Discontinuous Incremental Shift-Reduce Parsing

Wolfgang Maier
Universität Düsseldorf
Institut für Sprache und Information
Universitätsstr. 1, 40225 Düsseldorf, Germany
maierw@hhu.de

Abstract

We present an extension to incremental shift-reduce parsing that handles discontinuous constituents, using a linear classifier and beam search. We achieve very high parsing speeds (up to 640 sent./sec.) and accurate results (up to 79.52 F_1 on TiGer).

1 Introduction

Discontinuous constituents consist of more than one continuous block of tokens. They arise through phenomena which traditionally in linguistics would be analyzed as being the result of some kind of "movement", such as extraposition or topicalization. The occurrence of discontinuous constituents does not necessarily depend on the degree of freedom in word order that a language allows for. They can be found, e.g., in almost equal proportions in English and German treebank data (Evang and Kallmeyer, 2011).

Generally, discontinuous constituents are accounted for in treebank annotation. One annotation method consists of using trace nodes that denote the source of a movement and are co-indexed with the moved constituent. Another method is to annotate discontinuities directly by allowing for crossing branches. Fig. 1 shows an example for the latter approach with which we are concerned in this paper, namely, the annotation of (1). The tree contains a discontinuous VP due to the fact that the fronted pronoun is directly attached.

(1) Das wollen wir umkehren
That want we reverse
"This is what we want to reverse"

Several methods have been proposed for parsing such structures. Trace recovery can been framed as a separate pre-, post- or in-processing task to PCFG parsing (Johnson, 2002; Dienes and Dubey, 2003; Jijkoun, 2003; Levy and Manning, 2004; Schmid, 2006; Cai et al., 2011, among others); see particularly Schmid (2006) for more details. Directly annotated discontinuous constituents can be parsed with a dependency parser, given a reversible transformation from discontinuous constituency trees to non-projective dependency structures. Transformations have been proposed by Hall and Nivre (2008), who use complex edge labels that encode paths between lexical heads, and recently by Fernández-González and Martins (2015), who use edge labels to encode the attachment order of modifiers to heads.

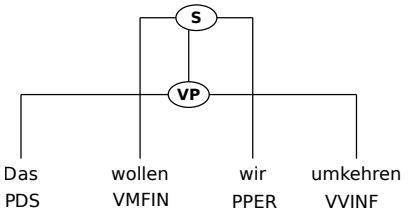

Figure 1: Example annotation with discontinuous constituents from TiGer

Direct parsing of discontinuous constituents can be done with Linear Context-Free Rewriting System (LCFRS), an extension of CFG which allows its non-terminals to cover more than one continuous block (Vijay-Shanker et al., 1987). LCFRS parsing is expensive: CYK chart parsing with a binarized grammar can be done in $\mathcal{O}(n^{3k})$ where k is the *block degree*, the maximal number of continuous blocks a non-terminal can cover (Seki et al., 1991). For a typical treebank LCFRS (Maier and Søgaard, 2008), $k \approx 3$, instead of $k = 1$ for PCFG. In order to improve on otherwise impractical parsing times, LCFRS chart parsers employ different strategies to speed up search: Kallmeyer

Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pages 1202–1212,
Beijing, China, July 26-31, 2015.

and Maier (2013) use A* search; van Cranenburgh (2012) and van Cranenburgh and Bod (2013) use a coarse-to-fine strategy in combination with Data-Oriented Parsing; Angelov and Ljunglöf (2014) use a novel cost estimation to rank parser items. Maier et al. (2012) apply a treebank transformation which limits the block degree and therewith also the parsing complexity.

Recently Versley (2014) achieved a breakthrough with a *EaFi*, a classifier-based parser that uses an "easy-first" approach in the style of Goldberg and Elhadad (2010). In order to obtain discontinuous constituents, the parser uses a strategy known from non-projective dependency parsing (Nivre, 2009; Nivre et al., 2009): For every non-projective dependency tree, there is a projective dependency tree which can be obtained by reordering the input words. Non-projective dependency parsing can therefore be viewed as projective dependency parsing with an additional reordering of the input words. The reordering can be done online during parsing with a "swap" operation that allows to process input words out of order. This idea can be transferred, because also for every discontinuous constituency tree, one can find a continuous tree by reordering the terminals. Versley (2014) uses an adaptive gradient method to train his parser. He reports a parsing speed of 40-55 sent./sec. and results that surpass those reported for the above mentioned chart parsers.

In (continuous) constituency parsing, incremental shift-reduce parsing using the structured perceptron is an established technique. While the structured perceptron for parsing has first been used by Collins and Roark (2004), classifier-based incremental shift-reduce parsing has been taken up by Sagae and Lavie (2005). A general formulation for the application of the perceptron algorithm to various problems, including shift-reduce constituency parsing, has been introduced by Zhang and Clark (2011b). Improvements have followed (Zhu et al., 2012; Zhu et al., 2013). A similar strategy has been shown to work well for CCG parsing (Zhang and Clark, 2011a), too.

In this paper, we contribute a perceptron-based shift-reduce parsing architecture with beam search (following Zhu et al. (2013) and Bauer (2014)) and extend it such that it can create trees with crossing branches (following Versley (2014)). We present strategies to improve performance on discontinuous structures, such as a new feature set. Our parser is very fast (up to 640 sent./sec.), and produces accurate results. In our evaluation, where we pay particular attention to the parser performance on discontinuous structures, we show among other things that surprisingly, a grammar-based parser has an edge over a shift-reduce approach concerning the reconstruction of discontinuous constituents.

The remainder of the paper is structured as follows. In subsection 2.1, we introduce the general parser architecture; the subsections 2.2 and 2.3 introduce the features we use and our strategy for handling discontinuous structures. Section 3 presents and discusses the experimental results, section 4 concludes the article.

2 Discontinuous Shift-Reduce Parsing

Our parser architecture follows previous work, particularly Zhu et al. (2013) and Bauer (2014).

2.1 Shift-reduce parsing with perceptron training

An *item* in our parser consists of a *queue* q of token/POS-pairs to be processed, and a *stack* s, which holds completed constituents.[1] The parser uses different transitions: SHIFT shifts a terminal from the queue on to the stack. UNARY-X reduces the first element on the stack to a new constituent labeled *X*. BINARY-X-L and BINARY-X-R reduce the first two elements on the stack to a new *X* constituent, with the lexical head coming from the left or the right child, respectively. FINISH removes the last element from the stack. We additionally use an IDLE transition, which can be applied any number of times after FINISH, to improve the comparability of analyses of different lengths (Zhu et al., 2013).

The application of a transition is subject to restrictions. UNARY-X, e.g., can only be applied when there is at least a single item on the stack. We implement all restrictions listed in the appendix of Zhang and Clark (2009), and add additional restrictions that block transitions involving the root label when not having arrived at the end of a derivation. We do not use an underlying grammar to filter out transitions which have not been seen during training.

For decoding, we use beam search (Zhang and Clark, 2011b). Decoding is started by putting the

[1] As in other shift-reduce approaches, we assume that POS tagging is done outside of the parser.

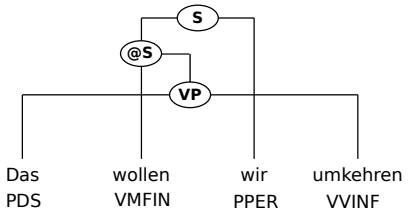

Figure 2: Binarization example

start item (empty stack, full queue) on the beam. Then, repeatedly, a candidate list is filled with all items that result from applying legal transitions to the items on the beam, followed by putting the highest scoring n of them back on the beam (given a beam size of n). Parsing is finished if the highest scoring item on the beam is a final item (stack holds one item labeled with the root label, queue is empty), which can be popped. Item scores are computed as in Zhang and Clark (2011b): The score of the $i+1$th item is computed as the sum of the score of the ith item and the dot product of a global feature weight vector and the local weight vector resulting from the changes induced by the corresponding transition to the $i+1$th item. The start item has score 0. We train the global weight vector with an averaged Perceptron with early update (Collins and Roark, 2004).

Parsing relies on binary trees. As in previous work, we binarize the incoming trees head-outward with binary top and bottom productions. Given a constituent X which is to be binarized, all intermediate nodes which are introduced will be labeled $@X$. Lexical heads are marked with Collins-style head rules. As an example, Fig. 2 shows the binarized version of the tree of Fig. 1.

Finally, since we are learning a sparse model, we also exploit the work of Goldberg and Elhadad (2011) who propose to include a feature in the calculation of a score only if it has been observed $\geq$ MINUPDATE times.

2.2 Features

Features are generated by applying templates to parser items. They reflect different configurations of stack and queue. As BASELINE features, we use the feature set from Zhang and Clark (2009) without the bracketing features (as used in Zhu et al. (2013)). We furthermore experiment with features that reflect the presence of separating punctuation ",", ":", ";" (SEPARATOR) (Zhang and Clark, 2009), and with the EXTENDED features of Zhu et al. (2013), which look deeper into the trees on the stack, i.e., up to the grand-children instead of only to children.

unigrams
$s_0tc, s_0wc, s_1tc, s_1wc, s_2tc, s_2wc, s_3tc, s_3wc,$
$q_0wt, q_1wt, q_2wt, q_3wt,$
$s_0lwc, s_0rwc, s_0uwc, s_1lwc, s_1rwc, s_1uwc$
bigrams
$s_0ws_1w, s_0ws_1c, s_0cs_1w, s_0cs_1c, s_0wq_0w, s_0wq_0t,$
$s_0cq_0w, s_0cq_0t, s_1wq_0w, s_1wq_0t, s_1cq_0w, s_1cq_0t,$
$q_0wq_1w, q_0wq_1t, q_0tq_1w, q_0tq_1t$
trigrams
$s_0cs_1cs_2w, s_0cs_1cs_2c, s_0cs_1cq_0w, s_0cs_1cq_0t,$
$s_0cs_1wq_0w, s_0cs_1wq_0t, s_0ws_1cs_2c, s_0ws_1cq_0t$
extended
$s_0llwc, s_0lrwc, s_0luwc, s_0rlwc, s_0rrwc,$
$s_0ruwc, s_0ulwc, s_0urwc, s_0uuwc, s_1llwc,$
$s_1lrwc, s_1luwc, s_1rlwc, s_1rrwc, s_1ruwc$
separator
$s_0wp, s_0wcp, s_0wq, s_0wcq, s_0cs_1cp, s_0cs_1cq$
$s_1wp, s_1wcp, s_1wq, s_1wcq$

Figure 3: Feature templates

Fig. 3 shows all the feature templates. Note that s_i and q_i stands for the ith stack and queue item, w stands for the head word, t for the head tag and c for the constituent label (w, t and c are identical on POS-level). l and r (ll and rr) represent the left and right children (grand-children) of the element on the stack; u handles the unary case. Concerning the separator features, p is a unique separator punctuation between the head words of s_0 and s_1, q is the count of any separator punctuation between s_0 and s_1.

2.3 Handling Discontinuities

In order to handle discontinuities, we use two variants of a swap transition which are similar to *swap-eager* and *swap-lazy* from Nivre (2009) and Nivre et al. (2009). The first variant, SINGLESWAP, swaps the second item of the stack back on the queue. The second variant COMPOUNDSWAP$_i$ bundles a maximal number of adjacent swaps. It swaps i items starting from the second item on the stack, with $1 \leq i < |s|$. Both swap operations can only be applied if

1. the item has not yet been FINISHed and the last transition has not been a transition with the root category,
2. the queue is not empty,
3. all elements to be swapped are pre-terminals, and

4. if the first item of the stack has a lower index than the second (this inhibits swap loops).

SINGLESWAP can only been applied if there are at least two items on the stack. For COMPOUNDSWAP$_i$, there must be at least $i + 1$ items.

Transition sequences are extracted from treebank trees with an algorithm that traverses the tree bottom-up and collects the transitions. For a given tree τ, intuitively, the algorithm works as follows. We start out with a queue t containing the preterminals of τ, a stack σ that receives finished constituents, a counter s that keeps track of the number of terminals to be swapped, and an empty sequence r that holds the result. First, the first element of t is pushed on σ and removed from t.

While $|\sigma| > 0$ or $|t| > 0$, we repeat the following two steps.

1. Repeat while transitions can be added:
   (a) if the top two elements on σ, l and r, have the same parent p labeled X and l/r is the head of p, add BINARY-X-l/r to r, pop two elements from σ and push p;
   (b) if the top element on σ is the only child of its parent p labeled X, add UNARY-X, pop an element of σ and push p.
2. If $|t| > 0$, while the first element of t is not equal to the leftmost pre-terminal dominated by the right child of the parent of the top element on σ (i.e., while there are terminals that must be swapped), add SHIFT to r, increment s, push the first element of t on σ and remove it from t. Finally, add another SHIFT to r, push first element of t to σ and remove it from t (this will contribute to the next reduction). If $s > 0$, we must swap. Either we add s many SWAP transitions or one COMPOUNDSWAP$_s$ to r. Then we move s many elements from σ to the front of t, starting with the second element of σ. Finally we set $s = 0$.

As an example, consider the transition sequence we would extract from the tree in Fig. 2. Using SINGLESWAP, we would obtain SHIFT, SHIFT, SHIFT, SHIFT, SINGLESWAP, SINGLESWAP, BINARY-VP-R, SHIFT, BINARY-@S-R, SHIFT, BINARY-S-L, FINISH. Using COMPOUNDSWAP$_i$, instead of two SINGLESWAPs, we would just obtain a single COMPOUNDSWAP$_2$.

unigrams
s_0xwc, s_1xwc, s_2xwc, s_3xwc,
s_0xtc, s_1xwc, s_2xtc, s_3xwc,
s_0xy, s_1xy, s_2xy, s_3xy
bigrams
s_0xs_1c, s_0xs_1w, s_0xs_1x, s_0ws_1x, s_0cs_1x,
s_0xs_2c, s_0xs_2w, s_0xs_2x, s_0ws_2x, s_0cs_2x,
s_0ys_1y, s_0ys_2y, s_0xq_0t, s_0xq_0w

Figure 4: Features for discontinuous structures

We explore two methods which improve the performance on discontinuous structures. Even though almost a third of all sentences in the German NeGra and TiGer treebanks contains at least one discontinuous constituent, among all constituents, the discontinuous ones are rare, making up only around 2%. The first, simple method addresses this sparseness by raising the importance of the features that model the actual discontinuities by counting all feature occurrences at a gold swap transition twice (IMPORTANCE).

Secondly, we use a new feature set (DISCO) with bigram and unigram features that conveys information about discontinuities. The features condition the possible occurrence of a gap on previous gaps and their properties.[2] The feature templates are shown in Fig. 4. x denotes the gap type of a tree on the stack. There are three possible values, either "none" (tree is fully continuous), "pass" (there is a gap at the root, i.e., this gap must be filled later further up in the tree), or "gap" (the root of this tree fills a gap, i.e., its children have gaps, but the root does not). Finally, y is the sum of all gap lengths.

3 Experiments

3.1 Data

We use the TiGer treebank release 2.2 (TIGER), and the NeGra treebank (NEGRA). For TIGER, we use the first half of the last 10,000 sentences for development and the second half for testing.[3] We also recreate the split of Hall and Nivre (2008) (TIGERHN), for which we split TiGer in 10 parts, assigning sentence i to part $i mod 10$. The first of those parts is used for testing, the concatenation of the rest for training.

[2] See Maier and Lichte (2011) for a formal account on gaps in treebanks.

[3] This split, which corresponds to the split used in the SPMRL 2013 shared task (Seddah et al., 2013), was proposed in Farkas and Schmid (2012). We exclude sentences 46,234 and 50,224, because of annotation errors. Both contain nodes with more than one parent node.

From NeGra, we exclude all sentences longer than 30 words (in order to make a comparison with `rparse` possible, see below), and split off the last 10% of the treebank for testing, as well as the previous 10% for development. As a preprocessing step, in both treebanks we remove spurious discontinuities that are caused by material which is attached to the virtual root node (mainly punctuation). All such elements are attached to the least common ancestor node of their left and right terminal neighbors (as proposed by Levy (2005), p. 163). We furthermore create a continuous variant NEGRACF of NEGRA with the method usually used for PCFG parsing: For all maximal continuous parts of a discontinuous constituent, a separate node is introduced (Boyd, 2007). Subsequently, all nodes that do not cover the head child of the discontinuous constituent are removed.

No further preprocessing or cleanup is applied.

3.2 Experimental Setup

Our parser is implemented in Java. We run all our experiments with Java 8 on an Intel Core i5, allocating 15 GB per experiment. All experiments are carried out with gold POS tags, as in previous work on shift-reduce constituency parsing (Zhang and Clark, 2009). Grammatical function labels are discarded.

For the evaluation, we use the corresponding module of *discodop*.[4] We report several metrics (as implemented in *discodop*):

- Extended labeled bracketing, in which a bracket for a single node consists of its label and a set of pairs of indices, delimiting the continuous blocks it covers. We do not include the root node in the evaluation and ignore punctuation. We report labeled precision, recall and F_1, as well as exact match (all brackets correct).
- Leaf-ancestor (Sampson and Babarczy, 2003), for which we consider all paths from leaves to the root.
- Tree edit distance (Emms, 2008), which consists of the minimum edit distance between gold tree and parser output.

Aside from a full evaluation, we also evaluate only the constituents that are discontinuous.

[4] `http://github.com/andreasvc/discodop`

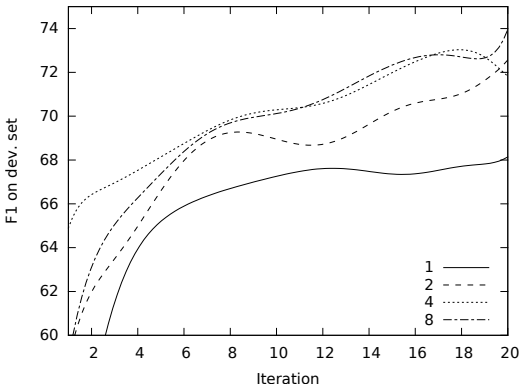

Figure 5: NEGRA dev results (F_1) for different beam sizes

We perform 20 training iterations unless indicated otherwise. When training stops, we average the model (as in Daumé III (2006)).

We run further experiments with *rparse*[5] (Kallmeyer and Maier, 2013) to facilitate a comparison with a grammar-based parser.

3.3 Results

We start with discontinuous parsing experiments on NEGRA and TIGER, followed by continuous parsing experiments, and a comparison to grammar-based parsing.

3.3.1 Discontinuous Parsing

NeGra The first goal is to determine the effect of different beam sizes with BASELINE features and the COMPOUNDSWAP$_i$ operation. We run experiments with beam sizes 1, 2, 4 and 8; Fig. 5 shows the results obtained on the dev set after each iteration. Fig. 6 shows the average decoding speed during each iteration for each beam size (both smoothed).

Tracking two items instead of one results in a large improvement. Raising the beam size from 2 to 4 results in a smaller improvement. The improvement obtained by augmenting the beam size from 4 to 8 is even smaller. This behavior is mirrored by the parsing speeds during training: The differences in parsing speed roughly align with the result differences. Note that fast parsing during training means that the parser does not perform well (yet) and that therefore, early update is done more often. Note finally that the average parsing speeds on the test set after the last training iteration

[5] `http://github.com/wmaier/rparse`

	All				Discont. only			
	LR	LP	LF_1	E	LR	LP	LF_1	E
BASELINE combined with								
SWAP	74.74	75.60	75.17	43.54	15.70	15.82	15.76	12.31
COMPOUNDSWAP$_i$	75.60	76.37	75.98	43.04	**16.46**	19.96	18.05	12.05
BASELINE + COMPOUNDSWAP$_i$ combined with								
SEPARATOR	75.20	75.74	75.47	42.61	13.11	16.73	14.70	9.89
EXTENDED	76.15	76.92	76.53	**44.46**	15.09	20.62	17.43	12.70
DISCO	75.86	76.39	76.12	43.94	15.42	22.95	18.45	12.86
IMPORTANCE	75.72	76.61	76.16	43.86	16.16	20.42	18.04	12.38
BASELINE + COMPOUNDSWAP$_i$ + DISCO combined with								
EXTENDED	76.68	**77.19**	76.93	44.10	15.27	26.88	19.47	13.61
EXTENDED + SEPARATOR	76.21	76.45	76.33	43.29	15.57	26.56	19.63	13.52
IMPORTANCE	76.22	76.86	76.54	43.75	16.01	**29.41**	**20.73**	**13.89**
EXTENDED + IMPORTANCE	**76.76**	77.13	**76.95**	44.30	15.09	28.86	19.82	13.23

Table 1: Results NEGRA, beam size 8

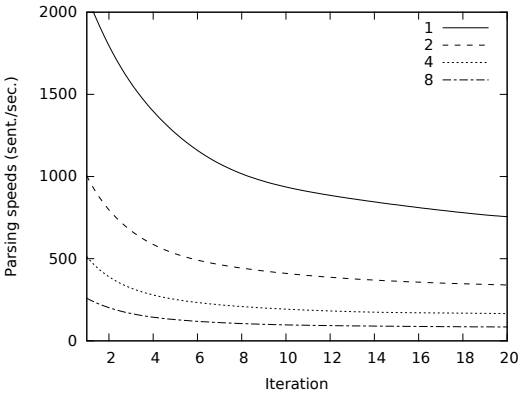

Figure 6: NEGRA dev average parsing speeds per sentence for different beam sizes

range from 640 sent./sec. (greedy) to 80 sent./sec. (beam size 8).

For further experiments on NeGra, we choose a beam size of 8. Tab. 1 shows the bracketing scores for various parser setups. In Tab. 2, the corresponding TED and Leaf-Ancestor scores are shown.

In the first block of the tables, we compare SWAP with COMPOUNDSWAP$_i$. On all

	TED	LA
BASELINE combined with		
SWAP	89.19	91.62
COMPOUNDSWAP$_i$	89.60	91.93
BASELINE + COMPOUNDSWAP$_i$ combined with		
SEPARATOR	89.41	91.77
EXTENDED	89.68	91.99
DISCO	89.42	91.83
BASELINE + COMPOUNDSWAP$_i$ + DISCO combined with		
IMPORTANCE	89.64	91.90
EXTENDED	89.68	91.99
EXTENDED + SEPARATOR	89.52	91.86
EXTENDED + IMPORTANCE	89.80	91.98

Table 2: Results NEGRA TED and Leaf-Ancestor

constituents, the latter beats the former by 0.8 (F_1). On discontinuous constituents, using COMPOUNDSWAP$_i$ gives an improvement of more than four points in precision and of about 0.8 points in recall. A manual analysis confirms that as expected, particularly discontinuous constituents with large gaps profit from bundling swap transitions.

In the second block, we run the BASELINE features with COMPOUNDSWAP$_i$ combined with SEPARATOR, EXTENDED and DISCO. The SEPARATOR features were not as successful as they were for Zhang and Clark (2009). All scores for discontinuous constituents drop (compared to the baseline). The EXTENDED features are more effective and give an improvement of about half a point F_1 on all constituents, as well as the highest exact match among all experiments. On discontinuous constituents, precision raises slightly but we loose about 1.4% in recall (compared to the baseline). The latter seems to be due to the fact that in comparison to the baseline, with EXTENDED, more sentences get erroneously analyzed as not containing any crossing branches. This effect can be explained with data sparseness and is less pronounced when more training data is available (see below). Similarly to EXTENDED, the new DISCO features lead to a slight gain over the baseline (on all constituents). As with EXTENDED, on discontinuous constituents, we again gain precision (3%) but loose recall (0.5%), because more sentences wrongly analyzed as not having discontinuities than in the BASELINE. A category-based evaluation of discontinuous constituents reveals that EXTENDED has an advantage over DISCO when considering all constituents. However, we can also see that the DISCO features yield better results than

EXTENDED particularly on the frequent discontinuous categories (NP, VP, AP, PP), which indicates that the information about gap type and gap length is useful for the recovery of discontinuities. IMPORTANCE (see Sec. 2.3) is not very successful, yielding results which lie in the vicinity of those of the BASELINE.

In the third block of the tables, we test the performance of the DISCO features in combination with other techniques, i.e., we use the BASELINE and DISCO features with COMPOUNDSWAP$_i$ and combine it with EXTENDED and SEPARATOR features as well as with the IMPORTANCE strategy. All experiments beat the BASELINE/DISCO combination in terms of F_1. EXTENDED and DISCO give a cumulative advantage, resulting in an increase of precision of almost 4%, resp. over 6% on discontinuous constituents, compared to the use of DISCO, resp. EXTENDED alone. Adding the SEPARATOR features to this combination does not bring an advantage. The IMPORTANCE strategy is the most successful one in combination with DISCO, causing a boost of almost 10% on precision of discontinuous constituents, leading to the highest overall discontinuous F_1 of 29.41 (notably more than 12 points higher than the baseline); also on all constituents we obtain the third-highest F_1. Combining DISCO with IMPORTANCE and EXTENDED leads to the highest overall F_1 on all constituents of 76.95, however, the results on discontinuous constituents are slightly lower than for IMPORTANCE alone. This confirms the previously observed behavior: The EXTENDED features help when considering all constituents, but they do not seem to be effective for the recovery of discontinuities in particular.

In the TED and LA scores (Tab. 2), we see much less variation than in the bracketing scores. As reported in the literature (e.g., Rehbein and van Genabith (2007)), this is because of the fact that with bracketing evaluation, a single wrong attachment can "break" brackets which otherwise would be counted as correct. Nevertheless, the trends from bracketing evaluation repeat.

To sum up, the COMPOUNDSWAP$_i$ operation works better than SWAP because the latter misses long gaps. The most useful feature sets were EXTENDED and DISCO, both when used independently and when used together. DISCO was particularly useful for discontinuous constituents. SEPARATOR yielded no usable improvements. IMPORTANCE has also proven to be effective, yielding the best results on discontinuous constituents (in combination with DISCO). Over almost all experiments, a common error is that on root level, CS and S get confused, indicating that the present features do not provide sufficient information for disambiguation of those categories. We can also confirm the tendency that discontinuous VPs in relatively short sentences are recognized correctly, as reported by Versley (2014).

TiGer We now repeat the most successful experiments on TIGER. Tab. 3 shows the parsing results for the test set.

Some of the trends seen on the experiments with NEGRA are repeated. EXTENDED and DISCO yields an improvement on all constituents. However, now not only DISCO, but also EXTENDED lead to improved scores on discontinuous constituents. As mentioned above, this can be explained with the fact that for the EXTENDED features to be effective, the amount of training data available in NEGRA was not enough. Other than in NEGRA, the DISCO features are now more effective when used alone, leading to the highest overall F_1 on discontinuous constituents of 19.45. They are, however, less effective in combination with EXTENDED. This is partially remedied by giving the swap transitions more IMPORTANCE, which leads to the highest overall F_1 on all constituents of 74.71.

The models we learn are sparse, therefore, as mentioned above, we can exploit the work of Goldberg and Elhadad (2011). They propose to only include the weight of a feature in the computation of a score if it has been seen more than MINUPDATE times. We repeat the BASELINE experiment with two different MINUPDATE settings (see Tab. 3). As expected, the MINUPDATE models are much smaller. The final model with the baseline experiment uses 8.3m features (parsing speed on test set 73 sent./sec.), with MINUPDATE 5 3.3m features (121 sent./sec.) and with MINUPDATE 10 1.8m features (124 sent./sec.). With MINUPDATE 10, the results do degrade. However, with MINUPDATE 5 in addition to the faster parsing we consistently improve over the baseline.

Finally, in order to check the convergence, we run a further experiment in which we limit training iterations to 40 instead of 20, together with beam size 4. We use the BASELINE features with COMPOUNDSWAP$_i$ combined with DISCO, EX-

	All				Discont. only			
	LR	LP	LF_1	E	LR	LP	LF_1	E
BASELINE + COMPOUNDSWAP$_i$	72.69	74.77	73.71	36.47	16.08	18.72	17.30	12.96
+ EXTENDED	73.52	75.50	74.50	37.26	15.86	20.04	17.71	13.20
+ DISCO	73.77	**75.35**	74.55	37.08	**16.68**	23.32	**19.45**	**14.43**
+ DISCO + EXTENDED	73.97	75.29	74.62	**37.54**	15.56	22.21	18.30	13.64
+ DISCO + EXTENDED + IMPORTANCE	**74.01**	75.41	**74.71**	37.20	15.61	**23.53**	18.77	13.84
BASELINE + COMPOUNDSWAP$_i$ with								
MINUPDATE 5	73.04	75.03	74.03	37.36	16.25	19.72	17.82	13.28
MINUPDATE 10	72.71	74.55	73.62	36.85	15.78	18.56	17.06	13.07

Table 3: Results TIGER, beam size 4

	LR	LP	LF_1	E
BASELINE	81.89	82.49	82.19	49.05
EXTENDED	**82.20**	**82.70**	**82.45**	**49.54**

Table 4: Results NEGRACF

	LR	LP	LF_1	E
All constituents	69.72	68.85	69.28	33.89
Disc. only	25.77	27.51	26.61	17.77

Table 5: Results NEGRA *rparse*

TENDED, and IMPORTANCE. The parsing speed on the test set drops to around 39 sentences per second. However, we achieve 75.10 F_1, i.e., a slight improvement over the experiments in Tab. 3 that confirms the tendencies visible in Fig. 5.

3.3.2 Continuous Parsing

We investigate the impact of the swap transitions on both speed and parsing results by running an experiment with NEGRACF using the BASELINE and EXTENDED features. The corresponding results are shown in Tab. 4.

Particularly high frequency categories (NP, VP, S) are much easier to find in the continuous case and show large improvements. This explains why without the swap transition, F_1 with BASELINE features is 6.9 points higher than the F_1 on discontinuous constituents (with COMPOUNDSWAP$_i$). With the EXTENDED features, we obtain a small improvement.

Note that with the shift-reduce approach, the difference between the computational cost of producing discontinuous constituents vs. the cost of producing continuous constituents is much lower than for a grammar-based approach. When producing continuous constituents, parsing is only 20% faster than with the swap transition, namely 97 instead of 81 sentences per second.

In order to give a different perspective on the role of discontinuous constituents, we perform two further evaluations. First, we remove the discontinuities from the output of the discontinuous baseline parser using the procedure described in Sec. 3.1 and evaluate the result against the continuous gold data. We obtain an F_1 of 76.70, 5.5 points lower than the continuous baseline. Secondly, we evaluate the output of the continuous baseline parser against the discontinuous gold data. This leads to an F_1 78.89, 2.9 point more than the discontinuous baseline. Both evaluations confirm the intuition that parsing is much easier when discontinuities (i.e., in our case the swap transition) do not have to be considered.

3.3.3 Comparison with other Parsers

rparse In order to compare our parser with a grammar-based approach, we now parse NEGRA with *rparse*, with the same training and test sets as before (i.e., we do not use the development set). We employ markovization with $v = 1$, $h = 2$ and head driven binarization with binary top and bottom productions.

The first thing to notice is that *rparse* is much slower than our parser. The average parsing speed is about 0.3 sent./sec.; very long sentences require over a minute to be parsed. The parsing results are shown in Tab. 5. They are about 5 points worse than those reported by Kallmeyer and Maier (2013). This is due to the fact that they train on the first 90% of the treebank, and not on the first 80% as we do, which leads to an increased number of unparsed sentences. In comparison to the baseline setting of the shift-reduce parser with beam size 8, the results are around 10 points worse. However, *rparse* reaches an F_1 of 26.61 on discontinuous constituents, which is 5.9 points more than we achieved with the best setting with our parser.

In order to investigate why the grammar-based approach outperforms our parser on discontinuous constituents, we count the frequency of LCFRS productions of a certain gap degree in the binarized grammar used in the *rparse* experiment. The

	LF_1	E
Versley (2014)	74.23	37.32
this work	79.52	44.32
H&N (2008)	79.93	37.78
F&M (2015)	**85.53**	**51.21**

Table 6: Results TIGERHN, sentence length ≤ 40

average occurrence count of rules with gap degree 0 is 12.18. Discontinuous rules have a much lower frequency, the average count of productions with one, two and three gaps being 3.09, 2.09, and 1.06, respectively. In PCFG parsing, excluding low frequency productions does not have a large effect (Charniak, 1996); however, this does not hold for LCFRS parsing, where they have a major influence (cf. Maier (2013, p. 205)): This means that removing low frequency productions has a negative impact on the parser performance particularly concerning discontinuous structures; however, it also means that low frequency discontinuous productions get triggered reliably. This hypothesis is confirmed by the fact that the our parser performs much worse on discontinuous constituents with a very low frequency (such as CS, making up only 0.62% of all discontinuous constituents) than it performs on those with a high frequency (such as VP, making up 60.65% of all discontinuous constituents), while *rparse* performs well on the low frequency constituents.

EaFi and Dependency Parsers We run an experiment with 40 iterations on TIGERHN, using DISCO, EXTENDED and IMPORTANCE. Tab. 6 lists the results, together with the corresponding results of Versley (2014), Hall and Nivre (2008) (H&N) and Fernández-González and Martins (2015) (F&M).

Our results exceed those of EaFi[6] and the exact match score of H&N. We are outperformed by the F&M parser. Note, that particularly the comparison to *EaFi* must be handled with care, since Versley (2014) uses additional preprocessing: PP-internal NPs are annotated explicitly, and the parenthetical sentences are changed to be embedded by their enclosing sentence (instead of vice versa).

We postpone a thorough comparison with both EaFi and the dependency parsers to future work.

[6]Note that Versley (2014) reports a parsing speed of 40-55 sent./sec.; depending on the beam size and the training set size, per second, our parser parses 39-640 sentences.

3.4 Discussion

To our knowledge, surprisingly, numerical scores for discontinuous constituents have not been reported anywhere in previous work. The relatively low overall performance with both grammar-based and shift-reduce based parsing, along with the fact that the grammar-based approach outperforms the shift-reduce approach, is striking. We have shown that it is possible to push the precision on discontinuous constituents, but not the recall, to the level of what can be achieved with a grammar-based approach.

Particularly the outcome of the experiments involving the EXTENDED features and IMPORTANCE drives us to the conclusion that the major problem when parsing discontinuous constituents is data sparseness. More features cannot be the only solution: A more reliable recognition of discontinuous constituents requires a more robust learning from larger amounts of data.

4 Conclusion

We have presented a shift-reduce parser for discontinuous constituents which combines previous work in shift-reduce parsing for continuous constituents with recent work in easy-first parsing of discontinuous constituents. Our experiments confirm that an incremental shift-reduce architecture with a swap transition can indeed be used to parse discontinuous constituents. The swap transition is associated with a low computational cost. We have obtained a speed-up of up to 2,000% in comparison to the grammar-based *rparse*, and we have shown that we obtain better results than with the grammar-based parser, even though the grammar-based strategy does better at the reconstruction of discontinuous constituents.

In future work, we will concentrate on methods that could remedy the data sparseness concerning discontinuous constituents, such as self-training. Furthermore, we will experiment with larger feature sets that add lexical information. An formal investigation of the expressivity of our parsing model is currently under way.

Acknowledgments

I wish to thank Miriam Kaeshammer for enlightening discussions and the three anonymous reviewers for helpful comments and suggestions. This work was partially funded by Deutsche Forschungsgemeinschaft (DFG).

References

Krasimir Angelov and Peter Ljunglöf. 2014. Fast statistical parsing with parallel multiple context-free grammars. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 368–376, Gothenburg, Sweden.

John Bauer. 2014. Stanford shift-reduce parser. `http://nlp.stanford.edu/software/srparser.shtml`.

Adriane Boyd. 2007. Discontinuity revisited: An improved conversion to context-free representations. In *Proceedings of The Linguistic Annotation Workshop (LAW) at ACL 2007*, pages 41–44, Prague, Czech Republic.

Shu Cai, David Chiang, and Yoav Goldberg. 2011. Language-independent parsing with empty elements. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 212–216, Portland, OR.

Eugene Charniak. 1996. Tree-bank grammars. Technical Report CS-96-02, Department of Computer Science, Brown University, Providence, RI.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 111–118, Barcelona, Spain.

Hal Daumé III. 2006. *Practical Structured Learning Techniques for Natural Language Processing*. Ph.D. thesis, University of Southern California, Los Angeles, CA.

Péter Dienes and Amit Dubey. 2003. Antecedent recovery: Experiments with a trace tagger. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, pages 33–40, Sapporo, Japan.

Martin Emms. 2008. Tree distance and some other variants of Evalb. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, pages 1373–1379, Marrakech, Morocco.

Kilian Evang and Laura Kallmeyer. 2011. PLCFRS parsing of English discontinuous constituents. In *Proceedings of the 12th International Conference on Parsing Technologies (IWPT 2011)*, pages 104–116, Dublin, Ireland.

Richard Farkas and Helmut Schmid. 2012. Forest reranking through subtree ranking. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1038–1047, Jeju Island, Korea.

Daniel Fernández-González and André F. T. Martins. 2015. Parsing as reduction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and Teh 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing*, Beijing, China. To appear.

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750, Los Angeles, CA.

Yoav Goldberg and Michael Elhadad. 2011. Learning sparser perceptron models. Technical report, Ben Gurion University of the Negev.

Johan Hall and Joakim Nivre. 2008. Parsing discontinuous phrase structure with grammatical functions. In Bengt Nordström and Aarne Ranta, editors, *Advances in Natural Language Processing*, volume 5221 of *Lecture Notes in Computer Science*, pages 169–180. Springer, Gothenburg, Sweden.

Valentin Jijkoun. 2003. Finding non-local dependencies: Beyond pattern matching. In *The Companion Volume to the Proceedings of 41st Annual Meeting of the Association for Computational Linguistics*, pages 37–43, Sapporo, Japan.

Mark Johnson. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 136–143, Philadelphia, PA.

Laura Kallmeyer and Wolfgang Maier. 2013. Data-driven parsing using probabilistic linear context-free rewriting systems. *Computational Linguistics*, 39(1):87–119.

Roger Levy and Christopher Manning. 2004. Deep dependencies from context-free statistical parsers: Correcting the surface dependency approximation. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 327–334, Barcelona, Spain.

Roger Levy. 2005. *Probabilistic Models of Word Order and Syntactic Discontinuity*. Ph.D. thesis, Stanford University.

Wolfgang Maier and Timm Lichte. 2011. Characterizing discontinuity in constituent treebanks. In *Formal Grammar. 14th International Conference, FG 2009. Bordeaux, France, July 25-26, 2009. Revised Selected Papers*, volume 5591 of *LNCS/LNAI*, pages 167–182. Springer-Verlag.

Wolfgang Maier and Anders Søgaard. 2008. Treebanks and mild context-sensitivity. In Philippe de Groote, editor, *Proceedings of the 13th Conference on Formal Grammar (FG-2008)*, pages 61–76, Hamburg, Germany. CSLI Publications.

Wolfgang Maier, Miriam Kaeshammer, and Laura Kallmeyer. 2012. Data-driven PLCFRS parsing revisited: Restricting the fan-out to two. In *Proceedings of the Eleventh International Conference on Tree Adjoining Grammars and Related Formalisms (TAG+11)*, pages 126–134, Paris, France.

Wolfgang Maier. 2013. *Parsing Discontinuous Structures*. Dissertation, University of Tübingen.

Joakim Nivre, Marco Kuhlmann, and Johan Hall. 2009. An improved oracle for dependency parsing with online reordering. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 73–76, Paris, France.

Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359, Singapore.

Ines Rehbein and Josef van Genabith. 2007. Evaluating evaluation measures. In *Proceedings of the 16th Nordic Conference of Computational Linguistics NODALIDA-2007*, pages 372–379, Tartu, Estonia.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 125–132, Vancouver, BC.

Geoffrey Sampson and Anna Babarczy. 2003. A test of the leaf-ancestor metric for parse accuracy. *Journal of Natural Language Engineering*, 9:365–380.

Helmut Schmid. 2006. Trace prediction and recovery with unlexicalized PCFGs and slash features. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 177–184, Sydney, Australia.

Djamé Seddah, Reut Tsarfaty, Sandra Kübler, Marie Candito, Jinho D. Choi, Richárd Farkas, Jennifer Foster, Iakes Goenaga, Koldo Gojenola Galletebeitia, Yoav Goldberg, Spence Green, Nizar Habash, Marco Kuhlmann, Wolfgang Maier, Yuval Marton, Joakim Nivre, Adam Przepiórkowski, Ryan Roth, Wolfgang Seeker, Yannick Versley, Veronika Vincze, Marcin Woliński, and Alina Wróblewska. 2013. Overview of the SPMRL 2013 shared task: A cross-framework evaluation of parsing morphologically rich languages. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 146–182, Seattle, WA.

Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On Multiple Context-Free Grammars. *Theoretical Computer Science*, 88(2):191–229.

Andreas van Cranenburgh and Rens Bod. 2013. Discontinuous parsing with an efficient and accurate DOP model. In *Proceedings of The 13th International Conference on Parsing Technologies*, Nara, Japan.

Andreas van Cranenburgh. 2012. Efficient parsing with linear context-free rewriting systems. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 460–470, Avignon, France.

Yannick Versley. 2014. Experiments with easy-first nonprojective constituent parsing. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 39–53, Dublin, Ireland.

K. Vijay-Shanker, David Weir, and Aravind K. Joshi. 1987. Characterising structural descriptions used by various formalisms. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, pages 104–111, Stanford, CA.

Yue Zhang and Stephen Clark. 2009. Transition-based parsing of the Chinese treebank using a global discriminative model. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 162–171, Paris, France.

Yue Zhang and Stephen Clark. 2011a. Shift-reduce CCG parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 683–692, Portland, OR.

Yue Zhang and Stephen Clark. 2011b. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, 37(1):105–151.

Muhua Zhu, Jingbo Zhu, and Huizhen Wang. 2012. Exploiting lexical dependencies from large-scale data for better shift-reduce constituency parsing. In *Proceedings of COLING 2012*, pages 3171–3186, Mumbai, India.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 434–443, Sofia, Bulgaria.