Incremental Discontinuous Phrase Structure Parsing with the GAP Transition

Maximin Coavoux[1,2] and **Benoît Crabbé**[1,2,3]
[1]Univ. Paris Diderot, Sorbonne Paris Cité
[2]Laboratoire de linguistique formelle (LLF, CNRS)
[3]Institut Universitaire de France
maximin.coavoux@etu.univ-paris-diderot.fr
benoit.crabbe@linguist.univ-paris-diderot.fr

Abstract

This article introduces a novel transition system for discontinuous lexicalized constituent parsing called SR-GAP. It is an extension of the shift-reduce algorithm with an additional gap transition. Evaluation on two German treebanks shows that SR-GAP outperforms the previous best transition-based discontinuous parser (Maier, 2015) by a large margin (it is notably twice as accurate on the prediction of discontinuous constituents), and is competitive with the state of the art (Fernández-González and Martins, 2015). As a side contribution, we adapt span features (Hall et al., 2014) to discontinuous parsing.

1 Introduction

Discontinuous constituent trees can be used to model directly certain specific linguistic phenomena, such as extraposition, or more broadly to describe languages with some degree of word-order freedom. Although these phenomena are sometimes annotated with indexed traces in CFG treebanks, other constituent treebanks are natively annotated with discontinuous constituents, e.g. the Tiger corpus (Brants, 1998).

From a parsing point of view, discontinuities pose a challenge. Mildly context-sensitive formalisms, that are expressive enough to model discontinuities have high parsing complexity. For example, the CKY algorithm for a binary probabilistic LCFRS is in $\mathcal{O}(n^{3k})$, where k is the fan-out of the grammar (Kallmeyer, 2010).

Recently, there have been several proposals for direct discontinuous parsing. They correspond roughly to three different parsing paradigms. (i) Chart parsers are based on probabilistic LCFRS (Kallmeyer and Maier, 2013; Maier, 2010; Evang and Kallmeyer, 2011), or on the Data-Oriented Parsing (DOP) framework (van Cranenburgh, 2012; van Cranenburgh and Bod, 2013; van Cranenburgh et al., 2016). However, the complexity of inference in this paradigm requires to design elaborate search strategies and heuristics to make parsing run-times reasonable. (ii) Several approaches are based on modified non-projective dependency parsers, for example Hall and Nivre (2008), or more recently Fernández-González and Martins (2015) who provided a surprisingly accurate parsing method that can profit from efficient dependency parsers with rich features. (iii) Transition-based discontinuous parsers are based on the easy-first framework (Versley, 2014a) or the shift-reduce algorithm augmented with a swap action (Maier, 2015). In the latter system, which we will refer to as SR-SWAP, a SWAP action pushes the second element of the stack back onto the buffer.

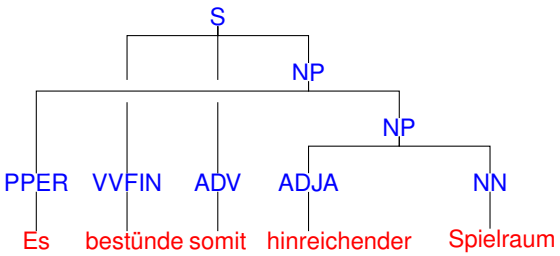

Figure 1: Discontinuous tree extracted from the Tiger corpus (punctuation removed).

Although SR-SWAP is fast and obtained good results, it underperforms Fernández-González and Martins (2015)'s parser by a large margin. We believe this result does not indicate a fatal problem for the transition-based framework for discontinuous parsing, but emphasizes several limitations inherent to SR-SWAP, in particular the length of derivations (Section 3.5).

Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers, pages 1259–1270,
Valencia, Spain, April 3-7, 2017.

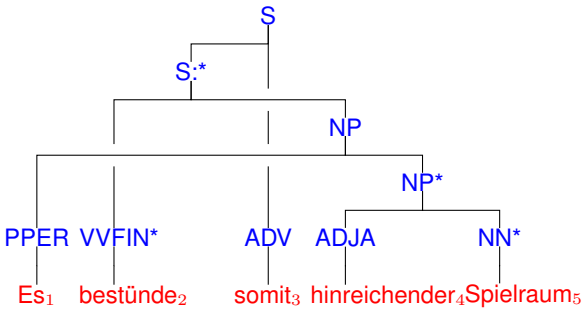

Figure 2: Lexicalized binarized tree. The symbol '*' encodes head information. Symbols suffixed by ':' are temporary symbols introduced by the binarization.

Contributions We introduce a novel transition system for discontinuous parsing we call SR-GAP. We evaluate this algorithm on two German treebanks annotated with discontinuous constituents, and show that, in the same experimental settings, it outperforms the previous best transition-based parser of Maier (2015), and matches the best published results on these datasets (Fernández-González and Martins, 2015). We provide a theoretical and empirical comparison between SR-GAP and SR-SWAP. Finally we adapt span-based features to discontinuous parsing.

The code and preprocessing scripts to reproduce experiments are available for download at `https://github.com/mcoavoux/mtg`.

2 Discontinuous Shift-Reduce Parsing

In this section we present SR-GAP, a transition system for discontinuous constituent parsing. SR-GAP is an extension of the shift-reduce system. In what follows, we assume that part-of-speech tags for the words of a sentence are given. A *terminal* refers to a tag-word couple.

2.1 Standard Shift-Reduce Constituent Parsing

The shift-reduce system is based on two data structures. The stack (S) contains tree fragments representing partial hypotheses and the buffer (B) contains the remaining terminals. A parsing configuration is a couple $\langle S, B\rangle$. Initially, B contains the sentence as a list of terminals and S is empty.

The three types of actions are defined as follows.

- SHIFT$(\langle S, w|B\rangle) = \langle S|w, B\rangle$
- REDUCE-X$(\langle S|A_1|A_2, B\rangle) = \langle S|X, B\rangle$
- REDUCEUNARY-X$(\langle S|A, B\rangle) = \langle S|X, B\rangle$

where X is any non-terminal in the grammar. The analysis terminates when the buffer is empty and the only symbol in the stack is an axiom of the grammar. This transition system can predict any labelled constituent tree over a set of non-terminal symbols N.

These three action types can only produce binary trees. In practice, shift-reduce parsers often assume that their data are binary. In this article, we assume that trees are binary, that each node X is annotated with a head h (notation: X[h]), and that the only unary nodes are parents to a terminal. Therefore, we only need unary reductions immediately after a SHIFT. We refer the reader to Section 3.1 for the description of the preprocessing operations.

2.2 SR-GAP Transition System

In order to handle discontinuous constituents, we need an algorithm expressive enough to predict non-projective trees.

Compared to the standard shift-reduce algorithm, the main intuition behind SR-GAP is that reductions do not always apply to the two top-most elements in the stack. Instead, the left element of a reduction can be any element in the stack and must be chosen dynamically.

To control the choice of the symbols to which a reduction applies, the usual stack is split into two data structures. A deque D represents its top and a stack S represents its bottom. Alternatively, we could see these two data structures as a single stack with two pointers indicating its top and a split point. The respective top-most element of D and S are those available for a reduction.

The transition system is given as a deductive system in Figure 3. A REDUCE-X action pops the top element of S and the top element of D, flushes the content of D to S and finally pushes a new non-terminal X on D. As feature extraction (Section 3.1) relies on the lexical elements, we use two types of binary reductions, left and right, to assign heads to new constituents. Unary reductions replace the top of D by a new non-terminal.

The SHIFT action flushes D to S, pops the next token from B and pushes it onto D.

Finally, the GAP action pops the first element of S and appends it at the bottom of D. This action enables elements below the top of S to be also available for a reduction with the top of D.

Input $t_1[w_1]t_2[w_2]\dots t_n[w_n]$

Axiom $\langle \epsilon, \epsilon, t_1[w_1]t_2[w_2]\dots t_n[w_n]\rangle$

Goal $\langle \epsilon, S[w], \epsilon\rangle$

SH $$\frac{\langle S, D, t[w]|B\rangle}{\langle S|D, t[w], B\rangle}$$

RU(X) $$\frac{\langle S, d_0[h], B\rangle}{\langle S, X[h], B\rangle}$$

RR(X) $$\frac{\langle S|s_0[h], D|d_0[h'], B\rangle}{\langle S|D, X[h'], B\rangle}$$

RL(X) $$\frac{\langle S|s_0[h], D|d_0[h'], B\rangle}{\langle S|D, X[h], B\rangle}$$

GAP $$\frac{\langle S|s_0[h], D, B\rangle}{\langle S, s_0[h]|D, B\rangle}$$

Figure 3: SR-GAP transition system for discontinuous phrase structure parsing. $X[h]$ denotes a non-terminal X and its head h. s_0 and d_0 denote the top-most elements of respectively S and D.

Constraints In principle, a tree can be derived by several distinct sequences of actions. If a SHIFT follows a sequence of GAPS, the GAPS will have no effect, because SHIFT flushes D to S before pushing a new terminal to D. In order to avoid useless GAPS, we do not allow a SHIFT to follow a GAP. A GAP must be followed by either another GAP or a binary reduction.

Moreover, as we assume that preprocessed trees do not contain unary nodes, except possibly above the terminal level, unary reductions are only allowed immediately after a SHIFT. Other constraints on the transition system are straightforward, we refer the reader to Table 7 of Appendix A for the complete list.

2.3 Oracle and Properties

Preliminary Definitions Following Maier and Lichte (2016), we define a discontinuous tree as a rooted connected directed acyclic graph $T = (V, E, r)$ where

- V is a set of nodes;
- $r \in V$ is the root node;
- $E : V \times V$ is a set of (directed) edges and E^* is the reflexive transitive closure of E.

If $(u, v) \in E$, then u is the parent of v. Each node has a unique parent (except the root that has none). Nodes without children are *terminals*.

The *right index* (resp. *left index*) of a node is the index of the rightmost (resp. leftmost) terminal dominated by this node. For example, the left index of the node labelled S: in Figure 2 is 1 and its right index is 5.

Oracle We extract derivations from trees by following a simple tree traversal. We start with an initial configuration. While the configuration is not final, we derive a new configuration by performing the gold action, which is chosen as follows:

- if the nodes at the top of S and at the top of D have the same parent node in the gold tree, perform a reduction with the parent node label;
- if the node at the top of D and the i^{th} node in S have the same parent node, perform $i - 1$ GAP;
- otherwise, perform a SHIFT, optionally followed by a unary reduction in the case where the parent node of the top of D has only one child.

For instance, the gold sequence of actions for the tree in Figure 2 is the sequence SH, SH, SH, SH, SH, RR(NP), GAP, GAP, RR(NP), GAP, RL(S:), RR(S). Table 1 details the sequence of configurations obtained when deriving this tree.

Given the constraints defined in Section 2.2, and if we ignore lexicalisation, there is a bijection between binary trees and derivations.[1] To see why, we define a total order $<$ on the nodes of a tree. Let n and n' be two nodes and let $n < n'$ iff either $rindex(n) < rindex(n')$ or $(n', n) \in E^*$.

It is immediate that if $(n', n) \in E^*$, then n must be reduced before n' in a derivation. An invariant of the GAP transition system is that the right index of the first element of D is equal to the index of the last shifted element. Therefore, after having shifted the terminal j, it is impossible to create nodes whose right-index is strictly smaller to j. We conclude that during a derivation, the nodes must be created according to the strict total order $<$ defined above. In other words, for a given tree, there is a unique possible derivation which enforces the constraints described above. Recip-

[1] But several binarized trees can correspond to the same n-ary tree.

S	D	B	Action
		Es bestünde somit hinreichender Spielraum	
	Es	bestünde somit hinreichender Spielraum	SH
Es	bestünde	somit hinreichender Spielraum	SH
Es bestünde	somit	hinreichender Spielraum	SH
Es bestünde somit	hinreichender	Spielraum	SH
Es bestünde somit hinreichender	Spielraum		SH
Es bestünde somit	NP[Spielraum]		RR(NP)
Es bestünde	somit NP[Spielraum]		GAP
Es	bestünde somit NP[Spielraum]		GAP
bestünde somit	NP[Spielraum]		RR(NP)
bestünde	somit NP[Spielraum]		GAP
somit	S:[bestünde]		RL(S:)
	S[bestünde]		RR(S)

Table 1: Example derivation for the sentence in Figure 2, part-of-speech tags are omitted.

rocally, a well-formed derivation corresponds to a unique tree.

Completeness and Soundness The GAP transition system is sound and complete for the set of discontinuous binary trees labelled with a set of non-terminal symbols. When augmented with certain constraints to make sure that predicted trees are unbinarizable (see Table 7 of Appendix A), this result also holds for the set of discontinuous n-ary trees (modulo binarization and unbinarization).

Completeness follows immediately from the correctness of the oracle, which corresponds to a tree traversal in the order specified by $<$.

To prove soundness, we need to show that any valid derivation sequence produces a discontinuous binary tree. It holds from the transition system that no node can have several parents, as parent assignation via REDUCE actions pops the children nodes and makes them unavailable to other reductions. This implies that at any moment, the content of the stack is a forest of discontinuous trees. Moreover, at each step, at least one action is possible (thanks to the constraints on actions). As there can be no cycles, the number of actions in a derivation is upper-bounded by $\frac{1}{2}(n^2+n)$ for a sentence of length n (see Appendix A.1). Therefore, the algorithm can always reach a final configuration, where the forest only contains one discontinuous tree.

The correctness of SR-GAP system holds only for the robust case: that is the full set of labeled discontinuous trees, and not, say, for the set of trees derived by a true LCFRS grammar also able to reject agrammatical sentences. From an empirical point of view, a transition system that overgenerates is necessary for robustness, and is desirable for fast approximate linear-time inference. However, from a formal point of view, the relationship of the SR-GAP transition system with automata explicitly designed for LCFRS parsing (Villemonte de La Clergerie, 2002; Kallmeyer and Maier, 2015) requires further investigations.

2.4 Length of Derivations

Any derivation produced by SR-GAP for a sentence of length n will contain exactly n SHIFTS and $n-1$ binary reductions. In contrast, the number of unary reductions and GAP actions can vary. Therefore several possible derivations for the same sentence may have different lengths.

This is a recurring problem for transition-based parsing because it undermines the comparability of derivation scores. In particular, Crabbé (2014) observed that the score of a parse item is approximatively linear in the number of previous transitions, which creates a bias towards long derivations.

Different strategies have been proposed to ensure that all derivations have the same length (Zhu et al., 2013; Crabbé, 2014; Mi and Huang, 2015). Following Zhu et al. (2013), we use an additional IDLE action that can only be performed when a parsing item is final. Thus, short derivations are padded until the last parse item in the beam is final. IDLE actions are scored exactly like any other action.

SR-CGAP As an alternative strategy to the problem of comparability of hypotheses, we also present a variant of SR-GAP, called SR-CGAP, in which the length of any derivation only depends on the length of the sentence. In SR-CGAP, each

SHIFT action must be followed by either a unary reduction or a ghost reduction (Crabbé, 2015), and each binary reduction must be preceded by exactly one compound GAP_i action ($i \in \{0, \dots m\}$) specifying the number i of consecutive standard GAPS. For example, GAP_0 will have no effect, and GAP_2 counts as a single action equivalent to two consecutive GAPS. We call these actions COMPOUNDGAP, following the COMPOUNDSWAP actions of Maier (2015).

With this set of actions, any derivation will have exactly $4n - 2$ actions, consisting of n shifts, n unary reductions or ghost reductions, $n - 1$ compound gaps, and $n - 1$ reductions.

The parameter m (maximum index of a compound gap) is determined by the maximum number of consecutive gaps observed in the training set. Contrary to SR-GAP, SR-CGAP is not complete, as some discontinuous trees whose derivation should contain more than m consecutive GAPS cannot be predicted.

2.5 Beam Search with a Tree-structured Stack

A naive beam implementation of SR-GAP will copy the whole parsing configuration at each step and for each item in the beam. This causes the parser algorithm to have a practical $\mathcal{O}(k \cdot n^2)$ complexity, where k is the size of the beam and n the length of a derivation. To overcome this, one can use a tree-structured stack (TSS) to factorize the representations of common prefixes in the stack as described by (Goldberg et al., 2013) for projective dependency parsing. However the discontinuites entail that a limited amount of copying cannot be entirely avoided. When a reduction follows n GAP actions, we need to grow a new branch of size $n + 1$ in the tree-structured stack to account for reordering. The complexity of the inference becomes $\mathcal{O}(k \cdot (n + g))$ where g is the number of gaps in the derivation. As there are very few gap actions (in proportion) in the dataset, the practical runtime is linear in the length of the derivation.

2.6 Relationship to Dependency Parsing Algorithms

The transition system presented in this article uses two distinct data structures to represent the stack. In this respect, it belongs to the family of algorithms presented by Covington (2001) for dependency parsing. Covington's algorithm iterates over every possible pair of words in a sentence and decides for each pair whether to attach them – with a left or right arc – or not. This algorithm can be formulated as a transition system with a split stack (Gómez-Rodríguez and Fernández-González, 2015).

3 Experiments

3.1 Datasets

We evaluated our model on two corpora, namely the Negra corpus (Skut et al., 1997) and the Tiger corpus (Brants, 1998). To ensure comparability with previous work, we carried out experiments on several instantiations of these corpora.

We present results on two instantiations of Negra. NEGRA-30 consists of sentences whose length is smaller than, or equal to, 30 words. We used the same split as Maier (2015). A second instantiation, NEGRA-ALL, contains all the sentences of the corpus, and uses the standard split (Dubey and Keller, 2003).

For the Tiger corpus, we also use two instantiations. TIGERHN08 is the split used by Hall and Nivre (2008). TIGERM15 is the split of Maier (2015), which corresponds to the SPMRL split (Seddah et al., 2013).[2] We refer the reader to Table 8 in Appendix A for further details on the splits used.

For both corpora, the first step of preprocessing consists in removing function labels and reattaching the nodes attached to the ROOT and causing artificial discontinuities (these are mainly punctuation terminals).[3]

Then, corpora are head-annotated using the headrules included in the DISCO-DOP package, and binarized by an order-0 head-Markovization (Klein and Manning, 2003). There is a rich literature on binarizing LCFRS (Gómez-Rodríguez et al., 2009; Gildea, 2010), because both the gap-degree and the rank of the resulting trees need to be minimized in order to achieve a reasonable complexity when using chart-based parsers (Kallmeyer and Maier, 2013). However, this seems not to be a problem for transition-based parsing, and the gains of using optimized binarization algorithms do not seem to be worth the

[2] As in previous work (Maier, 2015), two sentences (number 46234 and 50224) are excluded from the test set because they contain annotation errors.

[3] We used extensively the publicly available software TREETOOLS and DISCO-DOP for these preprocessing steps. The preprocessing scripts will be released with the parser source code for full replicability.

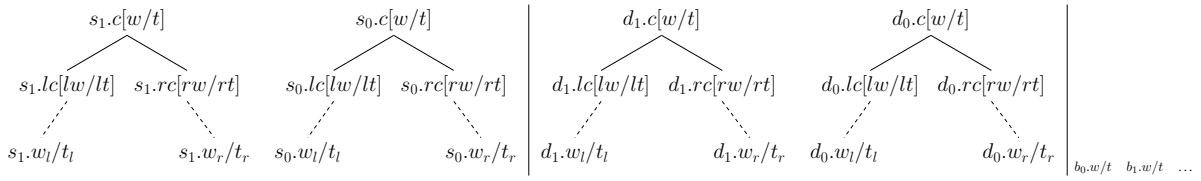

Figure 4: Schematic representation of the top-most elements of S, D and B, using the notations introduced in Table 2. Due to discontinuities, it is possible that both the left- and right- index of s_i are generated by the same child of s_i.

BASELINE				
b_0tw	b_1tw	b_2tw	b_3tw	d_0tc
d_0wc	s_0tc	s_0wc	s_1tc	s_1wc
s_2tc	s_2wc	s_0lwlc	s_0rwrc	d_0lwlc
d_0rwrc	s_0wd_0w	s_0wd_0c	s_0cd_0w	s_0cd_0c
b_0wd_0w	b_0td_0w	b_0wd_0c	b_0td_0c	b_0ws_0w
b_0ts_0w	b_0ws_0c	b_0ts_0c	b_0wb_1w	b_0wb_1t
b_0tb_1w	b_0tb_1t	$s_0cs_1wd_0c$	$s_0cs_1cd_0c$	$b_0ws_0cd_0c$
$b_0ts_0cd_0c$	$b_0ws_0wd_0c$	$b_0ts_0wd_0c$	$s_0cs_1cd_0w$	$b_0ts_0cd_0w$
+ EXTENDED				
s_3tc	s_3wc	s_1lwlc	s_1rwrc	d_1tc
d_1wc	d_2tc	d_2wc	$s_0cs_1cd_0c$	$s_2cs_0cs_1cd_0c$
$s_0cd_1cd_0c$	$s_0cd_1cs_1cd_0c$			
+ SPANS				
$d_0cw_lw_r$	$s_0cw_lw_r$	$d_0cw_ls_0w_r$	$d_0cw_rs_0w_l$	$d_0w_lw_rb_0w$
$d_0w_lw_rb_1w$	$d_0cw_rs_0w_{lo}$	$d_0ct_lw_r$	$d_0cw_lt_r$	$d_0ct_lt_r$
$s_0ct_lw_r$	$s_0cw_lt_r$	$s_0ct_lt_r$	$d_0ct_ls_0w_r$	$d_0cw_ls_0t_r$
$d_0ct_ls_0t_r$	$d_0ct_rs_0w_l$	$d_0cw_rs_0t_l$	$d_0ct_rs_0t_l$	$d_0w_lw_rb_0t$
$d_0w_lw_rb_1t$	d_0cw_{lo}	d_0ct_{lo}	s_0cw_{ro}	s_0ct_{ro}

Table 2: Feature templates. s, d and b refer respectively to the data structures (S, D, B) presented in Section 2.2. The integers are indices on these data structures. $l$eft and $r$ight refer to the children of nodes. We use c, w and t to denote a node's label, its head and the part-of-speech tag of its head. When used as a subscript, l (r) refers to the left (right) index of a node. Finally lo (ro) denotes the token immediately left to the left index (right to the right index). See Figure 4 for a representation of a configuration with these notations.

complexity of these algorithms (van Cranenburgh et al., 2016).

Unless otherwise indicated, we did experiments with gold part-of-speech tags, following a common practice for discontinuous parsing.

3.2 Classifier

We used an averaged structured perceptron (Collins, 2002) with early-update training (Collins and Roark, 2004). We use the hash trick (Shi et al., 2009) to speed up feature hashing. This has no noticeable effect on accuracy and this improves training and parsing speed. The only hyperparameter of the perceptron is the number of training epochs. We fixed it at 30 for every experiment, and shuffled the training set before each epoch.

Features We tested three feature sets described in Table 2 and Figure 4. The BASELINE feature set is the transposition of Maier (2015)'s baseline features to the GAP transition system. It is based on B, on S, and on the top element of D, but does not use information from the rest of D (i.e. the gapped elements). This feature set was designed in order to obtain an experimental setting as close as possible to that of Maier (2015).

In constrast, the EXTENDED feature set includes information from further in D, as well as additional context in S and n-grams of categories from both S and D.

The third feature set SPANS is based on the idea that constituent boundaries contain critical information (Hall et al., 2014) for phrase structure parsing. This intuition is also confirmed in the context of lexicalized transition-based constituent parsing (Crabbé, 2015). To adapt this type of features to discontinuous parsing, we only rely on the right and left index of nodes, and on the tokens preceding the left index or following the right index.[4]

Unknown Words In order to learn parameters for unknown words and limit feature sparsity, we replace hapaxes in the training set by an UNKNOWN pseudo-word. This accounts for an improvement of around 0.5 F1.

3.3 Results

We report results on test sets in Table 3. All the metrics were computed by DISCO-DOP with the parameters included in this package (`proper.prm`). The metrics are labelled F1, ignoring roots and punctuation. We also present metrics which consider only the discontinuous constituents (Disc. F1 in Tables 3 and 4), as these

[4]For discontinuous constituents, internal boundaries (for gaps) might prove useful too.

	Method	NEGRA-30 All	NEGRA-ALL $L \leq 40$	NEGRA-ALL All	TIGERHN08 $L \leq 40$	TIGERHN08 All	TIGERM15 SPMRL / standard
Fernández-González and Martins (2015)	dep2const	82.56†	81.08	80.52	**85.53**	**84.22**	80.62 / -
Hall and Nivre (2008)	dep2const	-	-	-	79.93	-	-/-
van Cranenburgh (2012)	DOP	-	72.33	71.08	-	-	-/-
van Cranenburgh and Bod (2013)	DOP	-	76.8	-	-	-	-/-
Kallmeyer and Maier (2013)	LCFRS	75.75	-	-	-	-	-/-
Versley (2014a)	EAFI	-	-	-	74.23	-	-/-
Maier (2015) (baseline, b=(Ne=8/Ti=4))	SR-SWAP	75.17 (15.76)	-	-	-	-	-/-
Maier (2015) (best, b=(Ne=8/Ti=4))	SR-SWAP	76.95 (19.82)	-	-	79.52	-	- / 74.71 (18.77)
Maier and Lichte (2016) (best, b=4)	SR-SWAP				80.02		-/ 76.46 (16.31)
This work, beam=4		F1 (Disc. F1)					
GAP, BASELINE	SR-GAP	79.31 (38.66)	79.29 (39.78)	78.53 (38.64)	82.84 (47.13)	81.67 (44.83)	78.77 / 78.86 (41.36)
GAP, +EXTENDED	SR-GAP	80.44 (41.13)	80.34 (43.42)	79.79 (43.56)	83.57 (50.91)	82.43 (48.81)	79.42 / 79.51 (43.76)
GAP, +SPANS	SR-GAP	81.64 (42.94)	81.70 (47.17)	81.28 (46.85)	84.40 (51.98)	83.16 (49.76)	80.30 / 80.40 (46.50)
CGAP, BASELINE	SR-CGAP	79.61 (41.06)	79.32 (43.49)	78.64 (42.13)	82.90 (47.86)	81.68 (45.55)	78.32 / 78.41 (39.99)
CGAP, +EXTENDED	SR-CGAP	80.26 (40.52)	80.48 (43.42)	79.98 (42.60)	83.23 (50.57)	82.00 (48.28)	79.32 / 79.42 (44.66)
CGAP, +SPANS	SR-CGAP	81.16 (42.39)	81.41 (44.73)	80.89 (44.13)	83.92 (50.83)	82.79 (48.84)	80.38 / 80.48 (46.17)
This work, beam=32		F1 (Disc. F1)					
GAP, BASELINE	SR-GAP	80.57 (42.16)	80.20 (43.87)	79.75 (42.80)	83.53 (51.91)	82.41 (49.63)	79.60 / 79.69 (44.77)
GAP, +EXTENDED	SR-GAP	81.61 (45.75)	81.13 (47.52)	80.54 (46.89)	84.33 (53.84)	83.17 (51.88)	80.50 / 80.59 (46.45)
GAP, +SPANS	SR-GAP	**82.46** (**47.35**)	**82.76** (**51.82**)	**82.16** (**50.00**)	85.11 (**55.99**)	84.01 (**54.26**)	**81.50** / **81.60** (**49.17**)

Table 3: Final test results. For TIGERM15, we report metrics computed with the SPMRL shared task parameters (see Section 3.3), as well as the standard parameters. †Trained on NEGRA-ALL.

can give some qualitative insight into the strengths and weaknesses of our model.

For experiments on TIGERM15, we additionally report evaluation scores computed with the SPMRL shared task parameters[5] for comparability with previous work.

SR-GAP VS SR-CGAP In most experimental settings, SR-CGAP slightly underperformed SR-GAP. This result came as a surprise, as both compound actions for discontinuities (Maier, 2015) and ghost reductions (Crabbé, 2014) were reported to improve parsing.

We hypothesize that this result is due to the rarity of unary constituents in the datasets and to the difficulty to predict COMPOUNDGAPS with a bounded look at D and S caused by our practical definition of feature templates (Table 2). In contrast, predicting gaps separately involves feature extraction at each step, which crucially helps.

Feature Sets The EXTENDED feature set outperforms the baseline by up to one point of F1. This emphasizes that information about gapped non-terminal is important. The SPANS feature set gives another 1 point improvement. This demonstrates clearly the usefulness of span features for discontinuous parsing. A direct extension of this feature set would include information about the boundaries of gaps in a discontinuous constituent. A difficulty of this extension is that the number of gaps in a constituent can vary.

[5]These are included in `http://pauillac.inria.fr/~seddah/evalb_spmrl2013.tar.gz`. In this setting, we reattach punctuation to the root node before evaluation.

Beam size	TIGERHN8 F1	TIGERHN8 Disc. F1	TIGERM15 F1	TIGERM15 Disc. F1
2	81.86	48.49	84.28	49.04
4	83.27	53.00	85.43	53.14
8	83.61	54.42	85.93	55.00
16	83.84	54.81	86.13	56.17
32	84.32	56.22	86.10	55.50
64	84.14	56.01	86.30	56.90
128	84.05	55.76	86.13	57.04

Table 4: Results on development sets for different beam sizes.

3.4 Comparisons with Previous Works

There are three main approaches to direct discontinuous parsing.[6] One such approach is based on unprojective or pseudo-projective dependency parsing (Hall and Nivre, 2008; Fernández-González and Martins, 2015), and aims at enriching dependency labels in such a way that constituents can be retrieved from the dependency tree. The advantage of such systems is that they can use off-the-shelf dependency parsers with rich features and efficient inference.

[6]As opposed to non-direct strategies, based for example on PCFG parsing and post-processing.

The second approach is chart-based parsing, as examplified by the DOP (Data Oriented Parsing) models of van Cranenburgh (2012) and van Cranenburgh et al. (2016) and Probabilistic LCFRS (Kallmeyer and Maier, 2013; Evang and Kallmeyer, 2011).

The last paradigm is transition-based parsing. Versley (2014a) and Versley (2014b) use an easy-first strategy with a swap transition. Maier (2015) and Maier and Lichte (2016) use a shift-reduce algorithm augmented with a swap transition.

Table 3 includes recent results from these various parsers. The most successful approach so far is that of Fernández-González and Martins (2015), which outperforms by a large margin transition-based parsers (Maier, 2015; Maier and Lichte, 2016).

SR-GAP VS SR-SWAP In the same settings (baseline features and beam size of 4), SR-GAP outperforms SR-SWAP by a large margin on all datasets. It is also twice as accurate when we only consider discontinuous constituents.

In Section 3.5, we analyse the properties of both transition systems and give hypotheses for the performance difference.

Absolute Scores On all datasets, our model reaches or outperforms the state of the art (Fernández-González and Martins, 2015). This is still the case in a more realistic experimental setup with predicted tags, as reported in Table 5.[7]

As pointed out by Maier and Lichte (2016), a limitation of shift-reduce based parsing is the locality of the feature scope when performing the search. The parser could be in states where the necessary information to take the right parsing decision is not accessible with the current scoring model.

To get more insight into this hypothesis, we tested large beam sizes. If the parser maintains a much larger number of hypotheses, we hope that it could compensate for the lack of information by delaying certain decisions. In Table 4, we present additional results on development sets of both instantiations of the TIGER corpus, with different beam sizes. As was expected, a larger beam size gives better results. The beam size controls the tradeoff between speed and accuracy.[8]

TIGERM15	F1 (spmrl.prm)	
	≤ 70	All
Versley (2014b)	73.90	-
Fernández-González and Martins (2015)	77.72	77.32
SR-GAP, beam=32, +SPANS	**79.44**	**79.26**

Table 5: Results on the Tiger corpus in the SPMRL predicted tag scenario.

Interestingly, the improvement from a larger beam size is greater on discontinuous constituents than overall. For example, from 16 to 32, F1 improves by 0.5 on TIGERHN8 and F1 on discontinuous constituents improves by 1.4.

This suggests that further improvements could be obtained by augmenting the lookahead on the buffer and using features further on S and D. We plan in the future to switch to a neural model such as a bi-LSTM in order to obtain more global representations of the whole data structures (S, D, B).

3.5 Discussion: Comparing SR-SWAP and SR-GAP

This section investigates some differences between SR-SWAP and SR-GAP. We think that characterizing properties of transition systems helps to gain better intuitions into the problems inherent to discontinuous parsing.

Derivation Length Assuming that GAP or SWAP are the hardest actions to predict and are responsible for the variability of the lengths of derivation, we hypothesize that the number of these actions, hence the length of a derivation, is an important factor. Shorter derivations are less prone to error propagation.

In both cases, the shortest possible derivation for a sentence of length n corresponds to a projective tree, as the derivation will not contain any SWAP or GAP.

In the worst case, i.e. the tree that requires the longest derivation to be produced by a transition system, SR-GAP is asymptotically twice as more economical than SR-SWAP (Table 6). In Figure 5 of Appendix A, we present the trees corresponding to the longest possible derivations in both cases.

[7] Like Fernández-González and Martins (2015), we used the predicted tags provided by the SPMRL shared task organizers.

[8] Training with the full-feature model took approximately 1h30 on TIGERM15 with a beam size of 4. Parsing speed, in the same setting, is approximately $4,700$ tokens per second (corresponding to 260 sentences per second) on a single Xeon 2.30 GHz CPU.

	SR-GAP	SR-SWAP	SR-CSWAP
Theoretical longest derivation	$\frac{n^2+n}{2}$	$n^2 - n + 1$	
Longest derivation	276	2187	1276
Total number of gaps/swaps	64096	411970	
Max consecutive gaps/swaps	10	69	
Avg. deriv. length wrt n	$2.03n$	$3.09n$	$2.66n$

Table 6: Statistics on Tiger train corpus. n is the length of a sentence. SR-CSWAP is a variant of SR-SWAP proposed by Maier (2015).

These trees maximise the number of GAP and SWAP actions.

The fact that derivations are shorter with SR-GAP is confirmed empirically. In Table 6, we present several metrics computed on the train section of TIGERM15. In average, SR-SWAP derivations are empirically 50% longer than SR-GAP derivations. Despite handling discontinuities, SR-GAP derivations are not noticeably longer than those we would get with a standard shift-reduce transition system (n shifts and $n-1$ binary reductions).

Intuitively, the difference in length of derivations comes from two facts. First, swapped terminals are pushed on the buffer and must be shifted once more, whereas with SR-GAP, each token is shifted exactly once. Second, transition systems for discontinuous parsing implicitly predict an order on terminals (discontinuous trees can be transformed to continuous trees by changing the precedence order on terminals). With SR-SWAP, reordering is done by swapping two terminals. In contrast, SR-GAP can swap complex non-terminals (already ordered chunks of terminals), making the reordering more efficient in terms of number of operations.

It would be interesting to see if SR-SWAP is improved when swapping non-terminals is allowed. However, it would make feature extraction more complex, because it would no longer be assumed that the buffer contains only tokens.

The effect of the derivation length is confirmed by Maier and Lichte (2016) who explored different oracles for SR-SWAP and found that oracles producing shorter derivations gave better results.

Feature Locality A second property which explains the performance of SR-GAP is the access to three data structures (vs two for SR-SWAP) for extracting features; SR-GAP has access to an extended domain of locality. Moreover, with SR-SWAP, the semantics of features from the queue does not make a distinction between swapped tokens and tokens that have not been shifted yet. When the parser needs to predict a long sequence of consecutive swaps, it is hardly in a position to have access to the relevant information. The use of three data structures, along with shorter sequences of GAP actions, seems to alleviate this problem.

4 Conclusion

We have introduced a novel transition system for lexicalized discontinuous parsing. The SR-GAP transition system produces short derivations, compared to SR-SWAP, while being able to derive any discontinuous tree.

Our experiments show that it outperforms the best previous transition system (Maier, 2015) in similar settings and different datasets. Combined with a span-based feature set, we obtained a very efficient parser with state-of-the-art results.

We also provide an efficient C++ implementation of our parser, based on a tree-structured stack.

Direct follow-ups to this work consist in switching to a neural scoring model to improve the representations of D and S and alleviate the locality issues in feature extraction (Kiperwasser and Goldberg, 2016; Cross and Huang, 2016).

Acknowledgments

We thank three anonymous reviewers, as well as Héctor Martínez Alonso, Olga Seminck and Chloé Braud for comments and suggestions to improve prior versions of this article. We also thank Djamé Seddah for help with the SPMRL dataset.

References

Thorsten Brants. 1998. The NeGra export format for annotated corpora. Technical Report 98, Universität des Saarlandes, Saarbrücken, April.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA. Association for Computational Linguistics.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 1–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

Michael A. Covington. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102.

Benoit Crabbé. 2014. An LR-inspired generalized lexicalized phrase structure parser. In *Proceedings of the twenty-fifth International Conference on Computational Linguistics*, Dublin, Ireland, August.

Benoit Crabbé. 2015. Multilingual discriminative lexicalized phrase structure parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1847–1856, Lisbon, Portugal, September. Association for Computational Linguistics.

James Cross and Liang Huang. 2016. Incremental parsing with minimal features using bi-directional lstm. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 32–37, Berlin, Germany, August. Association for Computational Linguistics.

Amit Dubey and Frank Keller. 2003. Probabilistic parsing for german using sister-head dependencies. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 96–103, Sapporo, Japan, July. Association for Computational Linguistics.

Kilian Evang and Laura Kallmeyer. 2011. Plcfrs parsing of english discontinuous constituents. In *Proceedings of the 12th International Conference on Parsing Technologies*, IWPT '11, pages 104–116, Stroudsburg, PA, USA. Association for Computational Linguistics.

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 the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1523–1533, Beijing, China, July. Association for Computational Linguistics.

Daniel Gildea. 2010. Optimal parsing strategies for linear context-free rewriting systems. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 769–776, Los Angeles, California, June. Association for Computational Linguistics.

Yoav Goldberg, Kai Zhao, and Liang Huang. 2013. Efficient implementation of beam-search incremental parsers. In *ACL (2)*, pages 628–633. The Association for Computer Linguistics.

Carlos Gómez-Rodríguez and Daniel Fernández-González. 2015. An efficient dynamic oracle for unrestricted non-projective parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 256–261, Beijing, China, July. Association for Computational Linguistics.

Carlos Gómez-Rodríguez, Marco Kuhlmann, Giorgio Satta, and David Weir. 2009. Optimal reduction of rule length in linear context-free rewriting systems. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 539–547, Boulder, Colorado, June. Association for Computational Linguistics.

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, 6th International Conference, GoTAL 2008, Gothenburg, Sweden, August 25-27, 2008, Proceedings*, volume 5221 of *Lecture Notes in Computer Science*, pages 169–180. Springer.

David Hall, Greg Durrett, and Dan Klein. 2014. Less grammar, more features. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Baltimore, Maryland, June. Association for Computational Linguistics.

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

Laura Kallmeyer and Wolfgang Maier. 2015. Lr parsing for lcfrs. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1250–1255, Denver, Colorado, May–June. Association for Computational Linguistics.

Laura Kallmeyer. 2010. *Parsing Beyond Context-Free Grammars*. Springer Publishing Company, Incorporated, 1st edition.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association of Computational Linguistics – Volume 4, Issue 1*, pages 313–327.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 423–430, Stroudsburg, PA, USA. Association for Computational Linguistics.

Wolfgang Maier and Timm Lichte. 2016. Discontinuous parsing with continuous trees. In *Proceedings of the Workshop on Discontinuous Structures in Natural Language Processing*, pages 47–57, San

Diego, California, June. Association for Computational Linguistics.

Wolfgang Maier. 2010. Direct parsing of discontinuous constituents in german. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 58–66, Los Angeles, CA, USA, June. Association for Computational Linguistics.

Wolfgang Maier. 2015. Discontinuous incremental shift-reduce parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1202–1212, Beijing, China, July. Association for Computational Linguistics.

Haitao Mi and Liang Huang. 2015. Shift-reduce constituency parsing with dynamic programming and pos tag lattice. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1030–1035, Denver, Colorado, May–June. Association for Computational Linguistics.

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, Joakim Nivre, Adam Przepiórkowski, Ryan Roth, Wolfgang Seeker, Yannick Versley, Veronika Vincze, Marcin Woliński, Alina Wróblewska, and Eric Villemonte de la Clergerie. 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, Washington, USA, October. Association for Computational Linguistics.

Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and S.V.N. Vishwanathan. 2009. Hash kernels for structured data. *J. Mach. Learn. Res.*, 10:2615–2637, December.

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

Andreas van Cranenburgh and Rens Bod. 2013. Discontinuous parsing with an efficient and accurate dop model. *Proceedings of the International Conference on Parsing Technologies (IWPT 2013)*.

Andreas van Cranenburgh, Remko Scha, and Rens Bod. 2016. Data-oriented parsing with discontinuous constituents and function tags. *J. Language Modelling*, 4(1):57–111.

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, April. Association for Computational Linguistics.

Yannick Versley. 2014a. 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, August. Dublin City University.

Yannick Versley. 2014b. Incorporating Semi-supervised Features into Discontinuous Easy-first Constituent Parsing. Technical report, SPMRL Shared Task.

Éric Villemonte de La Clergerie. 2002. Parsing mildly context-sensitive languages with thread automata. In *Proc. of COLING'02*, August.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *ACL (1)*, pages 434–443. The Association for Computer Linguistics.

A Supplemental Material

Action	Conditions
SHIFT	B is not empty. The last action is not GAP.
GAP	S has at least 2 elements. If d_0 is a temporary symbol, there must be at least one non temporary symbol in $S_{1:}$.
RU(X)	The last action is SHIFT. X is an axiom iff this is a one-word sentence.
R(R|L)(X)	S is not empty. X is an axiom iff B is empty, and S and D both have exactly one element. If X is a temporary symbol and if B is empty, there must be a non-temporary symbol in either $S_{1:}$ or $D_{1:}$.
RR(X)	s_0 is not a temporary symbol.
RL(X)	d_0 is not a temporary symbol.
IDLE	The configuration must be final, i.e. S and B are empty and the only element of S is the axiom.

Table 7: List of all constraints on actions for the SR-GAP transition system. The notation $S_{1:}$ is used to denote the elements of S without the first one.

A.1 Longest Derivation Computation

This section details the computation of the longest possible derivations for a sentence of length n. For the sake of simplicity, we ignore unary constituents.

		Number or index of sentences
NEGRA-30	train/dev/test	14669/1833/1833
NEGRA-ALL	train/test/dev	18602/1000/1000
TIGERM15	train/dev/test	40468/5000/5000
TIGERHN08	train/dev	$i \pmod{10} > 1 / i \equiv 1 \pmod{10}$
TIGERHN08	train/test	$i \not\equiv 0 \pmod{10} / i \equiv 0 \pmod{10}$

Table 8: Details on the standard splits.

SR-GAP There are n shifts and $n-1$ binary reductions in a derivation. The longest derivation maximises the number of gap actions, by performing as many gap actions as possible before each binary reductions. When S contains k elements, there are $k-1$ possible consecutive gap actions. So the longest derivation starts by n shifts, followed by $n-2$ gap actions, one binary reduction, $n-3$ gap actions, one binary reduction, and so on.

$$\begin{aligned} L_{gap}(n) &= n + ((n-2)+1) + \cdots + 1 \\ &= 1 + 2 + \cdots + n \\ &= \frac{n(n-1)}{2} \end{aligned}$$

This corresponds to the tree on the left-hand side of Figure 5.

SR-SWAP Using the oracle or Maier (2015), the longest derivation for a sentence of length n consists in maximising the number of swaps before each reduction.[9]

After the first shift, the derivation performs repeatedly $n-i$ shifts, $n-i-1$ swaps and one reduction, i being the number of shifted terminals before each iteration.

$$\begin{aligned} L_{swap}(n) &= 1 + \sum_{i=1}^{n-1} ((n-i) + (n-i-1) + 1) \\ &= 1 + 2\sum_{i=1}^{n-1} (n-i) \\ &= 1 + 2\frac{n(n-1)}{2} \\ &= n^2 - n + 1 \end{aligned}$$

This corresponds to the tree on the right-hand side of Figure 5.

[9] Other possible oracles (Maier and Lichte, 2016) are more efficient on this example and could have different (and better) worst cases.

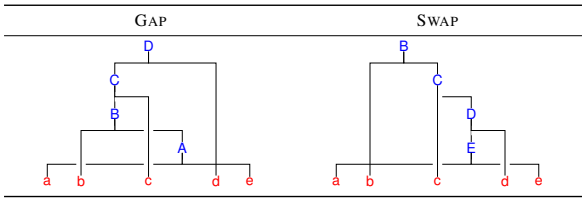

Figure 5: Example tree corresponding to the longest derivation for a sentence of length 5 with GAP and SWAP.