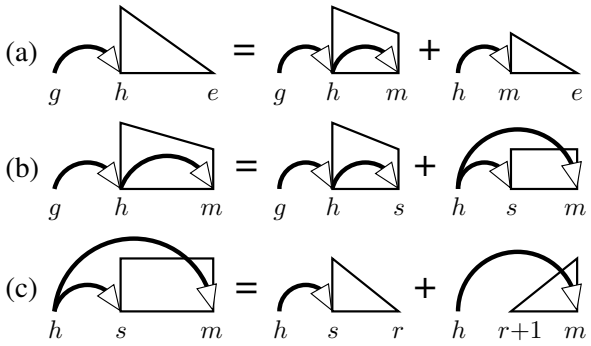


Figure 6: The dynamic-programming structures and derivations of Model 1. Right-headed and right-grandparented versions are omitted.

4.2 Model 1: all grand-siblings

We now describe our first third-order parsing algorithm. Model 1 decomposes each tree into a set of *grand-sibling* parts—combinations of sibling parts and grandchild parts. Specifically, a grand-sibling is a 4-tuple of indices (g, h, m, s) where (h, m, s) is a sibling part and (g, h, m) and (g, h, s) are grandchild parts. For example, in Figure 1, the words "must," "report," "sales," and "immediately" form a grand-sibling part.

In order to parse this factorization, we introduce sibling g-spans $S^h_{m,s}$, which are composed of a normal sibling span $S_{m,s}$ and an external index h, with the implication that (h, m, s) forms a valid sibling part. Figure 6 provides a graphical specification of the dynamic-programming algorithm for Model 1. The overall structure of the algorithm resembles the second-order sibling parser, with the addition of grandparent indices; as in Model 0, the grandparent indices can be set deterministically in all cases. Note that the sibling g-spans are crucial: they allow grand-sibling parts (g, h, m, s) to be read off of Figure 6(b), while simultaneously propagating grandparent indices to smaller g-spans.

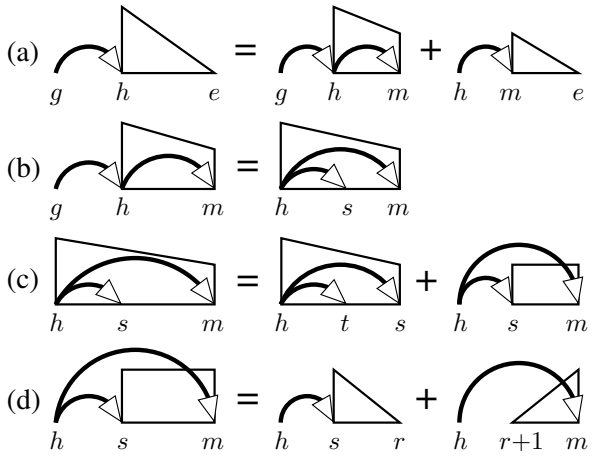


Figure 7: The dynamic-programming structures and derivations of Model 2. Right-headed and right-grandparented versions are omitted.

Like Model 0, Model 1 can be parsed via adaptations of standard chart-parsing techniques; we omit the details for brevity. Despite the move to third-order parts, each derivation is still defined by a g-span and a split point, so that parsing requires only $O(n^4)$ time and $O(n^3)$ space.

4.3 Model 2: grand-siblings and tri-siblings

Higher-order parsing algorithms have been proposed which extend the second-order sibling factorization to parts containing multiple siblings (McDonald and Pereira, 2006, also see Section 6 for discussion). In this section, we show how our g-span-based techniques can be combined with a third-order sibling parser, resulting in a parser that captures both grand-sibling parts and *tri-sibling* parts—4-tuples of indices (h, m, s, t) such that both (h, m, s) and (h, s, t) are sibling parts.

In order to parse this factorization, we introduce a new type of dynamic-programming structure: sibling-augmented spans, or *s-spans*. Formally, we denote an incomplete s-span as $I_{h,m,s}$ where $I_{h,m}$ is a normal incomplete span and s is an index lying in the strict interior of the range $[h, m]$, such that (h, m, s) forms a valid sibling part.

Figure 7 provides a graphical specification of the Model 2 parsing algorithm. An incomplete s-span is constructed by combining a smaller incomplete s-span, representing the next-innermost pair of modifiers, with a sibling g-span, covering the region between the outer two modifiers. As in Model 1, sibling g-spans are crucial for propagating grandparent indices, while allowing the recovery of tri-sibling parts (h, m, s, t). Figure 7(b) shows how an incomplete s-span can be converted into an incomplete g-span by exchanging the internal sibling index for an external grandparent index; in the process, grand-sibling parts (g, h, m, s) are enumerated. Since every derivation is defined by an augmented span and a split point, Model 2 can be parsed in $O(n^4)$ time and $O(n^3)$ space.

It should be noted that unlike Model 1, Model 2 produces grand-sibling parts only for the outermost pair of grandchildren,[4] similar to the behavior of the Carreras (2007) parser. In fact, the resemblance is more than passing, as Model 2 can *emulate* the Carreras (2007) algorithm by "demoting" each third-order part into a second-order part:

$$\textsc{ScoreGS}(\boldsymbol{x}, g, h, m, s) = \textsc{ScoreG}(\boldsymbol{x}, g, h, m)$$
$$\textsc{ScoreTS}(\boldsymbol{x}, h, m, s, t) = \textsc{ScoreS}(\boldsymbol{x}, h, m, s)$$

where ScoreG, ScoreS, ScoreGS and ScoreTS are the scoring functions for grandchildren, siblings, grand-siblings and tri-siblings, respectively. The emulated version has the same computational complexity as the original, so there is no practical reason to prefer it over the original. Nevertheless, the relationship illustrated above highlights the efficiency of our approach: we are able to recover third-order parts in place of second-order parts, at no additional cost.

4.4 Discussion

The technique of grandparent-index augmentation has proven fruitful, as it allows us to parse expressive third-order factorizations while retaining an efficient $O(n^4)$ runtime. In fact, our third-order parsing algorithms are "optimally" efficient in an asymptotic sense. Since each third-order part is composed of four separate indices, there are $\Theta(n^4)$ distinct parts. Any third-order parsing algorithm must at least consider the score of each part, hence third-order parsing is $\Omega(n^4)$ and it follows that the asymptotic complexity of Models 1 and 2 cannot be improved.

The key to the efficiency of our approach is a fundamental asymmetry in the structure of a directed tree: a head can have any number of modifiers, while a modifier always has exactly one head. Factorizations like that of Carreras (2007) obtain grandchild parts by augmenting spans with the indices of modifiers, leading to limitations on

[4] The reason for the restriction is that in Model 2, grand-siblings can only be derived via Figure 7(b), which does not recursively copy the grandparent index for reuse in smaller g-spans as Model 1 does in Figure 6(b).

the grandchildren that can participate in the factorization. Our method, by "inverting" the modifier indices into grandparent indices, exploits the structural asymmetry.

As a final note, the parsing algorithms described in this section fall into the category of *projective* dependency parsers, which forbid crossing dependencies. If crossing dependencies are allowed, it is possible to parse a first-order factorization by finding the maximum directed spanning tree (Chu and Liu, 1965; Edmonds, 1967; McDonald et al., 2005b). Unfortunately, designing efficient higher-order non-projective parsers is likely to be challenging, based on recent hardness results (McDonald and Pereira, 2006; McDonald and Satta, 2007).

5 Extensions

We briefly outline a few extensions to our algorithms; we hope to explore these in future work.

5.1 Probabilistic inference

Many statistical modeling techniques are based on partition functions and marginals—summations over the set of possible trees $\mathcal{Y}(\boldsymbol{x})$. Straightforward adaptations of the inside-outside algorithm (Baker, 1979) to our dynamic-programming structures would suffice to compute these quantities.

5.2 Labeled parsing

Our parsers are easily extended to labeled dependencies. Direct integration of labels into Models 1 and 2 would result in third-order parts composed of three labeled dependencies, at the cost of increasing the time and space complexities by factors of $O(L^3)$ and $O(L^2)$, respectively, where L bounds the number of labels per dependency.

5.3 Word senses

If each word in $\boldsymbol{x}$ has a set of possible "senses," our parsers can be modified to recover the best joint assignment of syntax and senses for $\boldsymbol{x}$, by adapting methods in Eisner (2000). Complexity would increase by factors of $O(S^4)$ time and $O(S^3)$ space, where S bounds the number of senses per word.

5.4 Increased context

If more vertical context is desired, the dynamic-programming structures can be extended with additional ancestor indices, resulting in a "spine" of ancestors above each span. Each additional ancestor lengthens the vertical scope of the factorization (e.g., from grand-siblings to "great-grand-siblings"), while increasing complexity by a factor of $O(n)$. Horizontal context can also be increased by adding internal sibling indices; each additional sibling widens the scope of the factorization (e.g., from grand-siblings to "grand-tri-siblings"), while increasing complexity by a factor of $O(n)$.

6 Related work

Our method augments each span with the index of the head that governs that span, in a manner superficially similar to parent annotation in CFGs (Johnson, 1998). However, parent annotation is a *grammar* transformation that is independent of any particular sentence, whereas our method annotates spans with *indices* into the current sentence. These indices allow the use of arbitrary features predicated on the position of the grandparent (e.g., word identity, POS tag, contextual POS tags) without affecting the asymptotic complexity of the parsing algorithm. Efficiently encoding this kind of information into a sentence-independent grammar transformation would be challenging at best.

Eisner (2000) defines dependency parsing models where each word has a set of possible "senses" and the parser recovers the best joint assignment of syntax and senses. Our new parsing algorithms could be implemented by defining the "sense" of each word as the index of its head. However, when parsing with senses, the complexity of the Eisner (2000) parser increases by factors of $O(S^3)$ time and $O(S^2)$ space (ibid., Section 4.2). Since each word has n potential heads, a direct application of the word-sense parser leads to time and space complexities of $O(n^6)$ and $O(n^4)$, respectively, in contrast to our $O(n^4)$ and $O(n^3)$.[5]

Eisner (2000) also uses head automata to score or recognize the dependents of each head. An interesting question is whether these automata could be coerced into modeling the grandparent indices used in our parsing algorithms. However, note that the head automata are defined in a sentence-independent manner, with two automata per word in the vocabulary (ibid., Section 2). The automata are thus analogous to the rules of a CFG and at-

[5] In brief, the reason for the inefficiency is that the word-sense parser is unable to exploit certain constraints, such as the fact that the endpoints of a sibling g-span must have the same head. The word-sense parser would needlessly enumerate all possible pairs of heads in this case.

tempts to use them to model grandparent indices would face difficulties similar to those already described for grammar transformations in CFGs.

It should be noted that third-order parsers have previously been proposed by McDonald and Pereira (2006), who remarked that their second-order sibling parser (see Figure 3) could easily be extended to capture $m > 1$ successive modifiers in $O(n^{m+1})$ time (ibid., Section 2.2). To our knowledge, however, Models 1 and 2 are the first third-order parsing algorithms capable of modeling grandchild parts. In our experiments, we find that grandchild interactions make important contributions to parsing performance (see Table 3).

Carreras (2007) presents a second-order parser that can score both sibling and grandchild parts, with complexities of $O(n^4)$ time and $O(n^3)$ space. An important limitation of the parser's factorization is that it only defines grandchild parts for outermost grandchildren: (g, h, m) is scored only when m is the outermost modifier of h in some direction. Note that Models 1 and 2 have the same complexity as Carreras (2007), but strictly greater expressiveness: for each sibling or grandchild part used in the Carreras (2007) factorization, Model 1 defines an enclosing grand-sibling, while Model 2 defines an enclosing tri-sibling or grand-sibling.

The factored parsing approach we focus on is sometimes referred to as "graph-based" parsing; a popular alternative is "transition-based" parsing, in which trees are constructed by making a series of incremental decisions (Yamada and Matsumoto, 2003; Attardi, 2006; Nivre et al., 2006; McDonald and Nivre, 2007). Transition-based parsers do not impose factorizations, so they can define arbitrary features on the tree as it is being built. As a result, however, they rely on greedy or approximate search algorithms to solve Eq. 1.

7 Parsing experiments

In order to evaluate the effectiveness of our parsers in practice, we apply them to the Penn WSJ Treebank (Marcus et al., 1993) and the Prague Dependency Treebank (Hajič et al., 2001; Hajič, 1998).[6] We use standard training, validation, and test splits[7] to facilitate comparisons. Accuracy is measured with unlabeled attachment score (UAS): the percentage of words with the correct head.[8]

7.1 Features for third-order parsing

Our parsing algorithms can be applied to scores originating from any source, but in our experiments we chose to use the framework of structured linear models, deriving our scores as:

$$\textsc{ScorePart}(\boldsymbol{x}, p) \;=\; \boldsymbol{w} \cdot \boldsymbol{f}(\boldsymbol{x}, p)$$

Here, $\boldsymbol{f}$ is a feature-vector mapping and $\boldsymbol{w}$ is a vector of associated parameters. Following standard practice for higher-order dependency parsing (McDonald and Pereira, 2006; Carreras, 2007), Models 1 and 2 evaluate not only the relevant third-order parts, but also the lower-order parts that are implicit in their third-order factorizations. For example, Model 1 defines feature mappings for dependencies, siblings, grandchildren, and grand-siblings, so that the score of a dependency parse is given by:

$$\begin{aligned}\textsc{Model1Score}(\boldsymbol{x}, y) \;=\; & \\ & \sum_{(h,m)\in y} \boldsymbol{w}_{\text{dep}} \cdot \boldsymbol{f}_{\text{dep}}(\boldsymbol{x}, h, m) \\ & \sum_{(h,m,s)\in y} \boldsymbol{w}_{\text{sib}} \cdot \boldsymbol{f}_{\text{sib}}(\boldsymbol{x}, h, m, s) \\ & \sum_{(g,h,m)\in y} \boldsymbol{w}_{\text{gch}} \cdot \boldsymbol{f}_{\text{gch}}(\boldsymbol{x}, g, h, m) \\ & \sum_{(g,h,m,s)\in y} \boldsymbol{w}_{\text{gsib}} \cdot \boldsymbol{f}_{\text{gsib}}(\boldsymbol{x}, g, h, m, s)\end{aligned}$$

Above, y is simultaneously decomposed into several different types of parts; trivial modifications to the Model 1 parser allow it to evaluate all of the necessary parts in an interleaved fashion. A similar treatment of Model 2 yields five feature mappings: the four above plus $\boldsymbol{f}_{\text{tsib}}(\boldsymbol{x}, h, m, s, t)$, which represents tri-sibling parts.

The lower-order feature mappings $\boldsymbol{f}_{\text{dep}}$, $\boldsymbol{f}_{\text{sib}}$, and $\boldsymbol{f}_{\text{gch}}$ are based on feature sets from previous work (McDonald et al., 2005a; McDonald and Pereira, 2006; Carreras, 2007), to which we added lexicalized versions of several features. For example, $\boldsymbol{f}_{\text{dep}}$ contains lexicalized "in-between" features that depend on the head and modifier words as well as a word lying in between the two; in contrast, previous work has generally defined in-between features for POS tags only. As another example, our

[6] For English, we extracted dependencies using Joakim Nivre's Penn2Malt tool with standard head rules (Yamada and Matsumoto, 2003); for Czech, we "projectivized" the training data by finding best-match projective trees.

[7] For Czech, the PDT has a predefined split; for English, we split the Sections as: 2–21 training, 22 validation, 23 test.

[8] As in previous work, English evaluation ignores any token whose gold-standard POS tag is one of {`` '' : , .}.

second-order mappings $\boldsymbol{f}_{\text{sib}}$ and $\boldsymbol{f}_{\text{gch}}$ define lexical trigram features, while previous work has generally used POS trigrams only.

Our third-order feature mappings $\boldsymbol{f}_{\text{gsib}}$ and $\boldsymbol{f}_{\text{tsib}}$ consist of four types of features. First, we define *4-gram features* that characterize the four relevant indices using words and POS tags; examples include POS 4-grams and mixed 4-grams with one word and three POS tags. Second, we define *4-gram context features* consisting of POS 4-grams augmented with adjacent POS tags: for example, $\boldsymbol{f}_{\text{gsib}}(\boldsymbol{x}, g, h, m, s)$ includes POS 7-grams for the tags at positions $(g, h, m, s, g{+}1, h{+}1, m{+}1)$. Third, we define *backed-off features* that track bigram and trigram interactions which are absent in the lower-order feature mappings: for example, $\boldsymbol{f}_{\text{tsib}}(\boldsymbol{x}, h, m, s, t)$ contains features predicated on the trigram (m, s, t) and the bigram (m, t), neither of which exist in any lower-order part. Fourth, noting that coordinations are typically annotated as grand-siblings (e.g., "report purchases and sales" in Figure 1), we define *coordination features* for certain grand-sibling parts. For example, $\boldsymbol{f}_{\text{gsib}}(\boldsymbol{x}, g, h, m, s)$ contains features examining the implicit head-modifier relationship (g, m) that are only activated when the POS tag of s is a coordinating conjunction.

Finally, we make two brief remarks regarding the use of POS tags. First, we assume that input sentences have been automatically tagged in a preprocessing step.[9] Second, for any feature that depends on POS tags, we include *two* copies of the feature: one using normal POS tags and another using coarsened versions[10] of the POS tags.

7.2 Averaged perceptron training

There are a wide variety of parameter estimation methods for structured linear models, such as log-linear models (Lafferty et al., 2001) and max-margin models (Taskar et al., 2003). We chose the averaged structured perceptron (Freund and Schapire, 1999; Collins, 2002) as it combines highly competitive performance with fast training times, typically converging in 5–10 iterations. We train each parser for 10 iterations and select parameters from the iteration that achieves the best score on the validation set.

Beam	Pass	Orac	Acc1	Acc2	Time1	Time2
0.0001	26.5	99.92	93.49	93.49	49.6m	73.5m
0.001	16.7	99.72	93.37	93.29	25.9m	24.2m
0.01	9.1	99.19	93.26	93.16	6.7m	7.9m

Table 1: Effect of the marginal-probability beam on English parsing. For each beam value, parsers were trained on the English training set and evaluated on the English validation set; the same beam value was applied to both training and validation data. **Pass** = %dependencies surviving the beam in training data, **Orac** = maximum achievable UAS on validation data, **Acc1/Acc2** = UAS of Models 1/2 on validation data, and **Time1/Time2** = minutes per perceptron training iteration for Models 1/2, averaged over all 10 iterations. For perspective, the English training set has a total of 39,832 sentences and 950,028 words. A beam of 0.0001 was used in all experiments outside this table.

7.3 Coarse-to-fine pruning

In order to decrease training times, we follow Carreras et al. (2008) and eliminate unlikely dependencies using a form of coarse-to-fine pruning (Charniak and Johnson, 2005; Petrov and Klein, 2007). In brief, we train a log-linear first-order parser[11] and for every sentence $\boldsymbol{x}$ in training, validation, and test data we compute the marginal probability $P(h, m \mid \boldsymbol{x})$ of each dependency. Our parsers are then modified to ignore any dependency (h, m) whose marginal probability is below $0.0001 \times \max_{h'} P(h', m \mid \boldsymbol{x})$. Table 1 provides information on the behavior of the pruning method.

7.4 Main results

Table 2 lists the accuracy of Models 1 and 2 on the English and Czech test sets, together with some relevant results from related work.[12] The models marked "†" are *not* directly comparable to our work as they depend on additional sources of information that our models are trained without—unlabeled data in the case of Koo et al. (2008) and

[9] For Czech, the PDT provides automatic tags; for English, we used MXPOST (Ratnaparkhi, 1996) to tag validation and test data, with 10-fold cross-validation on the training set. Note that the reliance on POS-tagged input can be relaxed slightly by treating POS tags as word senses; see Section 5.3 and McDonald (2006, Table 6.1).

[10] For Czech, we used the first character of the tag; for English, we used the first two characters, except `PRP` and `PRP$`.

[11] For English, we generate marginals using a projective parser (Baker, 1979; Eisner, 2000); for Czech, we generate marginals using a non-projective parser (Smith and Smith, 2007; McDonald and Satta, 2007; Koo et al., 2007). Parameters for these models are obtained by running exponentiated gradient training for 10 iterations (Collins et al., 2008).

[12] Model 0 was not tested as its factorization is a strict subset of the factorization of Model 1.

Parser	Eng	Cze
McDonald et al. (2005a,2005b)	90.9	84.4
McDonald and Pereira (2006)	91.5	85.2
Koo et al. (2008), standard	92.02	86.13
Model 1	93.04	87.38
Model 2	92.93	87.37
Koo et al. (2008), semi-sup†	93.16	87.13
Suzuki et al. (2009)†	93.79	88.05
Carreras et al. (2008)†	93.5	—

Table 2: UAS of Models 1 and 2 on test data, with relevant results from related work. Note that Koo et al. (2008) is listed with standard features and semi-supervised features. †: see main text.

Suzuki et al. (2009) and phrase-structure annotations in the case of Carreras et al. (2008). All three of the "†" models are based on versions of the Carreras (2007) parser, so modifying these methods to work with our new third-order parsing algorithms would be an interesting topic for future research. For example, Models 1 and 2 obtain results comparable to the semi-supervised parsers of Koo et al. (2008), and additive gains might be realized by applying their cluster-based feature sets to our enriched factorizations.

7.5 Ablation studies

In order to better understand the contributions of the various feature types, we ran additional ablation experiments; the results are listed in Table 3, in addition to the scores of Model 0 and the emulated Carreras (2007) parser (see Section 4.3). Interestingly, grandchild interactions appear to provide important information: for example, when Model 2 is used without grandchild-based features ("Model 2, no-G" in Table 3), its accuracy suffers noticeably. In addition, it seems that grandchild interactions are particularly useful in Czech, while sibling interactions are less important: consider that Model 0, a second-order grandchild parser with no sibling-based features, can easily outperform "Model 2, no-G," a *third*-order sibling parser with no grandchild-based features.

8 Conclusion

We have presented new parsing algorithms that are capable of efficiently parsing third-order factorizations, including both grandchild and sibling interactions. Due to space restrictions, we have been necessarily brief at some points in this paper; some additional details can be found in Koo (2010).

Parser	Eng	Cze
Model 0	93.07	87.39
Carreras (2007) emulation	93.14	87.25
Model 1	93.49	87.64
Model 1, no-3rd	93.17	87.57
Model 2	93.49	87.46
Model 2, no-3rd	93.20	87.43
Model 2, no-G	92.92	86.76

Table 3: UAS for modified versions of our parsers on validation data. The term **no-3rd** indicates a parser that was trained and tested with the third-order feature mappings $\boldsymbol{f}_{\text{gsib}}$ and $\boldsymbol{f}_{\text{tsib}}$ deactivated, though lower-order features were retained; note that "Model 2, no-3rd" is not identical to the Carreras (2007) parser as it defines grandchild parts for the pair of grandchildren. The term **no-G** indicates a parser that was trained and tested with the grandchild-based feature mappings $\boldsymbol{f}_{\text{gch}}$ and $\boldsymbol{f}_{\text{gsib}}$ deactivated; note that "Model 2, no-G" emulates the third-order sibling parser proposed by McDonald and Pereira (2006).

There are several possibilities for further research involving our third-order parsing algorithms. One idea would be to consider extensions and modifications of our parsers, some of which have been suggested in Sections 5 and 7.4. A second area for future work lies in applications of dependency parsing. While we have evaluated our new algorithms on standard parsing benchmarks, there are a wide variety of tasks that may benefit from the extended context offered by our third-order factorizations; for example, the 4-gram substructures enabled by our approach may be useful for dependency-based language modeling in machine translation (Shen et al., 2008). Finally, in the hopes that others in the NLP community may find our parsers useful, we provide a free distribution of our implementation.[2]

Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments and suggestions. We also thank Regina Barzilay and Alexander Rush for their much-appreciated input during the writing process. The authors gratefully acknowledge the following sources of support: Terry Koo and Michael Collins were both funded by a DARPA subcontract under SRI (#27-001343), and Michael Collins was additionally supported by NTT (Agmt. dtd. 06/21/98).

References

Giuseppe Attardi. 2006. Experiments with a Multilanguage Non-Projective Dependency Parser. In *Proceedings of the 10th CoNLL*, pages 166–170. Association for Computational Linguistics.

James Baker. 1979. Trainable Grammars for Speech Recognition. In *Proceedings of the 97th meeting of the Acoustical Society of America*.

Xavier Carreras, Michael Collins, and Terry Koo. 2008. TAG, Dynamic Programming, and the Perceptron for Efficient, Feature-rich Parsing. In *Proceedings of the 12th CoNLL*, pages 9–16. Association for Computational Linguistics.

Xavier Carreras. 2007. Experiments with a Higher-Order Projective Dependency Parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL*, pages 957–961. Association for Computational Linguistics.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine N-best Parsing and MaxEnt Discriminative Reranking. In *Proceedings of the 43rd ACL*.

Y.J. Chu and T.H. Liu. 1965. On the Shortest Arborescence of a Directed Graph. *Science Sinica*, 14:1396–1400.

John Cocke and Jacob T. Schwartz. 1970. Programming Languages and Their Compilers: Preliminary Notes. Technical report, New York University.

Michael Collins, Amir Globerson, Terry Koo, Xavier Carreras, and Peter L. Bartlett. 2008. Exponentiated Gradient Algorithms for Conditional Random Fields and Max-Margin Markov Networks. *Journal of Machine Learning Research*, 9:1775–1822, Aug.

Michael Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the 7th EMNLP*, pages 1–8. Association for Computational Linguistics.

Jack R. Edmonds. 1967. Optimum Branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240.

Jason Eisner. 1996. Three New Probabilistic Models for Dependency Parsing: An Exploration. In *Proceedings of the 16th COLING*, pages 340–345. Association for Computational Linguistics.

Jason Eisner. 2000. Bilexical Grammars and Their Cubic-Time Parsing Algorithms. In Harry Bunt and Anton Nijholt, editors, *Advances in Probabilistic and Other Parsing Technologies*, pages 29–62. Kluwer Academic Publishers.

Yoav Freund and Robert E. Schapire. 1999. Large Margin Classification Using the Perceptron Algorithm. *Machine Learning*, 37(3):277–296.

Jan Hajič, Eva Hajičová, Petr Pajas, Jarmila Panevova, and Petr Sgall. 2001. *The Prague Dependency Treebank 1.0, LDC No. LDC2001T10*. Linguistics Data Consortium.

Jan Hajič. 1998. Building a Syntactically Annotated Corpus: The Prague Dependency Treebank. In Eva Hajičová, editor, *Issues of Valency and Meaning. Studies in Honor of Jarmila Panevová*, pages 12–19.

Mark Johnson. 1998. PCFG Models of Linguistic Tree Representations. *Computational Linguistics*, 24(4):613–632.

Tadao Kasami. 1965. An Efficient Recognition and Syntax-analysis Algorithm for Context-free Languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Lab.

Terry Koo, Amir Globerson, Xavier Carreras, and Michael Collins. 2007. Structured Prediction Models via the Matrix-Tree Theorem. In *Proceedings of EMNLP-CoNLL*, pages 141–150. Association for Computational Linguistics.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple Semi-supervised Dependency Parsing. In *Proceedings of the 46th ACL*, pages 595–603. Association for Computational Linguistics.

Terry Koo. 2010. *Advances in Discriminative Dependency Parsing*. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, June.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the 18th ICML*, pages 282–289. Morgan Kaufmann.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

David A. McAllester. 1999. On the Complexity Analysis of Static Analyses. In *Proceedings of the 6th Static Analysis Symposium*, pages 312–329. Springer-Verlag.

Ryan McDonald and Joakim Nivre. 2007. Characterizing the Errors of Data-Driven Dependency Parsers. In *Proceedings of EMNLP-CoNLL*, pages 122–131. Association for Computational Linguistics.

Ryan McDonald and Fernando Pereira. 2006. Online Learning of Approximate Dependency Parsing Algorithms. In *Proceedings of the 11th EACL*, pages 81–88. Association for Computational Linguistics.

Ryan McDonald and Giorgio Satta. 2007. On the Complexity of Non-Projective Data-Driven Dependency Parsing. In *Proceedings of IWPT*.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005a. Online Large-Margin Training of Dependency Parsers. In *Proceedings of the 43rd ACL*, pages 91–98. Association for Computational Linguistics.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005b. Non-Projective Dependency Parsing using Spanning Tree Algorithms. In *Proceedings of HLT-EMNLP*, pages 523–530. Association for Computational Linguistics.

Ryan McDonald. 2006. *Discriminative Training and Spanning Tree Algorithms for Dependency Parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA, USA, July.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülşen Eryiğit, and Svetoslav Marinov. 2006. Labeled Pseudo-Projective Dependency Parsing with Support Vector Machines. In *Proceedings of the 10th CoNLL*, pages 221–225. Association for Computational Linguistics.

Slav Petrov and Dan Klein. 2007. Improved Inference for Unlexicalized Parsing. In *Proceedings of HLT-NAACL*, pages 404–411. Association for Computational Linguistics.

Adwait Ratnaparkhi. 1996. A Maximum Entropy Model for Part-Of-Speech Tagging. In *Proceedings of the 1st EMNLP*, pages 133–142. Association for Computational Linguistics.

Libin Shen, Jinxi Xu, and Ralph Weischedel. 2008. A New String-to-Dependency Machine Translation Algorithm with a Target Dependency Language Model. In *Proceedings of the 46th ACL*, pages 577–585. Association for Computational Linguistics.

David A. Smith and Noah A. Smith. 2007. Probabilistic Models of Nonprojective Dependency Trees. In *Proceedings of EMNLP-CoNLL*, pages 132–140. Association for Computational Linguistics.

Jun Suzuki, Hideki Isozaki, Xavier Carreras, and Michael Collins. 2009. An Empirical Study of Semi-supervised Structured Conditional Models for Dependency Parsing. In *Proceedings of EMNLP*, pages 551–560. Association for Computational Linguistics.

Ben Taskar, Carlos Guestrin, and Daphne Koller. 2003. Max margin markov networks. In Sebastian Thrun, Lawrence K. Saul, and Bernhard Schölkopf, editors, *NIPS*. MIT Press.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical Dependency Analysis with Support Vector Machines. In *Proceedings of the 8th IWPT*, pages 195–206. Association for Computational Linguistics.

David H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208.