Global Transition-based Non-projective Dependency Parsing

Carlos Gómez-Rodríguez
Universidade da Coruña
carlos.gomez@udc.es

Tianze Shi
Cornell University
tianze@cs.cornell.edu

Lillian Lee
Cornell University
llee@cs.cornell.edu

Abstract

Shi, Huang, and Lee (2017a) obtained state-of-the-art results for English and Chinese dependency parsing by combining dynamic-programming implementations of transition-based dependency parsers with a minimal set of bidirectional LSTM features. However, their results were limited to projective parsing. In this paper, we extend their approach to support non-projectivity by providing the first practical implementation of the MH₄ algorithm, an $O(n^4)$ mildly non-projective dynamic-programming parser with very high coverage on non-projective treebanks. To make MH₄ compatible with minimal transition-based feature sets, we introduce a transition-based interpretation of it in which parser items are mapped to sequences of transitions. We thus obtain the first implementation of global decoding for non-projective transition-based parsing, and demonstrate empirically that it is more effective than its projective counterpart in parsing a number of highly non-projective languages.

1 Introduction

Transition-based dependency parsers are a popular approach to natural language parsing, as they achieve good results in terms of accuracy and efficiency (Yamada and Matsumoto, 2003; Nivre and Scholz, 2004; Zhang and Nivre, 2011; Chen and Manning, 2014; Dyer et al., 2015; Andor et al., 2016; Kiperwasser and Goldberg, 2016). Until very recently, practical implementations of transition-based parsing were limited to approximate inference, mainly in the form of greedy search or beam search. While cubic-time exact inference algorithms for several well-known projective transition systems had been known since the work of Huang and Sagae (2010) and Kuhlmann et al. (2011), they had been considered of theoretical interest only due to their incompatibility with rich feature models: incorporation of complex features resulted in jumps in asymptotic runtime complexity to impractical levels.

However, the recent popularization of bi-directional long-short term memory networks (bi-LSTMs; Hochreiter and Schmidhuber, 1997) to derive feature representations for parsing, given their capacity to capture long-range information, has demonstrated that one may not need to use complex feature models to obtain good accuracy (Kiperwasser and Goldberg, 2016; Cross and Huang, 2016). In this context, Shi et al. (2017a) presented an implementation of the exact inference algorithms of Kuhlmann et al. (2011) with a minimal set of only two bi-LSTM-based feature vectors. This not only kept the complexity cubic, but also obtained state-of-the-art results in English and Chinese parsing.

While their approach provides both accurate parsing and the flexibility to use any of greedy, beam, or exact decoding with the same underlying transition systems, it does not support non-projectivity. Trees with crossing dependencies make up a significant portion of many treebanks, going as high as 63% for the Ancient Greek treebank in the Universal Dependencies¹ (UD) dataset version 2.0 and averaging around 12% over all languages in UD 2.0. In this paper, we extend Shi et al.’s (2017a) approach to mildly non-projective parsing in what, to our knowledge, is the first implementation of exact decoding for a non-projective transition-based parser.

As in the projective case, a mildly non-

¹http://universaldependencies.org/
The MH4 dynamic programming parser, which has been shown to pro-
provide very good non-projective coverage in \(O(n^4)\) time (Gómez-Rodríguez, 2016). While the MH4 parser was originally presented as a non-projective generalization of the dynamic program that later led to the arc-hybrid transition system (Gómez-Rodríguez et al., 2008; Kühlmann et al., 2011), its own relation to transition-based parsing was not known. Here, we show that MH4 can be interpreted as exploring a subset of the search space of a transition-based parser that generalizes the arc-hybrid system, under a mapping that differs from the “push computation” paradigm used by the previously-known dynamic-programming decoders for transition systems. This allows us to extend Shi et al. (2017a)’s work to non-projective parsing, by implementing MH4 with a minimal set of transition-based features.

Experimental results show that our approach outperforms the projective approach of Shi et al. (2017a) and maximum-spanning-tree non-projective parsing on the most highly non-projective languages in the CoNLL 2017 shared-task data that have a single treebank. We also compare with the third-order 1-Endpoint-Crossing (1EC) parser of Pitler (2014), the only other practical implementation of an exact mildly non-projective decoder that we know of, which also runs in \(O(n^4)\) but without a transition-based interpretation. We obtain comparable results for these two algorithms, in spite of the fact that the MH4 algorithm is notably simpler than 1EC. The MH4 parser remains effective in parsing projective treebanks, while our baseline parser, the fully non-projective maximum spanning tree algorithm, falls behind due to its unnecessarily large search space in parsing these languages. Our code, including our re-implementation of the third-order 1EC parser with neural scoring, is available at https://github.com/tzshi/mh4-parser-acl18.

2 Non-projective Dependency Parsing

In dependency grammar, syntactic structures are modeled as word-word asymmetrical subordinate relations among lexical entries (Kübler et al., 2009). These relations can be represented in a graph. For a sentence \(w = w_1, \ldots, w_n\), we first define a corresponding set of nodes \(\{0, 1, 2, \ldots, n\}\), where 0 is an artificial node denoting the root of the sentence. Dependency relations are encoded by edges of the form \((h, m)\), where \(h\) is the head and \(m\) the modifier of the bilexical subordinate relation.\(^2\)

As is conventional, we assume two more properties on dependency structures. First, each word has exactly one syntactic head, and second, the structure is acyclic. As a consequence, the edges form a directed tree rooted at node 0.

We say that a dependency structure is projective if it has no crossing edges. While in the CoNLL and Stanford conversions of the English Penn Treebank, over 99% of the sentences are projective (Chen and Manning, 2014) — see Fig. 1 for a non-projective English example — for other languages’ treebanks, non-projectivity is a common occurrence (see Table 3 for some statistics). This paper is targeted at learning parsers that can handle non-projective dependency trees.

3 \(MH_4\) Deduction System and Its Underlying Transition System

3.1 The \(MH_4\) Deduction System

The \(MH_4\) parser is the instantiation for \(k = 4\) of Gómez-Rodríguez et al.’s (2011) more general \(MH_k\) parser. \(MH_k\) stands for “multi-headed with at most \(k\) heads per item”: items in its deduction system take the form \([h_1, \ldots, h_p]\) for \(p \leq k\), indicating the existence of a forest of \(p\) dependency subtrees headed by \(h_1, \ldots, h_p\) such that their yields are disjoint and the union of their

\(^2\)To simplify exposition here, we only consider the unlabeled case. We use a separately-trained labeling module to obtain labeled parsing results in §5.
yields is the contiguous substring $h_1 \ldots h_p$ of the input. Deduction steps, shown in Figure 2, can be used to join two such forests that have an endpoint in common via graph union (combine); or to add a dependency arc to a forest that attaches an interior head as a dependent of any of the other heads (link).

In the original formulation by Gómez-Rodríguez et al. (2011), all valid items of the form $[i, i + 1]$ are considered to be axioms. In contrast, we follow Kuhlmann et al.’s (2011) treatment of $MH_3$: we consider $[0, 1]$ as the only axiom and include an extra shift step to generate the rest of the items of that form. Both formulations are equivalent, but including this shift rule facilitates giving the parser a transition-based interpretation.

Higher values of $k$ provide wider coverage of non-projective structures at an asymptotic runtime complexity of $O(n^k)$. When $k$ is at its minimum value of 3, the parser covers exactly the set of projective trees. And in fact, it can be seen as a transformation of the deduction system described in Gómez-Rodríguez et al. (2008) that gave rise to the projective arc-hybrid parser (Kuhlmann et al., 2011). For $k \geq 4$, the parser covers an increasingly larger set of non-projective structures. While a simple characterization of these sets has been lacking, empirical evaluation on a large number of treebanks (Gómez-Rodríguez, 2016) has shown $MH_k$ to provide the best known tradeoff between asymptotic complexity and efficiency for $k > 4$. When $k = 4$, its coverage is second only to the 1-EndPoint-Crossing parser of Pittler et al. (2013). Both parsers fully cover well over 80% of the non-projective trees observed in the studied treebanks.

3.2 The $MH_4$ Transition System

Kuhlmann et al. (2011) show how the items of a variant of $MH_3$ can be given a transition-based interpretation under the “push computation” framework, yielding the arc-hybrid projective transition system. However, such a derivation has not been made for the non-projective case ($k > 3$), and the known techniques used to derive previous associations between tabular and transition-based parsers do not seem to be applicable in this case. The specific issue is that the deduction systems of Kuhlmann et al. (2011) and Cohen et al. (2011) have in common that the structure of their derivations is similar to that of a Dyck (or balanced-brackets) language, where steps corresponding to shift transitions are balanced with those corresponding to reduce transitions. This makes it possible to group derivation subtrees, and the transition sequences that they yield, into “push computations” that increase the length of the stack by a constant amount. However, this does not seem possible in $MH_4$.

Instead, we derive a transition-based interpretation of $MH_4$ by a generalization of that of $MH_3$ that departs from push computations.

To do so, we start with the $MH_3$ interpretation of an item $[i, j]$ given by Kuhlmann et al. (2011). This item represents a set of computations (transition sequences) that start from a configuration of the form $(\sigma, i|\beta, A)$ (where $\sigma$ is the stack and $i|\beta$ is the buffer, with $i$ being the first buffer node) and take the parser to a configuration of the form $(\sigma|i, j|\beta', A)$. That is, the computation has the net effect of placing node $i$ on top of the previous contents of the stack, and it ends in a state where the first buffer element is $j$.

Under this item semantics, the combine deduction step of the $MH_3$ parser (i.e., the instantiation of the one in Fig. 2 for $k = 3$) simply concatenates transition sequences. The shift step generates a sequence with a single arc-hybrid shift.
transition:

\[ \text{sh}: (\sigma, h_m | \beta, A) \rightarrow (\sigma | h_m, \beta, A) \]

and the two possible instantiations of the COMBINE step when \( k = 3 \) take the antecedent transition sequence and add a transition to it, namely, one of the two arc-hybrid reduce transitions. Written in the context of the node indexes used in Figure 2, these are the following:

\[
\begin{align*}
(\sigma | h_1 | h_2, h_3 | \beta, A) & \rightarrow (\sigma | h_1, h_3 | \beta, A \cup \{ h_3 \rightarrow h_2 \}) \\
(\sigma | h_1 | h_2, h_3 | \beta, A) & \rightarrow (\sigma | h_1, h_3 | \beta, A \cup \{ h_1 \rightarrow h_2 \})
\end{align*}
\]

where \( h_1 \) and \( h_3 \) respectively can be simplified out to obtain the well-known arc-hybrid transitions:

\[
\begin{align*}
\text{la} : (\sigma | h_2, h_3 | \beta, A) & \rightarrow (\sigma, h_3 | \beta, A \cup \{ h_3 \rightarrow h_2 \}) \\
\text{ra} : (\sigma | h_1 | h_2, \beta, A) & \rightarrow (\sigma | h_1, \beta, A \cup \{ h_1 \rightarrow h_2 \})
\end{align*}
\]

Now, we assume the following generalization of the item semantics: an item \([h_1, \ldots, h_m]\) represents a set of computations that start from a configuration of the form \((\sigma | h_1 | \beta, A)\) and lead to a configuration of the form \((\sigma | h_1 | \ldots | h_{m-1}, h_m | \beta', A)\). Note that this generalization no longer follows the “push computation” paradigm of Kuhlmann et al. (2011) and Cohen et al. (2011) because the number of nodes pushed onto the stack depends on the value of \( m \).

Under this item semantics, the SHIFT and COMBINE steps have the same interpretation as for MH3. In the case of the LINK step, following the same reasoning as for the MH3 case, we obtain the following transitions:

\[
\begin{align*}
\text{la} : (\sigma | h_2, h_3 | \beta, A) & \rightarrow (\sigma, h_4 | \beta, A \cup \{ h_4 \rightarrow h_3 \}) \\
\text{ra} : (\sigma | h_2, h_3 | \beta, A) & \rightarrow (\sigma | h_2, \beta, A \cup \{ h_2 \rightarrow h_3 \}) \\
\text{la}' : (\sigma | h_2, h_3, h_4 | \beta, A) & \rightarrow (\sigma | h_3, h_4 | \beta, A \cup \{ h_3 \rightarrow h_2 \}) \\
\text{ra}' : (\sigma | h_1 | h_2, h_3 | \beta, A) & \rightarrow (\sigma | h_1, h_3 | \beta, A \cup \{ h_1 \rightarrow h_3 \}) \\
\text{la}_2 : (\sigma | h_2, h_3, h_4 | \beta, A) & \rightarrow (\sigma | h_3, h_4 | \beta, A \cup \{ h_4 \rightarrow h_2 \}) \\
\text{ra}_2 : (\sigma | h_1 | h_2, h_3, \beta, A) & \rightarrow (\sigma | h_1, h_2 | \beta, A \cup \{ h_1 \rightarrow h_3 \})
\end{align*}
\]

These transitions give us the MH4 transition system: a parser with four projective reduce transitions (la,ra,la’,ra’) and two Attardi-like, non-adjacent-arc reduce transitions (la2 and ra2).

It is worth mentioning that this MH4 transition system we have obtained is the same as one of the variants of Attardi’s algorithm introduced by Shi et al. (2018), there called ALLs0s1. However, in that paper they show that it can be tabularized in \( O(n^4) \) using the push computation framework. Here, we have derived it as an interpretation of the \( O(n^4) \) MH4 parser.

However, in this case the dynamic programming algorithm does not cover the full search space of the transition system: while each item in the MH4 parser can be mapped into a computation of this MH4 transition-based parser, the opposite is not true. This tree:

```
0 1 2 3 4 5
```

can be parsed by the transition system using the computation

\[ \text{sh}(0): \text{sh}(1); \text{sh}(2); \text{la}_2(3 \rightarrow 1); \text{sh}(3); \text{sh}(4); \text{la}_2(5 \rightarrow 3); \text{sh}(5); \text{ra}(4 \rightarrow 5); \text{ra}(2 \rightarrow 4); \text{ra}(0 \rightarrow 2) \]

but it is not covered by the dynamic programming algorithm, as no deduction sequence will yield an item representing this transition sequence. As we will see, this issue will not prevent us from implementing a dynamic-programming parser with transition-based scoring functions, or from achieving good practical accuracy.

4 Model

Given the transition-based interpretation of the MH4 system, the learning objective becomes to find a computation that gives the gold-standard parse. For each sentence \( w_1, \ldots, w_n \), we train parsers to produce the transition sequence \( \tau^* \) that corresponds to the annotated dependency structure. Thus, the model consists of two components: a parameterized scorer \( S(\tau) \), and a decoder that finds a sequence \( \hat{\tau} \) as prediction based on the scoring.

As discussed by Shi et al. (2017a), there exists some tension between rich-feature scoring models and choices of decoders. Ideally, a globally-optimal decoder finds the maximum-scoring transition sequence \( \hat{\tau} \) without brute-force searching the exponentially-large output space. To keep the runtime of our exact decoder at a practical low-order polynomial, we want its feature set to be
Table 1: Performance of local parsing models with varying number of features. We report average UAS over 10 languages on UD 2.0.

<table>
<thead>
<tr>
<th>Features</th>
<th>{s_0, b_0}</th>
<th>{s_1, s_0, b_0}</th>
<th>{s_2, s_1, s_0, b_0}</th>
</tr>
</thead>
<tbody>
<tr>
<td>UAS</td>
<td>49.83</td>
<td>85.17</td>
<td>85.27</td>
</tr>
</tbody>
</table>

minimal, consulting as few stack and buffer positions as possible. In what follows, we use \(s_0\) and \(s_1\) to denote the top two stack items and \(b_0\) and \(b_1\) to denote the first two buffer items.

### 4.1 Scoring and Minimal Features

This section empirically explores the lower limit on the number of necessary positional features. We experiment with both local and global decoding strategies. The parsers take features extracted from parser configuration \(c\), and score each valid transition \(t\) with \(S(t; c)\). The local parsers greedily take transitions with the highest score until termination, while the global parsers use the scores to find the globally-optimal solutions \(\hat{t} = \arg\max_{v_S(t)} S(t)\), where \(S(t)\) is the sum of scores for the component transitions.

Following prior work, we employ bi-LSTMs for compact feature representation. A bi-LSTM runs in both directions on the input sentence, and assigns a context-sensitive vector encoding to each token in the sentence: \(w_1, \ldots, w_n\). When we need to extract features, say, \(s_0\), from a particular stack or buffer position, say \(s_0\), we directly use the bi-LSTM vector \(w_{s_0}\), where \(i_{s_0}\) gives the index of the subroot of \(s_0\) into the sentence.

Shi et al. (2017a) showed that feature vectors \(\{s_0, b_0\}\) suffice for \(MH_3\). Table 1 and Table 2 show the use of small feature sets for \(MH_4\), for local and global parsing models, respectively. For a local parser to exhibit decent performance, we need at least \(\{s_1, s_0, b_0\}\), but adding \(s_2\) on top of that does not show any significant impact on the performance. Interestingly, in the case of global models, the two-vector feature set \(\{s_0, b_0\}\) already suffices. Adding \(s_1\) to the global setting (column “Hybrid” in Table 2) seems attractive, but entails resolving a technical challenge that we discuss in the following section.

### 4.2 Global Decoder

In our transition-system interpretation of \(MH_k\), sh transitions correspond to SHIFT and reduce transitions reflect the LINK steps. Since the \(\text{SHIFT}\) conclusions lose the contexts needed to score the transitions, we set the scores for all \(\text{SHIFT}\) rules to zero and delegate the scoring of the sh transitions to the \(\text{COMBINE}\) steps, as in Shi et al. (2017a); for example,

\[
[h_1, h_2]: v_1 \quad [h_2, h_3, h_4]: v_2 \\
[h_1, h_2, h_3, h_4]: v_1 + v_2 + S(\text{sh}; \{h_1, h_2\})
\]

Here the transition sequence denoted by \([h_2, h_3, h_4]\) starts from a sh, with \(h_1\) and \(h_2\) taking the \(s_0\) and \(b_0\) positions. If we further wish to access \(s_1\), such information is not readily available in the deduction step, apparently requiring extra bookkeeping that pushes the space and time complexity to an impractical \(O(n^4)\) and \(O(n^5)\), respectively. But, consider the scoring for the reduce transitions in the \(\text{LINK}\) steps:

\[
[h_1, h_2, h_3, h_4]: v \\
[h_1, h_2, h_3]: v + S(\text{sh}; \{h_2, h_3, h_4\})
\]

The deduction steps already keep indices for \(s_1\) (\(h_2\) in the first rule, \(h_1\) in the second) and thus provide direct access without any modification. To resolve the conflict between including \(s_1\) for richer representations and the unavailability of \(s_1\) in scoring the sh transitions in the \(\text{COMBINE}\) steps, we propose a hybrid scoring approach — we use features \(\{s_0, b_0\}\) when scoring a \(\text{sh}\) transition, and features \(\{s_1, s_0, b_0\}\) for consideration of reduce transitions. We call this method \(MH_4\)-hybrid, in contrast to \(MH_4\)-two, where we simply take \(\{s_0, b_0\}\) for scoring all transitions.

### 4.3 Large-Margin Training

We train the greedy parsers with hinge loss, and the global parsers with its structured version (Taskar et al., 2005). The loss function for each sentence is formally defined as:

\[
\max_{\hat{t}} \left( S(\hat{t}) + \text{cost}(t^*, \hat{t}) - S(t^*) \right)
\]
where the margin \( \text{cost}(t^*, \hat{t}) \) counts the number of mis-attached nodes for taking sequence \( \hat{t} \) instead of \( t^* \). Minimizing this loss can be thought of as optimizing for the attachment scores.

The calculation of the above loss function can be solved as efficiently as the deduction system if the \( \text{cost} \) function decomposes into the dynamic program. We achieve this by replacing the scoring of each reduce step by its cost-augmented version:

\[
[h_1, h_2, h_3, h_4] : v + S(la_2; \{h_2, h_3, h_4\}) + \Delta
\]

where \( \Delta = 1(\text{head}(w_{h_4}) \neq w_{h_4}) \). This loss function encourages the model to give higher contrast between gold-standard and wrong predictions, yielding better generalization results.

5 Experiments

Data and Evaluation We experiment with the Universal Dependencies (UD) 2.0 dataset used for the CoNLL 2017 shared task (Zeman et al., 2017). We restrict our choice of languages to be those with only one training treebank, for a better comparison with the shared task results.\(^5\) Among these languages, we pick the top 10 non-projective languages. Their basic statistics are listed in Table 3. For all development-set results, we assume gold-standard tokenization and sentence delimitation. When comparing to the shared task results on test sets, we use the provided baseline UDPipe (Straka et al., 2016) segmentation. Our models do not use part-of-speech tags or morphological tags as features, but rather leverage such information via stack propagation (Zhang and Weiss, 2016), i.e., we learn to predict them as a secondary training objective. We report unlabeled attachment F1-scores (UAS) on the development sets for better focus on comparing our (unlabeled) parsing modules. We report its labeled variant (LAS), the main metric of the shared task, on the test sets. For each experiment setting, we ran the model with 5 different random initializations, and report the mean and standard deviation. We detail the implementation details in the supplementary material.

Baseline Systems For comparison, we include three baseline systems with the same underlying feature representations and scoring paradigm. All the following baseline systems are trained with the cost-augmented large-margin loss function.

The \( MH_3 \) parser is the projective instantiation of the \( MH_k \) parser family. This corresponds to the global version of the arc-hybrid transition system (Kuhlmann et al., 2011). We adopt the minimal feature representation \( \{s_0, b_0\} \), following Shi et al. (2017a). For this model, we also implement a greedy incremental version.

The edge-factorized non-projective maximal spanning tree (MST) parser allows arbitrary non-projective structures. This decoding approach has been shown to be very competitive in parsing non-projective treebanks (McDonald et al., 2005), and was deployed in the top-performing system at the CoNLL 2017 shared task (Dozat et al., 2017). We score each edge individually, with the features being the bi-LSTM vectors \( \{h, m\} \), where \( h \) is the head, and \( m \) the modifier of the edge.

The crossing-sensitive third-order IEC parser provides a hybrid dynamic program for parsing 1-Endpoint-Crossing non-projective dependency trees with higher-order factorization (Pitler, 2014). Depending on whether an edge is crossed, we can access the modifier’s grandparent \( g \), head \( h \), and sibling \( s_i \). We take their corresponding bi-LSTM features \( \{g, h, m, s_i\} \) for scoring each edge. This is a re-implementation of Pitler (2014) with neural scoring functions.

Main Results Table 4 shows the development-set performance of our models as compared with baseline systems. MST considers non-projective structures, and thus enjoys a theoretical advantage over projective \( MH_3 \), especially for the most non-projective languages. However, it has a vastly larger output space, making the selection of correct structures difficult. Further, the scoring is edge-factorized, and does not take any structural contexts into consideration. This tradeoff leads to the similar performance of MST comparing to \( MH_3 \). In comparison, both 1EC and \( MH_4 \) are mildly non-projective parsing algorithms, limiting the size of the output space. 1EC includes higher-order features that look at tree-structural contexts; \( MH_4 \) derives its features from parsing configurations of a transition system, hence leveraging contexts within transition sequences. These considerations explain their significant improvements over MST. We also observe that \( MH_4 \) recovers more short dependencies than 1EC, while 1EC is better at longer-distance ones.
In comparison to \( MH_4 \)-two, the richer feature representation of \( MH_4 \)-hybrid helps in all our languages.

Interestingly, \( MH_4 \) and \( MH_3 \) react differently to switching from global to greedy models. \( MH_4 \) covers more structures than \( MH_3 \), and is naturally more capable in the global case, even when the feature functions are the same (\( MH_4 \)-two). However, its greedy version is outperformed by \( MH_3 \). We conjecture that this is because \( MH_4 \) explores only the same number of configurations as \( MH_3 \), despite the fact that introducing non-projectivity expands the search space dramatically.

### 6 Related Work

Exact inference for dependency parsing can be achieved in cubic time if the model is restricted to projective trees (Eisner, 1996). However, non-projectivity is needed for natural language parsers to satisfactorily deal with linguistic phenomena like topicalization, scrambling and extraposition, which cause crossing dependencies. In UD 2.0, 68 out of 70 treebanks were reported to contain
Alternatives include treebank transformation and exact inference parsers that support unrestricted non-projective complexity. For example, McDonald and Pereira (2006); Attardi (2006); Nivre (2009); Fernández-González and Gómez-Rodríguez (2017)) or focusing on sets of dependency trees that allow only restricted forms of non-projectivity. A number of such sets, called mildly non-projective classes of trees, have been identified that both exhibit good empirical coverage of the non-projective phenomena found in natural languages and are known to have polynomial-time exact parsing algorithms; see Gómez-Rodríguez (2016) for a survey.

However, most of these algorithms have not been implemented in practice due to their prohibitive complexity. For example, Corro et al. (2016) report an implementation of the WG$_1$ parser, a $O(n^3)$ mildly non-projective parser introduced in Gómez-Rodríguez et al. (2009), but it could not be run for real sentences of length greater than 20. On the other hand, Pitler et al. (2012) provide an implementation of an $O(n^5)$ parser for a mildly non-projective class of structures called gap-mining trees, but they need to resort to aggressive pruning to make it practical, exploring only a part of the search space in $O(n^4)$ time.

To the best of our knowledge, the only practical system that actually implements exact inference for mildly non-projective parsing is the 1-Endpoint-Crossing (1EC) parser of Pitler (2013; 2014), which runs in $O(n^4)$ worst-case time like the MH$_4$ algorithm used in this paper. Thus, the system presented here is the second practical implementation of exact mildly non-projective pars-
ing that has successfully been executed on real corpora.\textsuperscript{6}

Comparing with Pitler (2014)’s 1EC, our parser has the following disadvantages: (−1) It has slightly lower coverage, at least on the treebanks considered by Gómez-Rodríguez (2016). (−2) The set of trees covered by \( MH_4 \) has not been characterized with a non-operational definition, while the set of 1-Endpoint-Crossing trees can be simply defined.

However, it also has the following advantages: (+1) It can be given a transition-based interpretation, allowing us to use transition-based scoring functions and to implement the analogous algorithm with greedy or beam search apart from exact inference. No transition-based interpretation is known for 1EC. While a transition-based algorithm has been defined for a strict subset of 1-Endpoint-Crossing Interval trees (Pitler and McDonald, 2015), this is a separate algorithm with no known mapping or relation to 1EC or any other dynamic programming model. Thus, we provide the first exact inference algorithm for a non-projective transition-based parser with practical complexity. (+2) It is conceptually much simpler, with one kind of item and two deduction steps, while the 1-Endpoint-Crossing parser has five classes of items and several dozen distinct deduction steps. It is also a purely bottom-up parser, whereas the 1-Endpoint-Crossing parser does not have the bottom-up property. This property is necessary for models that involve compositional representations of subtrees (Dyer et al., 2015), and facilitates parallelization and partial parsing. (+3) It can be easily generalized to \( MH_k \) for \( k > 4 \), providing higher coverage, with time complexity \( O(n^k) \). Out of the mildly non-projective parsers studied in Gómez-Rodríguez (2016), \( MH_4 \) provides the maximum coverage with respect to its complexity for \( k > 4 \). (+4) As shown in §5, \( MH_4 \) obtains slightly higher accuracy than 1EC on average, albeit not by a conclusive margin.

It is worth noting that 1EC has recently been extended to graph parsing by Kurtz and Kuhlmann (2017), Kummerfeld and Klein (2017), and Cao et al. (2017a,b), with the latter providing a practical implementation of a parser for 1-Endpoint-Crossing, page-number-2 graphs.

\section{Conclusion}

We have extended the parsing architecture of Shi et al. (2017a) to non-projective dependency parsing by implementing the \( MH_4 \) parser, a mildly non-projective \( O(n^4) \) chart parsing algorithm, using a minimal set of transition-based bi-LSTM features. For this purpose, we have established a mapping between \( MH_4 \) items and transition sequences of an underlying non-projective transition-based parser.

To our knowledge, this is the first practical implementation of exact inference for non-projective transition-based parsing. Empirical results on a collection of highly non-projective datasets from Universal Dependencies show improvements in accuracy over the projective approach of Shi et al. (2017a), as well as edge FACTed maximum spanning-tree parsing. The results are on par with the 1-Endpoint-Crossing parser of Pitler (2014) (re-implemented under the same neural framework), but our algorithm is notably simpler and has additional desirable properties: it is purely bottom-up, generalizable to higher coverage, and compatible with transition-based semantics.

\section*{Acknowledgments}

We thank the three anonymous reviewers for their helpful comments. CG has received funding from the European Research Council (ERC), under the European Union’s Horizon 2020 research and innovation programme (FASTPARSE, grant agreement No 714150), from the TELEPARES-UDC project (FFI2014-51978-C2-2-R) and the ANSWER-ASAP project (TIN2017-85160-C2-1-R) from MINECO, and from Xunta de Galicia (ED431B 2017/01). TS and LL were supported in part by a Google Focused Research Grant to Cornell University. LL was also supported in part by NSF grant SES-1741441. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other sponsors.
References


Milan Straka, Jan Hajic, and Jana Straková. 2016. UD-Pipe: Trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, POS tagging and parsing. In *Proceedings
of the Tenth International Conference on Language Resources and Evaluation (LREC 2016), Paris, France. European Language Resources Association (ELRA).


