Parsing as Reduction

Daniel Fernández-González[†*] **André F. T. Martins**[‡#]
[†]Departamento de Informática, Universidade de Vigo, Campus As Lagoas, 32004 Ourense, Spain
[‡]Instituto de Telecomunicações, Instituto Superior Técnico, 1049-001 Lisboa, Portugal
[#]Priberam Labs, Alameda D. Afonso Henriques, 41, 2º, 1000-123 Lisboa, Portugal
danifg@uvigo.es, atm@priberam.pt

Abstract

We reduce phrase-based parsing to dependency parsing. Our reduction is grounded on a new intermediate representation, "head-ordered dependency trees," shown to be isomorphic to constituent trees. By encoding order information in the dependency labels, we show that any off-the-shelf, trainable dependency parser can be used to produce constituents. When this parser is non-projective, we can perform discontinuous parsing in a very natural manner. Despite the simplicity of our approach, experiments show that the resulting parsers are on par with strong baselines, such as the Berkeley parser for English and the best non-reranking system in the SPMRL-2014 shared task. Results are particularly striking for discontinuous parsing of German, where we surpass the current state of the art by a wide margin.

1 Introduction

Constituent parsing is a central problem in NLP—one at which statistical models trained on treebanks have excelled (Charniak, 1996; Klein and Manning, 2003; Petrov and Klein, 2007). However, most existing parsers are slow, since they need to deal with a heavy grammar constant. Dependency parsers are generally faster, but less informative, since they do not produce constituents, which are often required by downstream applications (Johansson and Nugues, 2008; Wu et al., 2009; Berg-Kirkpatrick et al., 2011; Elming et al., 2013). How to get the best of both worlds?

Coarse-to-fine decoding (Charniak and Johnson, 2005) and shift-reduce parsing (Sagae and Lavie, 2005; Zhu et al., 2013) were a step forward to accelerate constituent parsing, but typical runtimes still lag those of dependency parsers. This is only made worse if **discontinuous** constituents are allowed—such discontinuities are convenient to represent wh-movement, scrambling, extraposition, and other linguistic phenomena common in free word order languages. While non-projective dependency parsers, which are able to model such phenomena, have been widely developed in the last decade (Nivre et al., 2007; McDonald et al., 2006; Martins et al., 2013), discontinuous constituent parsing is still taking its first steps (Maier and Søgaard, 2008; Kallmeyer and Maier, 2013).

In this paper, we show that an off-the-shelf, trainable, **dependency parser** is enough to build a highly-competitive constituent parser. This (surprising) result is based on a **reduction**[1] of constituent to dependency parsing, followed by a simple post-processing procedure to recover unaries. Unlike other constituent parsers, ours does not require estimating a grammar, nor binarizing the treebank. Moreover, when the dependency parser is non-projective, our method can perform discontinuous constituent parsing in a very natural way.

Key to our approach is the notion of **head-ordered dependency trees** (shown in Figure 1): by endowing dependency trees with this additional layer of structure, we show that they become isomorphic to constituent trees. We encode this structure as part of the dependency labels, enabling a dependency-to-constituent conversion. A related conversion was attempted by Hall and Nivre (2008) to parse German, but their complex encoding scheme blows up the number of arc labels, affecting the final parser's quality. By contrast, our light encoding achieves a 10-fold decrease in the label alphabet, leading to more accurate parsing.

While simple, our reduction-based parsers are on par with the Berkeley parser for English (Petrov

[*]This research was carried out during an internship at Priberam Labs.

[1]The title of this paper is inspired by the seminal paper of Pereira and Warren (1983) "Parsing as Deduction."

and Klein, 2007), and with the best single system in the recent SPMRL shared task (Seddah et al., 2014), for eight morphologically rich languages. For discontinuous parsing, we surpass the current state of the art by a wide margin on two German datasets (TIGER and NEGRA), while achieving fast parsing speeds. We provide a free distribution of our parsers along with this paper, as part of the TurboParser toolkit.[2]

2 Background

We start by reviewing constituent and dependency representations, and setting up the notation. Following Kong and Smith (2014), we use c-/d- prefixes for convenience (*e.g.*, we write c-parser for constituent parser and d-tree for dependency tree).

2.1 Constituent Trees

Constituent-based representations are commonly seen as derivations according to a context-free grammar (CFG). Here, we focus on properties of the c-trees, rather than of the grammars used to generate them. We consider a broad scenario that permits c-trees with discontinuities, such as the ones derived with linear context-free rewriting systems (LCFRS; Vijay-Shanker et al. (1987)). We also assume that the c-trees are lexicalized.

Formally, let $w_1 w_2 \dots w_L$ be a sentence, where w_i denotes the word in the ith position. A **c-tree** is a rooted tree whose leaves are the words $\{w_i\}_{i=1}^{L}$, and whose internal nodes (constituents) are represented as a tuple $\langle Z, h, \mathcal{I} \rangle$, where Z is a non-terminal symbol, $h \in \{1, \dots, L\}$ indicates the lexical head, and $\mathcal{I} \subseteq \{1, \dots, L\}$ is the node's yield. Each word's parent is a pre-terminal unary node of the form $\langle p_i, i, \{i\} \rangle$, where p_i denotes the word's part-of-speech (POS) tag. The yields and lexical heads are defined so that for every constituent $\langle Z, h, \mathcal{I} \rangle$ with children $\{\langle X_k, m_k, \mathcal{J}_k \rangle\}_{k=1}^{K}$, (i) we have $\mathcal{I} = \bigcup_{k=1}^{K} \mathcal{J}_k$; and (ii) there is a unique k such that $h = m_k$. This kth node (called the head-child node) is commonly chosen applying a handwritten set of head rules (Collins, 1999; Yamada and Matsumoto, 2003).

A c-tree is **continuous** if all nodes $\langle Z, h, \mathcal{I} \rangle$ have a contiguous yield $\mathcal{I}$, and **discontinuous** otherwise. Trees derived by a CFG are always continuous; those derived by a LCFRS may have discontinuities, the yield of a node being a union of spans, possibly with gaps in the middle. Figure 1 shows an example of a continuous and a discontinuous c-tree. Discontinuous c-trees have crossing branches, if the leaves are drawn in left-to-right surface order. An internal node which is not a pre-terminal is called a **proper node**. A node is called unary if it has exactly one child. A c-tree without unary proper nodes is called **unaryless**. If all proper nodes have exactly two children then it is called a **binary** c-tree. Continuous binary trees may be regarded as having been generated by a CFG in Chomsky normal form.

Prior work. There has been a long string of work in statistical c-parsing, shifting from simple models (Charniak, 1996) to more sophisticated ones using structural annotation (Johnson, 1998; Klein and Manning, 2003), latent grammars (Matsuzaki et al., 2005; Petrov and Klein, 2007), and lexicalization (Eisner, 1996; Collins, 1999). An orthogonal line of work uses ensemble or reranking strategies to further improve accuracy (Charniak and Johnson, 2005; Huang, 2008; Björkelund et al., 2014). Discontinuous c-parsing is considered a much harder problem, involving mildly context-sensitive formalisms such as LCFRS or range concatenation grammars, with treebank-derived c-parsers exhibiting near-exponential runtime (Kallmeyer and Maier, 2013, Figure 27). To speed up decoding, prior work has considered restrictons, such as bounding the fan-out (Maier et al., 2012) and requiring well-nestedness (Kuhlmann and Nivre, 2006; Gómez-Rodríguez et al., 2010). Other approaches eliminate the discontinuities via tree transformations (Boyd, 2007; Kübler et al., 2008), sometimes as a pruning step in a coarse-to-fine parsing approach (van Cranenburgh and Bod, 2013). However, reported runtimes are still superior to 10 seconds per sentence, which is not practical. Recently, Versley (2014a) proposed an easy-first approach that leads to considerable speed-ups, but is less accurate. In this paper, we design fast discontinuous c-parsers that outperform all the ones above by a wide margin, with similar runtimes as Versley (2014a).

2.2 Dependency Trees

In this paper, we use d-parsers as a black box to parse constituents. Given a sentence $w_1 \dots w_L$, a **d-tree** is a directed tree spanning all the words in the sentence.[3] Each arc in this tree is a tuple

[2] `http://www.ark.cs.cmu.edu/TurboParser`

[3] We assume throughout that dependency trees have a single root among $\{w_1, \dots, w_L\}$. Therefore, there is no need to

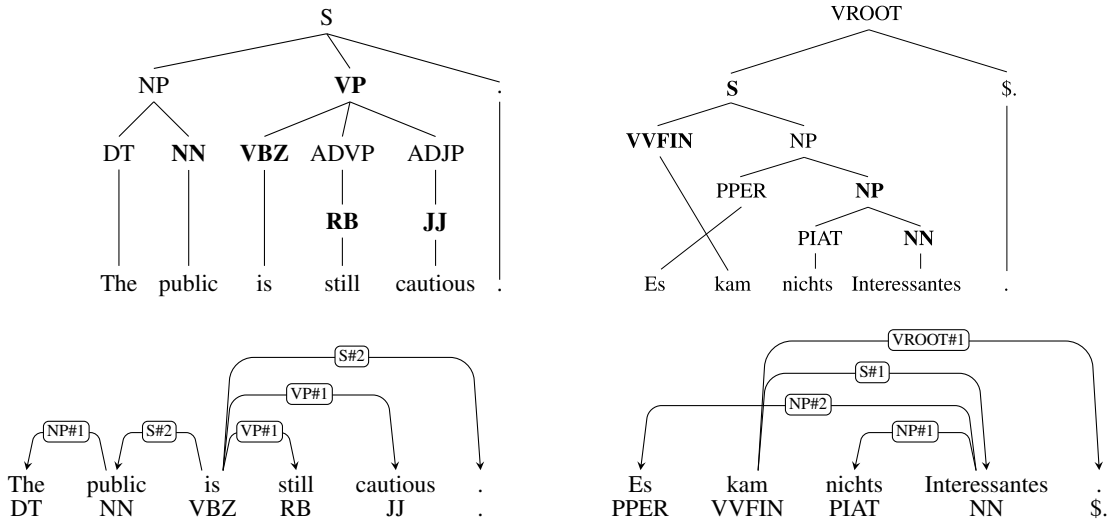

Figure 1: Top: a continuous (left) and a discontinuous (right) c-tree, taken from English PTB §22 and German NEGRA, respectively. Head-child nodes are in bold. Bottom: corresponding head-ordered d-trees. The indices #1, #2, etc. denote the order of attachment events for each head. Note that the English unary nodes `ADVP` and `ADJP` are dropped in the conversion.

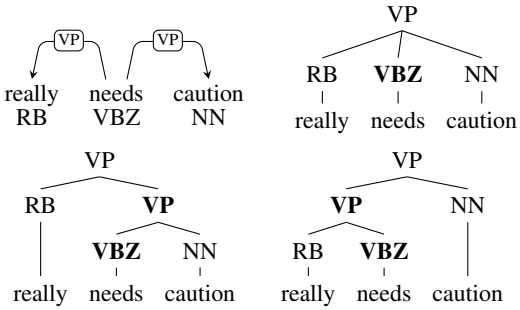

Figure 2: Three different c-structures for the VP "*really needs caution*." All are consistent with the d-structure at the top left.

$\langle h, m, \ell \rangle$, expressing a typed dependency relation ℓ between the head word w_h and the modifier w_m.

A d-tree is **projective** if for every arc $\langle h, m, \ell \rangle$ there is a directed path from h to all words that lie between h and m in the surface string (Kahane et al., 1998). Projective d-trees can be obtained from continuous c-trees by reading off the lexical heads and dropping the internal nodes (Gaifman, 1965). However, this relation is many-to-one: as shown in Figure 2, several c-trees may project onto the same d-tree, differing on their flatness and on left or right-branching decisions. In the next section, we introduce the concept of head-ordered d-trees and express one-to-one mappings between these two representations.

Prior work. There has been a considerable amount of work developing rich-feature d-parsers. While projective d-parsers can use dynamic programming (Eisner and Satta, 1999; Koo and Collins, 2010), non-projective d-parsers typically rely on approximate decoders, since the underlying problem is NP-hard beyond arc-factored models (McDonald and Satta, 2007). An alternative are transition-based d-parsers (Nivre et al., 2006; Zhang and Nivre, 2011), which achieve observed linear time. Since d-parsing algorithms do not have a grammar constant, typical implementations are significantly faster than c-parsers (Rush and Petrov, 2012; Martins et al., 2013). The key contribution of this paper is to reduce c-parsing to d-parsing, allowing to bring these runtimes closer.

consider an extra root symbol, as often done in the literature.

3 Head-Ordered Dependency Trees

We next endow d-trees with another layer of structure, namely **order information**. In this framework, not all modifiers of a head are "born equal." Instead, their attachment to the head occurs as a sequence of "events," which reflect the head's preference for attaching some modifiers before others. As we will see, this additional structure will undo the ambiguity expressed in Figure 2.

3.1 Strictly Ordered Dependency Trees

Let us start with the simpler case where the attachment order is strict. For each head word h with modifiers $M_h = \{m_1, \ldots, m_K\}$, we endow M_h with a **strict order relation** $\prec_h$, so we can organize all the modifiers of h as a chain, $m_{i_1} \prec_h m_{i_2} \prec_h \ldots \prec_h m_{i_K}$. We regard this chain as reflecting the order by which words are attached (*i.e.*, if $m_i \prec_h m_j$ this means that "m_i is attached

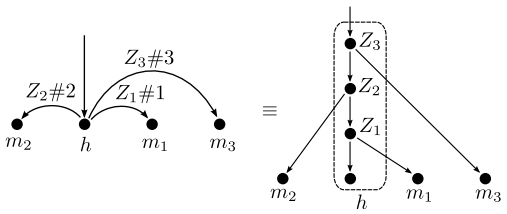

Figure 3: Transformation of a strictly-ordered d-tree into a binary c-tree. Each node is split into a linked list forming a spine, to which modifiers are attached in order.

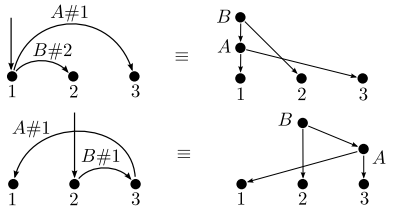

Figure 4: Two discontinuous constructions caused by a non-nested order (top) and a non-projective d-tree (bottom). In both cases node A has a non-contiguous yield.

to h before m_j"). We represent this graphically by decorating d-arcs with indices ($\#1, \#2, \ldots$) to denote the order of events, as we do in Figure 1.

A d-tree endowed with a strict order for each head is called a **strictly ordered d-tree**. We establish below a correspondence between strictly ordered d-trees and binary c-trees. Before doing so, we need a few more definitions about c-trees. For each word position $h \in \{1, \ldots, L\}$, we define $\psi(h)$ as the node higher in the c-tree whose lexical head is h. We call the path from $\psi(h)$ down to the pre-terminal p_h the **spine** of h. We may regard a c-tree as a set of L spines, one per word, which attach to each other to form a tree (Carreras et al., 2008). We then have the following

Proposition 1. *Binary c-trees and strictly-ordered d-trees are isomorphic, i.e., there is a one-to-one correspondence between the two sets, where the number of symbols is preserved.*

Proof. We use the construction in Figure 3. A formal proof is given as supplementary material. □

3.2 Weakly Ordered Dependency Trees

Next, we relax the strict order assumption, restricting the modifier sets $M_h = \{m_1, \ldots, m_K\}$ to be only **weakly ordered**. This means that we can partition the K modifiers into J equivalence classes, $M_h = \bigcup_{j=1}^{J} \bar{M}_h^j$, and define a strict order $\prec_h$ on the quotient set: $\bar{M}_h^1 \prec_h \ldots \prec_h \bar{M}_h^J$. Intuitively, there is still a sequence of events (1 to J), but now at each event j it may happen that multiple modifiers (the ones in the equivalence set $\bar{M}_h^j$) are simultaneously attached to h. A **weakly ordered d-tree** is a d-tree endowed with a weak order for each head and such that any pair m, m' in the same equivalence class (written $m \equiv_h m'$) receive the same dependency label ℓ.

Algorithm 1 Conversion from c-tree to d-tree

Input: c-tree $\mathcal{C}$.
Output: head-ordered d-tree $\mathcal{D}$.
1: Nodes := GETPOSTORDERTRAVERSAL($\mathcal{C}$).
2: Set $j(h) := 1$ for every $h = 1, \ldots, L$.
3: **for** $v := \langle Z, h, \mathcal{I} \rangle \in$ Nodes **do**
4: **for** every $u := \langle X, m, \mathcal{J} \rangle$ which is a child of v **do**
5: **if** $m \neq h$ **then**
6: Add to $\mathcal{D}$ an arc $\langle h, m, Z \rangle$, and put it in $\bar{M}_h^{j(h)}$.
7: **end if**
8: **end for**
9: Set $j(h) := j(h) + 1$.
10: **end for**

We now show that Proposition 1 can be generalized to weakly ordered d-trees.

Proposition 2. *Unaryless c-trees and weakly-ordered d-trees are isomorphic.*

Proof. This is a simple extension of Proposition 1. The construction is the same as in Figure 3, but now we can collapse some of the nodes in the linked list, originating multiple modifiers attaching to the same position of the spine—this is only possible for sibling arcs with the same index and arc label. Note, however, that if we start with a c-tree with unary nodes and apply the inverse procedure to obtain a d-tree, the unary nodes will be lost, since they do not involve attachment of modifiers. In a chain of unary nodes, only the last node is recovered in the inverse transformation. □

We emphasize that Propositions 1–2 hold without blowing up the number of symbols. That is, the dependency label alphabet is exactly the same as the set of phrasal symbols in the constituent representations. Algorithms 1–2 convert back and forth between the two formalisms, performing the construction of Figure 3. Both algorithms run in linear time with respect to the size of the sentence.

3.3 Continuous and Projective Trees

What about the more restricted class of projective d-trees? Can we find an equivalence relation with continuous c-trees? In this section, we give a precise answer to this question. It turns out that we need an additional property, illustrated in Figure 4.

We say that $\prec_h$ has the **nesting property** iff closer words in the same direction are always attached first, *i.e.*, iff $h < m_i < m_j$ or $h > m_i >$

Algorithm 2 Conversion from d-tree to c-tree

Input: head-ordered d-tree $\mathcal{D}$.
Output: c-tree $\mathcal{C}$.

Nodes := GETPOSTORDERTRAVERSAL($\mathcal{D}$).
for $h \in$ Nodes **do**
  Create $v := \langle p_h, h, \{h\}\rangle$ and set $\psi(h) := v$.
  Sort $M_h(\mathcal{D})$, yielding $\bar{M}_h^1 \prec_h \bar{M}_h^2 \prec_h \ldots \prec_h \bar{M}_h^J$.
  for $j = 1, \ldots, J$ **do**
    Let Z be the label in $\{\langle h, m, Z\rangle \mid m \in \bar{M}_h^j\}$.
    Obtain c-nodes $\psi(h) = \langle X, h, \mathcal{I}\rangle$ and $\psi(m) = \langle Y_m, m, \mathcal{J}_m\rangle$ for all $m \in \bar{M}_h^j$.
    Add c-node $v := \langle Z, h, \mathcal{I} \cup \bigcup_{m \in \bar{M}_h^j} \mathcal{J}_m\rangle$ to $\mathcal{C}$.
    Set $\psi(h)$ and $\{\psi(m) \mid m \in \bar{M}_h^j\}$ as children of v.
    Set $\psi(h) := v$.
  end for
end for

m_j implies that either $m_i \equiv_h m_j$ or $m_i \prec_h m_j$. A weakly-ordered d-tree which is projective and whose orders $\prec_h$ have the nesting property for every h is called a **nested-weakly ordered projective d-tree**. We then have the following result.

Proposition 3. *Continuous unaryless c-trees and nested-weakly ordered projective d-trees are isomorphic.*

Proof. See the supplementary material. □

Together, Propositions 1–3 have as corollary that nested-strictly ordered projective d-trees are in a one-to-one correspondence with binary continuous c-trees. The intuition is simple: if $\prec_h$ has the nesting property, then, at each point in time, all one needs to decide about the next event is whether to attach the closest available modifier on the *left* or on the *right*. This corresponds to choosing between left-branching or right-branching in a c-tree. While this is potentially interesting for most continuous c-parsers, which work with binarized c-trees when running the CKY algorithm, our c-parsers (to be described in §4) do not require any binarization since they work with weakly-ordered d-trees, using Proposition 2.

4 Reduction-Based Constituent Parsers

We now invoke the equivalence results established in §3 to build c-parsers when only a trainable d-parser is available. Given a c-treebank provided as input, our procedure is outlined as follows:

1. Convert the c-treebank to dependencies (Algorithm 1).
2. Train a labeled d-parser on this treebank.
3. For each test sentence, run the labeled d-parser and convert the predicted d-tree into a c-tree without unary nodes (Algorithm 2).
4. Do post-processing to recover unaries.

The next subsections describe each of these steps in detail. Along the way, we illustrate with experiments using the English Penn Treebank (Marcus et al., 1993), which we lexicalized by applying the head rules of Collins (1999).[4]

4.1 Dependency Encoding

The first step is to convert the c-treebank to head-ordered dependencies, which we do using Algorithm 1. If the original treebank has discontinuous c-trees, we end up with non-projective d-trees or with violations of the nested property, as established in Proposition 3. We handle this gracefully by training a non-projective d-parser in the subsequent stage (see §4.2). Note also that this conversion drops the unary nodes (a consequence of Proposition 2). These nodes will be recovered in the last stage, as described in §4.4.

Since in this paper we are assuming that only an off-the-shelf d-parser is available, we need to convert head-ordered d-trees to plain d-trees. We do so by encoding the order information in the dependency labels. We tried two different strategies. The first one, **direct encoding**, just appends suffixes #1, #2, etc., as in Figure 1. A disadvantage is that the number of labels grows unbounded with the treebank size, as we may encounter complex substructures where the event sequences are long. The second strategy is a **delta-encoding** scheme where, rather than writing the absolute indices in the dependency label, we write the *differences* between consecutive ones.[5] We used this strategy for the continuous treebanks only, whose d-trees are guaranteed to satisfy the nested property.

For comparison, we also implemented a replication of the encoding proposed by Hall and Nivre (2008), which we call **H&N-encoding**. This strategy concatenates all the c-nodes' symbols in the modifier's spine with the attachment position in the head's spine (*e.g.*, in Figure 3, if the modifier m_2 has a spine with nodes X_1, X_2, X_3, the generated d-label would be $\texttt{X}_1|\texttt{X}_2|\texttt{X}_3\#2$; our direct encoding scheme generates $\texttt{Z}_2\#2$ instead). Since their strategy encodes the entire spines into com-

[4] We train on §02–21, use §22 for validation, and test on §23. We predict automatic POS tags with *TurboTagger* (Martins et al., 2013), with 10-fold jackknifing on the training set.

[5] For example, if $\#1, \#3, \#4$ and $\#2, \#3, \#3, \#5$ are respectively the sequence of indices from the head to the left and to the right, we encode these sequences as $\#1, \#2, \#1$ and $\#2, \#1, \#0, \#2$ (using 3 distinct indices instead of 5).

plex arc labels, many such labels will be generated, leading to slower runtimes and poorer generalization, as we will see.

For the training portion of the English PTB, which has 27 non-terminal symbols, the direct encoding strategy yields 75 labels, while delta encoding yields 69 labels (2.6 indices per symbol). By contrast, the H&N-encoding procedure yields 731 labels, more than 10 times as many. We later show (in Tables 1–2) that delta-encoding leads to a slightly higher c-parsing accuracy than direct encoding, and that both strategies are considerably more accurate than H&N-encoding.

4.2 Training the Labeled Dependency Parser

The next step is to train a labeled d-parser on the converted treebank. If we are doing continuous c-parsing, we train a projective d-parser; otherwise we train a non-projective one.

In our experiments, we found it advantageous to perform labeled d-parsing in two stages, as done by McDonald et al. (2006): first, train an unlabeled d-parser; then, train a dependency labeler.[6] Table 1 compares this approach against a one-shot strategy, experimenting with various off-the-shelf d-parsers: *MaltParser* (Nivre et al., 2007), *MSTParser* (McDonald et al., 2005), *ZPar* (Zhang and Nivre, 2011), and *TurboParser* (Martins et al., 2013), all with the default settings. For *TurboParser*, we used basic, standard and full models.

Our separate d-labeler receives as input a backbone d-structure and predicts a label for each arc. For each head h, we predict the modifiers' labels using a simple sequence model, with features of the form $\boldsymbol{\phi}(h, m, \ell)$ and $\boldsymbol{\phi}(h, m, m', \ell, \ell')$, where m and m' are two consecutive modifiers (possibly on opposite sides of the head) and ℓ and ℓ' are their labels. We use the same arc label features $\boldsymbol{\phi}(h, m, \ell)$ as *TurboParser*. For $\boldsymbol{\phi}(h, m, m', \ell, \ell')$, we use the POS triplet $\langle p_h, p_m, p_{m'} \rangle$, plus unilexical features where each of the three POS is replaced by the word form. Both features are conjoined with the label pair ℓ and ℓ'. Decoding under this model can be done by running the Viterbi algorithm independently for each head. The runtime is almost negligible compared with the time to parse: it took 2.1 seconds to process PTB §22, a fraction of about 5% of the total runtime.

[6]The reason why a two-stage approach is preferable is that one-shot d-parsers, for efficiency reasons, use label features parsimoniously. However, for our reduction approach, d-labels are crucial and strongly interdependent, since they jointly encode the c-structure.

Dependency Parser	UAS	LAS	F_1	# toks/s.
MaltParser	90.93	88.95	86.87	5,392
MSTParser	92.17	89.86	87.93	363
ZPar	92.93	91.28	89.50	1,022
TP-Basic	92.13	90.23	87.63	2,585
TP-Standard	93.55	91.58	90.41	1,658
TP-Full	93.70	91.70	90.53	959
TP-Full + Lab., H&N enc.	93.80	87.86	89.39	871
TP-Full + Lab, direct enc.	93.80	91.99	90.89	912
TP-Full + Lab., delta enc.	**93.80**	**92.00**	**90.94**	912

Table 1: Results on English PTB §22 achieved by various d-parsers and encoding strategies. For dependencies, we report unlabeled/labeled attachment scores (UAS/LAS), excluding punctuation. For constituents, we show F_1-scores (without punctuation and root nodes), as provided by `EVALB` (Black et al., 1992). We report total parsing speeds in tokens per second (including time spent on pruning, decoding, and feature evaluation), measured on a Intel Xeon processor @2.30GHz.

	direct enc.		**delta enc.**	
	# labels	**F_1**	**# labels**	**F_1**
Basque	26	85.04	17	85.17
French	61	79.93	56	80.05
German	66	83.44	59	83.39
Hebrew	62	83.26	43	83.29
Hungarian	24	86.54	15	86.67
Korean	44	79.79	16	79.97
Polish	47	92.39	34	92.64
Swedish	29	77.02	25	77.19

Table 2: Impact of direct and delta encodings on the dev sets of the SPMRL14 shared task. Reported are the number of labels and the F_1-scores yielded by each encoding technique.

4.3 Decoding into Unaryless Constituents

After training the labeled d-parser, we can run it on the test data. Then, we need to convert the predicted d-tree into a c-tree without unaries.

To accomplish this step, we first need to recover, for each head h, the weak order of its modifiers M_h. We do this by looking at the predicted dependency labels, extracting the event indices j, and using them to build and sort the equivalent classes $\{\bar{M}_h^j\}_{j=1}^J$. If two modifiers have the same index j, we force them to have consistent labels (by always choosing the label of the modifier which is the closest to the head). For continuous c-parsing, we also decrease the index j of the modifier closer to the head as much as necessary to make sure that the nesting property holds. In PTB §22, these corrections were necessary only for 0.6% of the tokens. Having done this, we use Algorithm 2 to obtain a predicted c-tree without unary nodes.

4.4 Recovery of Unary Nodes

The last stage is to recover the unary nodes. Given a unaryless c-tree as input, we predict unaries by running independent classifiers at each node in the tree (a simple unstructured task). Each class is either NULL (in which case no unary node is appended to the current node) or a concatenation of unary node labels (*e.g.*, S->ADJP for a node JJ). We obtained 64 classes by processing the training sections of the PTB, the fraction of unary nodes being about 11% of the total number of nodes. To reduce complexity, for each node symbol we only consider classes that have been observed with that symbol in the training data. In PTB §22, this yields an average of 9.9 candidates per node occurrence.

The classifiers are trained on the original c-treebank, stripping off unary nodes and trained to recover those nodes. We used the following features (conjoined with the class and with a flag indicating if the node is a pre-terminal):

- The production rules above and beneath the node (*e.g.*, S->NP VP and NP->DT NN);
- The node's label, alone and conjoined with the parent's label or the left/right sibling's label;
- The leftmost and rightmost word/lemma/POS tag/morpho-syntactic tags in the node's yield;
- If the left/right node is a pre-terminal, the word/lemma/morpho-syntactic tags beneath.

This is a relatively easy task: when gold unaryless c-trees are provided as input, we obtain an EVALB F_1-score of 99.43%. This large figure is due to the small amount of unary nodes, making this module have less impact on the final parser than the d-parser. Being a lightweight unstructured task, this step took only 0.7 seconds to run on PTB §22, a tiny fraction (less than 2%) of the total runtime.

Table 1 shows the accuracies obtained with the d-parser followed by the unary predictor. Since two-stage TP-Full with delta-encoding is the best strategy, we use this configuration in the sequel. To further explore the impact of delta encoding, we report in Table 2 the scores obtained by direct and delta encodings on eight other treebanks (see §5.2 for details on these datasets). With the exception of German, in all cases the delta encoding yielded better EVALB F_1-scores with fewer labels.

5 Experiments

To evaluate the performance of our reduction-based parsers, we conduct experiments in a variety of treebanks, both continuous and discontinuous.

Parser	LR	LP	F1	#Toks/s.
Charniak (2000)	89.5	89.9	89.5	–
Klein and Manning (2003)	85.3	86.5	85.9	143
Petrov and Klein (2007)	90.0	90.3	90.1	169
Carreras et al. (2008)	90.7	91.4	91.1	–
Zhu et al. (2013)	90.3	90.6	90.4	1,290
Stanford Shift-Reduce (2014)	89.1	89.1	89.1	655
Hall et al. (2014)	88.4	88.8	88.6	12
This work	89.9	90.4	90.2	957
Charniak and Johnson (2005)*	91.2	91.8	91.5	84
Socher et al. (2013)*	89.1	89.7	89.4	70
Zhu et al. (2013)*	91.1	91.5	91.3	–

Table 3: Results on the English PTB §23. All systems reporting runtimes were run on the same machine. Marked as * are reranking and semi-supervised c-parsers.

5.1 Results on the English PTB

Table 3 shows the accuracies and speeds achieved by our system on the English PTB §23, in comparison to state-of-the-art c-parsers. We can see that our simple reduction-based c-parser surpasses the three Stanford parsers (Klein and Manning, 2003; Socher et al., 2013, and Stanford Shift-Reduce), and is on par with the Berkeley parser (Petrov and Klein, 2007), while being more than 5 times faster.

The best supervised competitor is the recent shift-reduce parser of Zhu et al. (2013), which achieves similar, but slightly better, accuracy and speed. Our technique has the advantage of being flexible: since the time for d-parsing is the dominating factor (see §4.4), plugging a faster d-parser automatically yields a faster c-parser. While reranking and semi-supervised systems achieve higher accuracies, this aspect is orthogonal, since the same techniques can be applied to our parser.

5.2 Results on the SPMRL Datasets

We experimented with datasets for eight languages, from the SPMRL14 shared task (Seddah et al., 2014). We used the official training, development and test sets with the provided predicted POS tags. For French and German, we used the lexicalization rules detailed in Dybro-Johansen (2004) and Rehbein (2009), respectively. For Basque, Hungarian and Korean, we always took the rightmost modifier as head-child node. For Hebrew and Polish we used the leftmost modifier instead. For Swedish we induced head rules from the provided dependency treebank, as described in Versley (2014b). These choices were based on dev-set experiments.

Table 4 shows the results. For all languages ex-

cept French, our system outperforms the Berkeley parser (Petrov and Klein, 2007), with or without prescribed POS tags. Our average F_1-scores are superior to the best non-reranking system participating in the shared task (Crabbé and Seddah, 2014) and to the c-parser of Hall et al. (2014), achieving the best results for 4 out of 8 languages.

5.3 Results on the Discontinuous Treebanks

Finally, we experimented on two widely-used discontinuous German treebanks: TIGER (Brants et al., 2002) and NEGRA (Skut et al., 1997). For the former, we used two different splits: TIGER-SPMRL, provided in the SPMRL14 shared task; and TIGER-H&N, used by Hall and Nivre (2008). For NEGRA, we used the standard splits. In these experiments, we skipped the unary recovery stage, since very few unary nodes exist in the data.[7] We ran *TurboTagger* to predict POS tags for TIGER-H&N and NEGRA, while in TIGER-SPMRL we used the predicted POS tags provided in the shared task. All treebanks were lexicalized using the head-rule sets of Rehbein (2009). For comparison to related work, sentence length cut-offs of 30, 40 and 70 were applied during the evaluation.

Table 5 shows the results. We observe that our approach outperforms all the competitors considerably, achieving state-of-the-art accuracies for both datasets. The best competitor, van Cranenburgh and Bod (2013), is more than 3 points behind, both in TIGER-H&N and in NEGRA. Our reduction-based parsers are also much faster: van Cranenburgh and Bod (2013) report 3 hours to parse NEGRA with $L \leq 40$. Our system parses all NEGRA sentences (regardless of length) in 27.1 seconds in a single core, which corresponds to a rate of 618 tokens per second. This approaches the speed of the easy-first system of Versley (2014a), who reports runtimes in the range 670–920 tokens per second, but is much less accurate.

TIGER-SPMRL	$L \leq 70$	all
V14b, *gold*	76.46 / 41.05	76.11 / 40.94
Ours, *gold*	**80.98 / 43.44**	**80.62 / 43.32**
V14b, *pred*	73.90 / 37.00	– / –
Ours, *pred*	**77.72 / 38.75**	**77.32 / 38.64**
TIGER-H&N	$L \leq 40$	all
HN08, *gold*	79.93 / 37.78	– / –
V14a, *gold*	74.23 / 37.32	– / –
Ours, *gold*	**85.53 / 51.21**	**84.22 / 49.63**
HN08, *pred*	75.33 / 32.63	– / –
CB13, *pred*	78.8– / 40.8–	– / –
Ours, *pred*	**82.57 / 45.93**	**81.12 / 44.48**
NEGRA	$L \leq 30$	$L \leq 40$	all
M12, *gold*	74.5– / –	– / –	– / –
C12, *gold*	– / –	72.33 / 33.16	71.08 / 32.10
KM13, *gold*	75.75 / –	– / –	– / –
CB13, *gold*	– / –	76.8– / 40.5–	– / –
Ours, *gold*	**82.56 / 52.13**	**81.08 / 48.04**	**80.52 / 46.70**
CB13, *pred*	– / –	74.8– / 38.7–	– / –
Ours, *pred*	**79.63 / 48.43**	**77.93 / 44.83**	**76.95 / 43.50**

Table 5: F_1 / exact match scores on TIGER and NEGRA test sets, with gold and predicted POS tags. These scores are computed by the DISCO-DOP evaluator ignoring root nodes and, for TIGER-H&N and NEGRA, punctuation tokens. The baselines are published results by Hall and Nivre 2008 (HN08), Maier et al. 2012 (M12), van Cranenburgh 2012 (C12), Kallmeyer and Maier 2013 (KM13), van Cranenburgh and Bod 2013 (CB13), and Versley 2014a, 2014b (V14a, V14b).

6 Related Work

Conversions between constituents and dependencies have been considered by De Marneffe et al. (2006) in one direction, and by Collins et al. (1999) and Xia and Palmer (2001) in the other, toward multi-representational treebanks (Xia et al., 2008). This prior work aimed at linguistically sound conversions, involving grammar-specific transformation rules to handle the kind of ambiguities expressed in Figure 2. Our work differs in that we are not concerned about the linguistic plausibility of our conversions, but only with the formal aspects that underlie the two representations.

The work most related to ours is Hall and Nivre (2008), who also convert dependencies to constituents to prototype a c-parser for German. Their encoding strategy is compared to ours in §4.1: they encode the entire spines into the dependency labels, which become rather complex and numerous. A similar strategy has been used by Versley (2014a) for discontinuous c-parsing. Both are largely outperformed by our system, as shown in §5.3. The crucial difference is that we encode only the top node's label and its position in the spine—besides being a much lighter representation, ours has an interpretation as a weak ordering, leading to the isomorphisms expressed in Propositions 1–3.

Joint constituent and dependency parsing have been tackled by Carreras et al. (2008) and Rush et al. (2010), but the resulting parsers, while accurate, are more expensive than a single c-parser. Very recently, Kong et al. (2015) proposed a much cheaper pipeline in which d-parsing is performed first, followed by a c-parser constrained to be con-

[7] NEGRA has no unaries; for the TIGER-SPMRL and H&N dev-sets, the fraction of unaries is 1.45% and 1.01%.

Parser	Basque	French	German	Hebrew	Hungar.	Korean	Polish	Swedish	Avg.
Berkeley	70.50	**80.38**	78.30	86.96	81.62	71.42	79.23	79.19	78.45
Berkeley Tagged	74.74	79.76	78.28	85.42	85.22	78.56	86.75	80.64	81.17
Hall et al. (2014)	83.39	79.70	78.43	87.18	**88.25**	**80.18**	90.66	82.00	83.72
Crabbé and Seddah (2014)	85.35	79.68	77.15	86.19	87.51	79.35	**91.60**	82.72	83.69
This work	**85.90**	78.75	**78.66**	**88.97**	88.16	79.28	91.20	**82.80**	**84.22**
Björkelund et al. (2014)	88.24	82.53	81.66	89.80	91.72	83.81	90.50	85.50	86.72

Table 4: F_1-scores on eight treebanks of the SPMRL14 shared task, computed with the provided `EVALB_SPMRL` tool, which takes into account all tokens except root nodes. Berkeley Tagged is a version of Petrov and Klein (2007) using the predicted POS tags provided by the organizers. Crabbé and Seddah (2014) is the best non-reranking system in the shared task, and Björkelund et al. (2014) the ensemble and reranking-based system which won the official task. We report their published scores.

sistent with the predicted d-structure. Our work differs in which we do not need to run a c-parser in the second stage—instead, the d-parser already stores constituent information in the arc labels, and the only necessary post-processing is to recover unary nodes. Another advantage of our method is that it can be readily used for discontinuous parsing, while their constrained CKY algorithm can only produce continuous parses.

7 Conclusion

We proposed a reduction technique that allows to implement a c-parser when only a d-parser is given. The technique is applicable to any d-parser, regardless of its nature or kind. This reduction was accomplished by endowing d-trees with a weak order relation, and showing that the resulting class of head-ordered d-trees is isomorphic to constituent trees. We showed empirically that the our reduction leads to highly-competitive c-parsers for English and for eight morphologically rich languages; and that it outperforms the current state of the art in discontinuous parsing of German.

Acknowledgments

We would like to thank the three reviewers for their insightful comments, and Slav Petrov, Djamé Seddah, Yannick Versley, David Hall, Muhua Zhu, Lingpeng Kong, Carlos Gómez-Rodríguez, and Andreas van Cranenburgh for valuable feedback and help in preparing data and running software code. This research has been partially funded by the Spanish Ministry of Economy and Competitiveness and FEDER (project TIN2010-18552-C03-01), Ministry of Education (FPU Grant Program) and Xunta de Galicia (projects R2014/029 and R2014/034). A. M. was supported by the EU/FEDER programme, QREN/POR Lisboa (Portugal), under the Intelligo project (contract 2012/24803), and by the FCT grants UID/EEA/50008/2013 and PTDC/EEI-SII/2312/2012.

References

Taylor Berg-Kirkpatrick, Dan Gillick, and Dan Klein. 2011. Jointly learning to extract and compress. In *Proc. of Annual Meeting of the Association for Computational Linguistics*.

Anders Björkelund, Özlem Çetinoğlu, Agnieszka Faleńska, Richárd Farkas, Thomas Mueller, Wolfgang Seeker, and Zsolt Szántó. 2014. Introducing the ims-wrocław-szeged-cis entry at the spmrl 2014 shared task: Reranking and morpho-syntax meet unlabeled data. In *Proc. of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*.

Ezra Black, John Lafferty, and Salim Roukos. 1992. Development and evaluation of a broad-coverage probabilistic grammar of english-language computer manuals. In *Proc. of Annual Meeting on Association for Computational Linguistics*.

Adriane Boyd. 2007. Discontinuity revisited: An improved conversion to context-free representations. In *Proc. of Linguistic Annotation Workshop*.

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2002. The TIGER treebank. In *Proc. of the workshop on treebanks and linguistic theories*.

Xavier Carreras, Michael Collins, and Terry Koo. 2008. TAG, Dynamic Programming, and the Perceptron for Efficient, Feature-rich Parsing. In *Proc. of the International Conference on Natural Language Learning*.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proc. of Annual Meeting of the Association for Computational Linguistics*.

Eugene Charniak. 1996. Tree-bank grammars. In *Proc. of the National Conference on Artificial Intelligence*.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proc. of the North American Chapter of the Association for Computational Linguistics Conference*.

Michael Collins, Lance Ramshaw, Jan Hajič, and Christoph Tillmann. 1999. A Statistical Parser for Czech. In *Proc. of the Annual Meeting of the Association for Computational Linguistics on Computational Linguistics*.

Michael Collins. 1999. *Head-driven statistical models for natural language parsing*. Ph.D. thesis, University of Pennsylvania.

Benoit Crabbé and Djamé Seddah. 2014. Multilingual discriminative shift reduce phrase structure parsing for the SPMRL 2014 shared task. In *Proc. of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*.

Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. 2006. Generating typed dependency parses from phrase structure parses. In *Proc. of the Meeting of the Language Resources and Evaluation Conference*.

Ane Dybro-Johansen. 2004. Extraction automatique de Grammaires d'Arbres Adjoints à partir d'un corpus arboré du français. Master's thesis, Université Paris 7.

Jason Eisner and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proc. of Annual Meeting of the Association for Computational Linguistics*.

Jason Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proc. of International Conference on Computational Linguistics*.

Jakob Elming, Anders Johannsen, Sigrid Klerke, Emanuele Lapponi, Hector Martinez Alonso, and Anders Søgaard. 2013. Down-stream effects of tree-to-dependency conversions. In *Proc. of the Annual Conference of the Human Language Technologies - North American Chapter of the Association for Computational Linguistics*.

Haim Gaifman. 1965. Dependency systems and phrase-structure systems. *Information and control*.

Carlos Gómez-Rodríguez, Marco Kuhlmann, and Giorgio Satta. 2010. Efficient parsing of well-nested linear context-free rewriting systems. In *Proc. of the Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

Johan Hall and Joakim Nivre. 2008. A dependency-driven parser for german dependency and constituency representations. In *Proc. of the Workshop on Parsing German*.

David Hall, Greg Durrett, and Dan Klein. 2014. Less grammar, more features. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*.

Richard Johansson and Pierre Nugues. 2008. Dependency-based Semantic Role Labeling of PropBank. In *Empirical Methods for Natural Language Processing*.

Mark Johnson. 1998. PCFG models of linguistic tree representations. *Computational Linguistics*.

Sylvain Kahane, Alexis Nasr, and Owen Rambow. 1998. Pseudo-projectivity: a polynomially parsable non-projective dependency grammar. In *Proc. of the International Conference on Computational Linguistics*.

Laura Kallmeyer and Wolfgang Maier. 2013. Data-driven parsing using probabilistic linear context-free rewriting systems. *Computational Linguistics*.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proc. of Annual Meeting on Association for Computational Linguistics*.

Lingpeng Kong and Noah A Smith. 2014. An empirical comparison of parsing methods for stanford dependencies. *arXiv preprint arXiv:1404.4314*.

Lingpeng Kong, Alexander M. Rush, and Noah A. Smith. 2015. Transforming dependencies into phrase structures. In *Proc. of the Conference of the North American Chapter of the Association for Computational Linguistics*.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proc. of Annual Meeting of the Association for Computational Linguistics*.

Sandra Kübler, Wolfgang Maier, Ines Rehbein, and Yannick Versley. 2008. How to compare treebanks. In *Proc. of the Meeting of the Language Resources and Evaluation Conference*.

Marco Kuhlmann and Joakim Nivre. 2006. Mildly non-projective dependency structures. In *Proc. of the joint conference of the International Committee on Computational Linguistics and the Association for Computational Linguistics*.

Wolfgang Maier and Anders Søgaard. 2008. Treebanks and mild context-sensitivity. In *Proc. of Formal Grammar*.

Wolfgang Maier, Miriam Kaeshammer, and Laura Kallmeyer. 2012. Data-driven plcfrs parsing revisited: Restricting the fan-out to two. In *Proc. of the Eleventh International Conference on Tree Adjoining Grammars and Related Formalisms*.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*.

André F. T. Martins, Miguel B. Almeida, and Noah A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Probabilistic CFG with latent annotations. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*.

Ryan McDonald and Giorgio Satta. 2007. On the complexity of non-projective data-driven dependency parsing. In *Proc. of International Conference on Parsing Technologies*.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-projective dependency pars-

ing using spanning tree algorithms. In *Proc. of Empirical Methods for Natural Language Processing*.

Ryan McDonald, Kevin Lerman, and Fernando Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proc. of International Conference on Natural Language Learning*.

Joakim Nivre, Johan Hall, Jens Nilsson, Gülsen Eryiğit, and Svetoslav Marinov. 2006. Labeled pseudo-projective dependency parsing with support vector machines. In *Proc. of International Conference on Natural Language Learning*.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryiğit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*.

Fernando C. N. Pereira and David H. D. Warren. 1983. Parsing as Deduction. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proc. of the North American Chapter of the Association for Computational Linguistics*.

Ines Rehbein. 2009. *Treebank-Based Grammar Acquisition for German*. Ph.D. thesis, School of Computing, Dublin City University.

Alexander M Rush and Slav Petrov. 2012. Vine pruning for efficient multi-pass dependency parsing. In *Proc. of the North American Chapter of the Association for Computational Linguistics*.

Alexander Rush, David Sontag, Michael Collins, and Tommi Jaakkola. 2010. On dual decomposition and linear programming relaxations for natural language processing. In *Proc. of Empirical Methods for Natural Language Processing*.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proc. of the Ninth International Workshop on Parsing Technology*.

Djamé Seddah, Sandra Kübler, and Reut Tsarfaty. 2014. Introducing the spmrl 2014 shared task on parsing morphologically-rich languages. In *Proc. of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, August.

Wojciech Skut, Brigitte Krenn, Thorsten Brants, and Hans Uszkoreit. 1997. An annotation scheme for free word order languages. In *Proc. of the Fifth Conference on Applied Natural Language Processing ANLP-97*.

Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. 2013. Parsing with compositional vector grammars. In *Proc. of Annual Meeting of the Association for Computational Linguistics*.

Andreas van Cranenburgh and Rens Bod. 2013. Discontinuous parsing with an efficient and accurate dop model. *Proc. of International Conference on Parsing Technologies*.

Andreas van Cranenburgh. 2012. Efficient parsing with linear context-free rewriting systems. In *Proc. of the Conference of the European Chapter of the Association for Computational Linguistics*.

Yannick Versley. 2014a. Experiments with easy-first nonprojective constituent parsing. In *Proc. of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*.

Yannick Versley. 2014b. Incorporating semi-supervised features into discontinuous easy-first constituent parsing. *CoRR*, abs/1409.3813.

Krishnamurti Vijay-Shanker, David J Weir, and Aravind K Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *Proc. of the Annual Meeting on Association for Computational Linguistics*.

Yuanbin Wu, Qi Zhang, Xuanjing Huang, and Lide Wu. 2009. Phrase dependency parsing for opinion mining. In *Proc. of Empirical Methods for Natural Language Processing*.

Fei Xia and Martha Palmer. 2001. Converting dependency structures to phrase structures. In *Proc. of the First International Conference on Human Language Technology Research*.

Fei Xia, Owen Rambow, Rajesh Bhatt, Martha Palmer, and Dipti Misra Sharma. 2008. Towards a multi-representational treebank. *LOT Occasional Series*.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proc. of International Conference on Parsing Technologies*.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Proc. of Annual Meeting of the Association for Computational Linguistics*.

A Supplementary Material

A.1 Proof of Proposition 1

We will show that, given an arbitrary strictly-ordered d-tree $\mathcal{D}$, we can perform an invertible transformation to turn it into a binary c-tree $\mathcal{C}$; and vice-versa. Let $\mathcal{D}$ be given. We visit each node $h \in \{1, \ldots, L\}$ and split it into $K + 1$ nodes, where $K = |M_h|$, organized as a linked list, as Figure 3 illustrates (this will become the spine of h in the c-tree). For each modifier $m_k \in M_h$ with $m_1 \prec_h \ldots \prec_h m_K$, move the tail of the arc $\langle h, m_k, Z_k \rangle$ to the $(K + 1 - k)$th node of the linked list and assign the label Z_k to this node, letting h be its lexical head. Since the incoming and outgoing arcs of the linked list component are the same as in the original node h, the tree structure is preserved. After doing this for every h, add the leaves and propagate the yields bottom up. It is straightforward to show that this procedure yields a valid binary c-tree. Since there is no loss of information (the orders $\prec_h$ are implied by the order of the nodes in each spine), this construction can be inverted to recover the original d-tree. Conversely, if we start with a binary c-tree, traverse the spine of each h, and attach the modifiers $m_1 \prec_h \ldots \prec_h m_K$ in order, we get a strictly ordered d-tree (also an invertible procedure).

A.2 Proof of Proposition 3

We need to show that (i) Algorithm 1, when applied to a continuous c-tree $\mathcal{C}$, retrieves a head ordered d-tree $\mathcal{D}$ which is projective and has the nesting property, (ii) vice-versa for Algorithm 2. To see (i), note that the projectiveness of $\mathcal{D}$ is ensured by the well-known result of Gaifman (1965) about the projection of continuous trees. To show that it satisfies the nesting property, note that nodes higher in the spine of a word h are always attached by modifiers farther apart (otherwise edges in $\mathcal{C}$ would cross, which cannot happen for a continuous $\mathcal{C}$). To prove (ii), we use induction. We need to show that every created c-node in Algorithm 2 has a contiguous span as yield. The base case (line 3) is trivial. Therefore, it suffices to show that in line 8, assuming the yields of (the current) $\psi(h)$ and each $\psi(m)$ are contiguous spans, the union of these yields is also contiguous. Consider the node v when these children have been appended (line 9), and choose $m \in \bar{M}_h^j$ arbitrarily. We only need to show that for any d between h and m, d belongs to the yield of v. Since $\mathcal{D}$ is projective and there is a d-arc between h and m, we have that d must descend from h. Furthermore, since projective trees cannot have crossing edges, we have that h has a unique child a, also between h and m, which is an ancestor of d (or d itself). Since a is between h and m, from the nesting property, we must have $\langle h, m, \ell \rangle \not\prec_h \langle h, a, \ell' \rangle$ Therefore, since we are processing the modifiers in order, we have that $\psi(a)$ is already a descendent of v after line 9, which implies that the yield of $\psi(a)$ (which must include d, since d descends from a) must be contained in the yield of v.