Incremental Discontinuous Phrase Structure Parsing with the GAP Transition

Maximin Coavoux1,2 and Benoît Crabré1,2,3
1Univ. Paris Diderot, Sorbonne Paris Cité
2Laboratoire de linguistique formelle (LLF, CNRS)
3Institut 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 $O(n^{3k})$, 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.

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).
Contributions We introduce a novel transition system for discontinuous parsing we call SR-GAP. We evaluate this algorithm on two German tree-banks 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-UNARY-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  \[ t_1[w_1]t_2[w_2] \ldots t_n[w_n] \]
Axiom  \[ \langle \epsilon, \epsilon, t_1[w_1]t_2[w_2] \ldots t_n[w_n] \rangle \]
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)**  \[ \frac{\langle S| S_0[h], D| d_0[h'], B \rangle}{\langle S, S_0[h], D, B \rangle} \]
**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**, **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 \( \text{rindex}(n) < \text{rindex}(n') \) 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.
generates 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 \( n \) \textsc{shifts} and \( n - 1 \) binary reductions. In contrast, the number of unary reductions and \textsc{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 approximately 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 \textsc{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. \textsc{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

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
\textbf{S} & \textbf{D} & \textbf{B} & \textbf{Action} \\
\hline
Es bestünde & somit hinreichender Spielraum & SH & \\
Es bestünde & somit hinreichender Spielraum & SH & \\
Es bestünde & somit hinreichender Spielraum & SH & \\
Es bestünde somit & Spielraum & SH & \\
Es bestünde somit hinreichender Spielraum & NP[Spielraum] & RR(NP) & \\
Es bestünde somit NP[Spielraum] & GAP & \\
Es bestünde somit NP[Spielraum] & GAP & \\
bestünde somit NP[Spielraum] & GAP & \\
bestünde somit NP[Spielraum] & GAP & \\
bestünde somit & NP[Spielraum] & RR(NP) & \\
bestünde somit & NP[Spielraum] & RR(NP) & \\
smot & S:[bestünde] & RL(S) & \\
S[bestünde] & S & RR(S) & \\
\hline
\end{tabular}
\caption{Example derivation for the sentence in Figure 2, part-of-speech tags are omitted.}
\end{table}
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 action \( (i \in \{0, \ldots, m\}) \) 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 COMPOUND\(\text{SWAP}\) 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 \( O(k \cdot n^2) \) 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 discontinuities 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 \( O(k \cdot (n + g)) \) 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 Ne- gra. **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. **TIGER-HN08** is the split used by Hall and Nivre (2008). **TIGER-M15** 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 headdrules 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 $s_1$ are generated by the same child of $s_i$.

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.  

**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

---

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

<table>
<thead>
<tr>
<th>Method</th>
<th>NEGRA-30 All</th>
<th>NEGRA-ALL All</th>
<th>TIGERHN8 F1</th>
<th>TIGERM15 F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>dep2const</td>
<td>82.56</td>
<td>81.08</td>
<td>80.52</td>
<td>85.53</td>
</tr>
<tr>
<td>Hall and Nivre (2008)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>van Cranenburgh (2012)</td>
<td>DOP</td>
<td>-</td>
<td>72.33</td>
<td>71.08</td>
</tr>
<tr>
<td>van Cranenburgh and Bod (2013)</td>
<td>DOP</td>
<td>-</td>
<td>76.8</td>
<td>-</td>
</tr>
<tr>
<td>Kallmeyer and Maier (2013)</td>
<td>LCRFS</td>
<td>75.75</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Verdey (2014a)</td>
<td>EA-Fi</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Maier (2015)</td>
<td>SR-SWAP</td>
<td>75.17 (15.76)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Maier and Lichte (2016)</td>
<td>SR-SWAP</td>
<td>76.95 (19.82)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>This work, beam=4</td>
<td>SR-GAP</td>
<td>79.31 (38.66)</td>
<td>79.29 (39.78)</td>
<td>78.53 (38.64)</td>
</tr>
<tr>
<td>GAP, BASELINE</td>
<td>SR-GAP</td>
<td>80.44 (41.13)</td>
<td>80.34 (43.42)</td>
<td>79.79 (43.56)</td>
</tr>
<tr>
<td>GAP, +SPANS</td>
<td>SR-GAP</td>
<td>81.64 (42.94)</td>
<td>81.70 (47.17)</td>
<td>81.28 (46.85)</td>
</tr>
<tr>
<td>CGAP, BASELINE</td>
<td>SR-CGAP</td>
<td>79.61 (41.06)</td>
<td>79.32 (43.49)</td>
<td>78.64 (42.13)</td>
</tr>
<tr>
<td>CGAP, +EXTENDED</td>
<td>SR-CGAP</td>
<td>80.26 (40.52)</td>
<td>80.48 (43.42)</td>
<td>79.98 (42.60)</td>
</tr>
<tr>
<td>CGAP, +SPANS</td>
<td>SR-CGAP</td>
<td>81.16 (42.39)</td>
<td>81.41 (44.73)</td>
<td>80.89 (44.13)</td>
</tr>
</tbody>
</table>

This work, beam=32

<table>
<thead>
<tr>
<th>Beam size</th>
<th>TIGERHN8 F1</th>
<th>TIGERM15 F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>81.86</td>
<td>84.28</td>
</tr>
<tr>
<td>4</td>
<td>83.27</td>
<td>85.43</td>
</tr>
<tr>
<td>8</td>
<td>83.61</td>
<td>85.93</td>
</tr>
<tr>
<td>16</td>
<td>83.84</td>
<td>86.13</td>
</tr>
<tr>
<td>32</td>
<td>84.32</td>
<td>86.10</td>
</tr>
<tr>
<td>64</td>
<td>84.14</td>
<td>86.30</td>
</tr>
<tr>
<td>128</td>
<td>84.05</td>
<td>86.13</td>
</tr>
</tbody>
</table>

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

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 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.

### 3.4 Comparisons with Previous Works

There are three main approaches to direct discontinuous parsing. 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.

---

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

6As opposed to non-direct strategies, based for example on PCFG parsing and post-processing.
The second approach is chart-based parsing, as exemplified 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.

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.

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.

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

<table>
<thead>
<tr>
<th></th>
<th>F1 (spmrl.prm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Versley (2014b)</td>
<td>73.90</td>
</tr>
<tr>
<td>Fernández-González and Martins (2015)</td>
<td>77.72</td>
</tr>
<tr>
<td>SR-GAP, beam=32, +SPANS</td>
<td>79.44</td>
</tr>
<tr>
<td>All</td>
<td>79.26</td>
</tr>
</tbody>
</table>

*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 1,700 tokens per second (corresponding to 260 sentences per second) on a single Xeon 2.30 GHz CPU.

---

7 Like Fernandez-Gonzalez and Martins (2015), we used the predicted tags provided by the SPMRL shared task organizers.
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 TIGER-M15. 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


A Supplemental Material

<table>
<thead>
<tr>
<th>Action</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SHIFT</strong></td>
<td>( B ) is not empty.</td>
</tr>
<tr>
<td><strong>GAP</strong></td>
<td>( S ) has at least 2 elements. If ( d_0 ) is a temporary symbol, there must be at least one non temporary symbol in ( S_L ).</td>
</tr>
<tr>
<td><strong>RU(X)</strong></td>
<td>The last action is <strong>SHIFT</strong>. ( X ) is an axiom iff this is a one-word sentence.</td>
</tr>
<tr>
<td><strong>R(L)(X)</strong></td>
<td>( 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 ( B ) is empty, there must be a non-temporary symbol in either ( S_L ) or ( D_L ).</td>
</tr>
<tr>
<td><strong>RR(X)</strong></td>
<td>( s_0 ) is a temporary symbol.</td>
</tr>
<tr>
<td><strong>RL(X)</strong></td>
<td>( d_b ) is not a temporary symbol.</td>
</tr>
<tr>
<td><strong>IDLE</strong></td>
<td>The configuration must be final, i.e. ( S ) and ( B ) are empty and the only element of ( S ) is the axiom.</td>
</tr>
</tbody>
</table>

Table 7: List of all constraints on actions for the SR-GAP transition system. The notation \( S_L \) 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.
Table 8: Details on the standard splits.

<table>
<thead>
<tr>
<th>Number or index of sentences</th>
<th>NEGRA-30 train/dev/test</th>
<th>14669/1833/1833</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEGRA-ALL train/test dev</td>
<td>18602/1000/1000</td>
<td></td>
</tr>
<tr>
<td>TIGER-M15 train/dev/test</td>
<td>40468/5000/5000</td>
<td></td>
</tr>
<tr>
<td>TIGER-HN08 train/test</td>
<td>i (mod 10) &gt; i/i ≡ 1 (mod 10)</td>
<td></td>
</tr>
<tr>
<td>TIGER-HN08 i = (mod 10)</td>
<td>i/i ≡ 0 (mod 10)</td>
<td></td>
</tr>
</tbody>
</table>

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 reduction. 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.

\[
L_{gap}(n) = n + ((n - 2) + 1) + \cdots + 1
= 1 + 2 + \cdots + n
= \frac{n(n - 1)}{2}
\]

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.

\[
L_{swap}(n) = 1 + \sum_{i=1}^{n-1} ((n - i) + (n - i - 1) + 1)
= 1 + 2 \sum_{i=1}^{n-1} (n - i)
= 1 + 2 \frac{n(n - 1)}{2}
= n^2 - n + 1
\]

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.