Abstract

We first present a minimal feature set for transition-based dependency parsing, continuing a recent trend started by Kiperwasser and Goldberg (2016a) and Cross and Huang (2016a) of using bi-directional LSTM features. We plug our minimal feature set into the dynamic-programming framework of Huang and Sagae (2010) and Kuhlmann et al. (2011) to produce the first implementation of worst-case $O(n^3)$ exact decoders for arc-hybrid and arc-eager transition systems. With our minimal features, we also present $O(n^3)$ global training methods. Finally, using ensembles including our new parsers, we achieve the best unlabeled attachment score reported (to our knowledge) on the Chinese Treebank and the “second-best-in-class” result on the English Penn Treebank.

1 Introduction

It used to be the case that the most accurate dependency parsers made global decisions and employed exact decoding. But transition-based dependency parsers (TBDPs) have recently achieved state-of-the-art performance, despite the fact that for efficiency reasons, they are usually trained to make local, rather than global, decisions and the decoding process is done approximately, rather than exactly (Weiss et al., 2015; Dyer et al., 2015; Andor et al., 2016). The key efficiency issue for decoding is as follows. In order to make accurate (local) attachment decisions, historically, TBDPs have required a large set of features in order to access rich information about particular positions in the stack and buffer of the current parser configuration. But consulting many positions means that although polynomial-time exact-decoding algorithms do exist, having been introduced by Huang and Sagae (2010) and Kuhlmann et al. (2011), unfortunately, they are prohibitively costly in practice, since the number of positions considered can factor into the exponent of the running time. For instance, Huang and Sagae employ a fairly reduced set of nine positions, but the worst-case running time for the exact-decoding version of their algorithm is $O(n^6)$ (originally reported as $O(n^7)$) for a length-$n$ sentence. As an extreme case, Dyer et al. (2015) use an LSTM to summarize arbitrary information on the stack, which completely rules out dynamic programming.

Recently, Kiperwasser and Goldberg (2016a) and Cross and Huang (2016a) applied bi-directional long short-term memory networks (Graves and Schmidhuber, 2005, bi-LSTMs) to derive feature representations for parsing, because these networks capture wide-window contextual information well. Collectively, these two sets of authors demonstrated that with bi-LSTMs, four positional features suffice for the arc-hybrid parsing system (K&G), and three suffice for arc-standard (C&H).\footnote{We note that K&G were not focused on minimizing positions, although they explicitly noted the implications of doing so: “While not explored in this work, [fewer positions] results in very compact state signatures, [which is] very appealing for use in transition-based parsers that employ dynamic-programming search” (pg. 319). C&H also noted in their follow-up (Cross and Huang, 2016b) the possibility of future work using dynamic programming thanks to simple features.}

Inspired by their work, we arrive at a minimal feature set for arc-hybrid and arc-eager: it contains only two positional bi-LSTM vectors, suffers almost no loss in performance in comparison to larger sets, and out-performs a single position. (Details regarding the situation with arc-standard can be found in §2.)

Our minimal feature set plugs into Huang and Sagae’s and Kuhlmann et al.’s dynamic program-
ming framework to produce the first implementation of $O(n^3)$ exact decoders for arc-hybrid and arc-eager parsers. We also enable and implement $O(n^3)$ global training methods. Empirically, ensembles containing our minimal-feature, globally-trained and exactly-decoded models produce the best unlabeled attachment score (UAS) reported (to our knowledge) on the Chinese Treebank and the “second-best-in-class” result on the English Penn Treebank.\footnote{Our ideas were subsequently adapted to the labeled setting by Shi, Wu, Chen, and Cheng (2017) in their submission to the CoNLL 2017 shared task on Universal Dependencies parsing. Their team achieved the second-highest labeled attachment score in general and had the top average performance on the surprise languages.}

Additionally, we provide a slight update to the theoretical connections previously drawn by Gómez-Rodríguez, Carroll, and Weir (2008, 2011) between TBDPs and the graph-based dependency parsing algorithms of Eisner (1996) and Eisner and Satta (1999), including results regarding the arc-eager parsing system.

2 A Minimal Feature Set

TBDPs incrementally process a sentence by making transitions through search states representing parser configurations. Three of the main transition systems in use today (formal introduction in §3.1) all maintain the following two data structures in their configurations: (1) a stack of partially parsed sub-trees and (2) a buffer (mostly) of unprocessed sentence tokens.

To featurize configurations for use in a scoring function, it is common to have features that extract information about the first several elements on the stack and the buffer, such as their word forms and part-of-speech (POS) tags. We refer to these as positional features, as each feature relates to a particular position in the stack or buffer. Typically, millions of sparse indicator features (often developed via manual engineering) are used.

In contrast, Chen and Manning (2014) introduce a feature set consisting of dense word-, POS-, and dependency-label embeddings. While dense, these features are for the same 18 positions that have been typically used in prior work. Recently, Kiperwasser and Goldberg (2016a) and Cross and Huang (2016a) adopt bi-directional LSTMs, which have nice expressiveness and context-sensitivity properties, to reduce the number of positions considered down to four and three, for different transition systems, respectively.

This naturally begs the question, what is the lower limit on the number of positional features necessary for a parser to perform well? Kiperwasser and Goldberg (2016a) reason that for the arc-hybrid system, the first and second items on the stack and the first buffer item — denoted by $s_0$, $s_1$, and $b_0$, respectively — are required; they additionally include the third stack item, $s_2$, because it may not be adjacent to the others in the original sentence. For arc-standard, Cross and Huang (2016a) argue for the necessity of $s_0$, $s_1$, and $b_0$.

We address the lower-limit question empirically, and find that, surprisingly, two positions suffice for the greedy arc-eager and arc-hybrid parsers. We also provide empirical support for Cross and Huang’s argument for the necessity of three features for arc-standard. In the rest of this section, we explain our experiments, run only on an English development set, that support this conclusion; the results are depicted in Table 1. We later explore the implementation implications in §3-4 and then test-set parsing-accuracy in §6.

<table>
<thead>
<tr>
<th>Feature Sets</th>
<th>Arc-standard</th>
<th>Arc-hybrid</th>
<th>Arc-eager</th>
</tr>
</thead>
<tbody>
<tr>
<td>${ \overline{s_2}, \overline{s_1}, \overline{s_0}, \overline{b_0} }$</td>
<td>93.95±0.12</td>
<td>94.08±0.13</td>
<td>93.92±0.04</td>
</tr>
<tr>
<td>${ \overline{s_1}, \overline{s_0}, \overline{b_0} }$</td>
<td>94.13±0.06</td>
<td>94.08±0.05</td>
<td>93.91±0.07</td>
</tr>
<tr>
<td>${ \overline{s_0}, \overline{b_0} }$</td>
<td>54.47±0.36</td>
<td>94.03±0.12</td>
<td>93.92±0.07</td>
</tr>
<tr>
<td>${ \overline{b_0} }$</td>
<td>47.11±0.44</td>
<td>52.39±0.23</td>
<td>79.15±0.06</td>
</tr>
</tbody>
</table>

Table 1: *Top:* English PTB dev-set UAS\% for progressively smaller sets of positional features, for greedy parsers with different transition systems. The “double-arrow” notation indicates vectors produced by a bi-directional LSTM. Internal lines highlight large performance drop-offs when a feature is deleted. *Bottom:* sizes of the minimal feature sets in Kiperwasser and Goldberg (2016a), Cross and Huang (2016a), and our work.
is the output of the bi-LSTM at time step $i$. (The double-arrow notation for these vectors emphasizes the bi-directionality of their origin). Then, for a given parser configuration, stack positions are represented by $\overrightarrow{s}_j$, defined as $\overrightarrow{w}_{i(s_j)}$ where $i(s_j)$ gives the position in the sentence of the token that is the head of the tree in $s_j$. Similarly, buffer positions are represented by $\overrightarrow{b}_j$, defined as $\overrightarrow{w}_{i(b_j)}$ for the token at buffer position $j$. Finally, as in Chen and Manning (2014), we use a multilayer perceptron to score possible transitions from the given configuration, where the input is the concatenation of some selection of the $\overrightarrow{s}_j$s and $\overrightarrow{b}_k$s. We use greedy decoders, and train the models with dynamic oracles (Goldberg and Nivre, 2013).

Table 1 reports the parsing accuracy that results for feature sets of size four, three, two, and one for three commonly-used transition systems. The data is the development section of the English Penn Treebank (PTB), and experimental settings are as described in our other experimental section, §6. We see that we can go down to three or, in the arc-hybrid and arc-eager transition systems, even two positions with very little loss in performance, but not further. We therefore call $\{\overrightarrow{s}_0, \overrightarrow{b}_0\}$ our minimal feature set with respect to arc-hybrid and arc-eager, and empirically confirm that Cross and Huang’s $\{\overrightarrow{s}_0, \overrightarrow{s}_1, \overrightarrow{b}_0\}$ is minimal for arc-standard; see Table 1 for a summary.

3 Dynamic Programming for TBDPs

As stated in the introduction, our minimal feature set from §2 plugs into Huang and Sagae and Kuhlmann et al.’s dynamic programming (DP) framework. To help explain the connection, this section provides an overview of the DP framework. We draw heavily from the presentation of Kuhlmann et al. (2011).

3.1 Three Transition Systems

Transition-based parsing (Nivre, 2008; Kübler et al., 2009) is an incremental parsing framework based on transitions between parser configurations. For a sentence to be parsed, the system starts from a corresponding initial configuration, and attempts to sequentially apply transitions until a configuration corresponding to a full parse is produced. Formally, a transition system is defined as $\mathcal{S} = (\mathcal{C}, \mathcal{T}, c^0, C_T)$, where $\mathcal{C}$ is a nonempty set of configurations, each $t \in \mathcal{T} : \mathcal{C} \rightarrow \mathcal{C}$ is a transition function between configurations, $c^0$ is an initialization function that maps an input sentence to an initial configuration, and $C_T \subseteq \mathcal{C}$ is a set of terminal configurations.

All systems we consider share a common tripartite representation for configurations: when we write $c = (\sigma, \beta, A)$ for some $c \in \mathcal{C}$, we are referring to a stack $\sigma$ of partially parsed subtrees; a buffer $\beta$ of unprocessed tokens and, optionally, at its beginning, a subtree with only left descendants; and a set $A$ of elements ($h, m$), each of which is an attachment (dependency arc) with head $h$ and modifier $m$. We write $m \leftarrow h$ to indicate that $m$ left-modifies $h$, and $h \leftarrow m$ to indicate that $m$ right-modifies $h$. For a sentence $w = w_1, ..., w_n$, the initial configuration is $c_0 = (\sigma_0, \beta_0, A_0)$, where $\sigma_0$ and $A_0$ are empty and $\beta_0 = [\text{ROOT} | w_1, ..., w_n]$: ROOT is a special node denoting the root of the parse tree (vertical bars are a notational convenience for indicating different parts of the buffer or stack; our convention is to depict the buffer first element leftmost, and to depict the stack first element rightmost). All terminal configurations have an empty buffer and a stack containing only ROOT.

Arc-Standard The arc-standard system (Nivre, 2004) is motivated by bottom-up parsing: each dependent has to be complete before being attached. The three transitions, shift (sh, move a token from the buffer to the stack), right-reduce (re$_\rightarrow$, reduce and attach a right modifier), and left-reduce (re$_\leftarrow$, reduce and attach a left modifier), are defined as:

$$\text{sh}[\sigma, b_0, \beta, A] = (\sigma, \beta, A)$$
$$\text{re}_{\rightarrow}[\sigma, s_1 | s_0, \beta, A] = (\sigma, s_1, \beta, A \cup \{(s_1, s_0)\})$$
$$\text{re}_{\leftarrow}[\sigma, s_1 | s_0, \beta, A] = (\sigma, s_0, \beta, A \cup \{(s_0, s_1)\})$$

Arc-Hybrid The arc-hybrid system (Yamada and Matsumoto, 2003; Gómez-Rodríguez et al., 2008; Kuhlmann et al., 2011) has the same definitions of sh and re$_\leftarrow$, as arc-standard, but forces transitions of $\sigma, b_0, \beta, A$ into $\sigma, b_0, \beta, A$. For simplicity, we only present unlabeled parsing here. See Shi et al. (2017) for labeled-parsing results.

Other presentations place ROOT at the end of the buffer or omit it entirely (Ballesteros and Nivre, 2013).
the collection of left modifiers before right modifiers via its $b_0$-modifier re$_-$ transition. This contrasts with arc-standard, where the attachment of left and right modifiers can be interleaved on the stack.

$$\text{sh}[(\sigma, b_0 | \beta, A)] = (\sigma | b_0, \beta, A)$$

$$\text{re}_-[(\sigma | s_1 | s_0, \beta, A)] = (\sigma | s_1, \beta, A \cup \{s_1, s_0\})$$

$$\text{re}_-[(\sigma | s_0, b_0 | \beta, A)] = (\sigma, b_0 | \beta, A \cup \{b_0, s_0\})$$

**Arc-Eager** In contrast to the former two systems, the arc-eager system (Nivre, 2003) makes attachments as early as possible — even if a modifier has not yet received all of its own modifiers. This behavior is accomplished by decomposing the right-reduce transition into two independent transitions, one making the attachment (ra) and one reducing the right-attached child (re).

$$\text{sh}[(\sigma, b_0 | \beta, A)] = (\sigma | b_0, \beta, A)$$

$$\text{re}_-[(\sigma | s_0, b_0 | \beta, A)] = (\sigma, b_0 | \beta, A \cup \{b_0, s_0\})$$

( precondition: $s_0$ not attached to any word)

$$\text{ra}[(\sigma | s_0, b_0 | \beta, A)] = (\sigma | s_0 | b_0, \beta, A \cup \{s_0, b_0\})$$

( precondition: $s_0$ has been attached to its head)

3.2 Deduction and Dynamic Programming

Kuhlmann et al. (2011) reformulate the three transition systems just discussed as deduction systems (Pereira and Warren, 1983; Shieber et al., 1995), wherein transitions serve as inference rules; these are given as the lefthand sides of the first three subfigures in Figure 1. For a given $w = w_1, \ldots, w_n$, assertions take the form $[i, j, k]$ (or, when applicable, a two-index shorthand to be discussed soon), meaning that there exists a sequence of transitions that, starting from a configuration wherein $\text{head}(s_0) = w_i$, results in an ending configuration wherein $\text{head}(s_0) = w_j$ and $\text{head}(b_0) = w_k$. If we define $w_0$ as ROOT and $w_{n+1}$ as an end-of-sentence marker, then the goal theorem can be stated as $[0, 0, n + 1]$.

For arc-standard, we depict an assertion $[i, h, k]$ as a subtree whose root (head) is the token at $h$. Assertions of the form $[i, i, k]$ play an important role for arc-hybrid and arc-eager, and we employ the special shorthand $[i, k]$ for them in Figure 1. In that figure, we also graphically depict such situations as two consecutive half-trees with roots $w_i$ and $w_k$, where all tokens between $i$ and $k$ are already attached. The superscript $b$ in an arc-eager assertion $[i^b, j]$ is an indicator variable for whether $w_i$ has been attached to its head ($b = 1$) or not ($b = 0$) after the transition sequence is applied.

Kuhlmann et al. (2011) show that all three deduction systems can be directly “tabularized” and dynamic programming (DP) can be applied, such that, ignoring for the moment the issue of incorporating complex features (we return to this later), time and space needs are low-order polynomial. Specifically, as the two-index shorthand $[i, j]$ suggests, arc-eager and arc-hybrid systems can be implemented to take $O(n^2)$ space and $O(n^3)$ time; the arc-standard system requires $O(n^3)$ space and $O(n^4)$ time (if one applies the so-called hook trick (Eisner and Satta, 1999)).

Since an $O(n^4)$ running time is not sufficiently practical even in the simple-feature case, in the remainder of this paper we consider only the arc-hybrid and arc-eager systems, not arc-standard.

4 Practical Optimal Algorithms Enabled By Our Minimal Feature Set

Until now, no one had suggested a set of positional features that was both information-rich enough for accurate parsing and small enough to obtain the $O(n^3)$ running-time promised above. Fortunately, our bi-LSTM-based $\{s_0, b_0\}$ feature set qualifies, and enables the fast optimal procedures described in this section.

4.1 Exact Decoding

Given an input sentence, a TBDP must choose among a potentially exponential number of corresponding transition sequences. We assume access to functions $f_t$ that score individual configurations, where these functions are indexed by the transition functions $t \in T$. For a fixed transition sequence $t = t_1, t_2, \ldots$, we use $c_t$ to denote the configuration that results after applying $t_t$.

Typically, for efficiency reasons, greedy left-to-right decoding is employed: the next transition $t^{*}_i$ out of $c_{t-1}$ is arg max$_i f_t(c_{t-1})$, so that past and future decisions are not taken into account. The score $F(t)$ for the transition sequence is induced by summing the relevant $f_t(c_{t-1})$ values.

However, our use of minimal feature sets enables direct computation of an argmax over the entire space of transition sequences, arg max$_t F(t)$, via dynamic programming, because our positions don’t rely on any information “outside” the deduction rule indices, thus eliminating the need for ad-
Figure 1: 1a-1c: Kuhlmann et al.’s inference rules for three transition systems, together with CKY-style visualizations of the local structures involved and, to their right, conditions for the rule to apply. 1d: the edge-factored graph-based parsing algorithm (Eisner and Satta, 1999) discussed in §5.
dential state-keeping.

We show how to integrate the scoring functions for the arc-eager system; the arc-hybrid system is handled similarly. The score-annotated rules are as follows:

$$\frac{[k^b, j]: v}{[j^0, j + 1]: 0} \text{(sh)} \quad \frac{[k^b, i]: v_1}{[k^b, j]: v_1 + v_2 + \Delta} \text{(re}_{-})$$

where \( \Delta = f_{sh}(\w_k, \w_i) + f_{re}_{-}(\w_i, \w_j) \) — abusing notation by referring to configurations by their features. The left-reduce rule says that we can first take the sequence of transitions asserted by \([k^b, i]\), which has a score of \(v_1\), and then a shift transition moving \(w_i\) from \(b_0\) to \(s_0\). This means that the initial condition for \([i^0, j]\) is met, so we can take the sequence of transitions asserted by \([i^0, j]\) — say it has score \(v_2\) — and finally a left-reduce transition to finish composing the larger transition sequence. Notice that the scores for \(sh\) and \(ra\) are 0, as the scoring of these transitions is accounted for by reduce rules elsewhere in the sequence.

4.2 Global Training

We employ large-margin training that considers training sentence \(w = w_1, \ldots, w_n\) with gold transition sequence \(t^{\text{gold}}\), our loss function is

$$\max_t \left(F(t) + \text{cost}(t^{\text{gold}}, t) - F(t^{\text{gold}})\right)$$

where \(\text{cost}(t^{\text{gold}}, t)\) is a custom margin for taking \(t\) instead of \(t^{\text{gold}}\) — specifically, the number of mis-attached nodes. Computing this max can again be done efficiently with a slight modification to the scoring of reduce transitions:

$$\frac{[k^b, i]: v_1}{[k^b, j]: v_1 + v_2 + \Delta'}$$

where \(\Delta' = \Delta + 1 (\text{head}(w_i) \neq w_j)\). This loss-augmented inference or cost-augmented decoding (Taskar et al., 2005; Smith, 2011) technique has previously been applied to graph-based parsing by Kiperwasser and Goldberg (2016a).

Efficiency Note The computation decomposes into two parts: scoring all feature combinations, and using DP to find a proof for the goal theorem in the deduction system. Time-complexity analysis is usually given in terms of the latter, but the former might have a large constant factor, such as \(10^4\) or worse for neural-network-based scoring functions. As a result, in practice, with a small \(n\), scoring with the feature set \(\{s_0, b_0\}\) \((O(n^2))\) can be as time-consuming as the decoding steps \((O(n^3))\) for the arc-hybrid and arc-eager systems.

5 Theoretical Connections

Our minimal feature set brings implementation of practical optimal algorithms to TBDPs, whereas previously only graph-based dependency parsers (GBDPs) — a radically different, non-incremental paradigm — enjoyed the ability to deploy them. Interestingly, for both the transition- and graph-based paradigms, the optimal algorithms build dependency trees bottom-up from local structures. It is thus natural to wonder if there are deeper, more formal connections between the two.

In previous work, Kuhlmann et al. (2011) related the arc-standard system to the classic CKY algorithm (Cocke, 1969; Kasami, 1965; Younger, 1967) in a manner clearly suggested by Figure 1a; CKY can be viewed as a very simple graph-based approach. Gómez-Rodríguez et al. (2008, 2011) formally prove that sequences of steps in the edge-factored GBDP (Eisner, 1996) can be used to emulate any individual step in the arc-hybrid system (Yamada and Matsumoto, 2003) and the Eisner and Satta (1999, Figure 1d) version. However, they did not draw an explicitly direct connection between Eisner and Satta (1999) and TBDPs.

Here, we provide an update to these previous findings, stated in terms of the expressiveness of scoring functions, considered as parameterization.

For the edge-factored GBDP, we write the score for an edge as \(f_G(h, m)\), where \(h\) is the head and \(m\) the modifier. A tree’s score is the sum of its edge scores. We say that a parameterized dependency parsing model A contains model B if for every instance of parameterization in model B, there exists an instance of model A such that the two models assign the same score to every parse tree. We claim:

Lemma 1. The arc-eager model presented in §4.1 contains the edge-factored model.

Proof Sketch. Consider a given edge-factored GBDP parameterized by \(f_G\). For any parse tree, every edge \(i^0\)-\(j\) involves two deduction rules, and their contribution to the score of the final proof is \(f_{sh}(\w_k, \w_{i}) + f_{re}_{-}(\w_{i}, \w_{j})\). We set \(f_{sh}(\w_k, \w_{i}) = 0\) and \(f_{re}_{-}(\w_{i}, \w_{j}) = f_G(\w_{i}, \w_{j})\). Similarly, for edges \(k^\ast i\) in the other direction, we set
\begin{table}[h]
\centering
\begin{tabular}{ |c|c|c|c|c|c| }
\hline
Model & Training & Features & PTB & & CTB \\
\hline
Arc-standard & Local & \{s_2, s_1, s_0, b_0\} & 93.95\pm0.12 & 52.29\pm0.66 & 88.01\pm0.26 & 36.87\pm0.53 \\
Arc-hybrid & Local & \{s_2, s_1, s_0, b_0\} & 93.89\pm0.10 & 50.82\pm0.75 & 87.87\pm0.17 & 35.47\pm0.48 \\
 & Local & \{s_0, b_0\} & 93.80\pm0.12 & 49.66\pm0.43 & 87.78\pm0.09 & 35.09\pm0.40 \\
 & Global & \{s_0, b_0\} & 94.43\pm0.08 & 53.03\pm0.71 & 88.38\pm0.11 & 36.59\pm0.27 \\
Arc-eager & Local & \{s_2, s_1, s_0, b_0\} & 93.80\pm0.12 & 49.66\pm0.43 & 87.49\pm0.20 & 33.15\pm0.72 \\
 & Local & \{s_0, b_0\} & 94.37\pm0.08 & 49.71\pm0.24 & 87.33\pm0.11 & 34.17\pm0.41 \\
 & Global & \{s_0, b_0\} & 94.53\pm0.05 & 53.77\pm0.46 & \textbf{88.62}\pm0.09 & \textbf{37.75}\pm0.87 \\
Edge-factored & Global & \{h, m\} & 94.50\pm0.13 & \textbf{53.86}\pm0.78 & 88.25\pm0.12 & 36.42\pm0.52 \\
\hline
\end{tabular}
\caption{Test set performance for different training regimes and feature sets. The models use the same decoders for testing and training. For each setting, the average and standard deviation across 5 runs with different random initializations are reported. Boldface: best (averaged) result per dataset/measure.}
\end{table}

\[ f_{ra}(w_k, w_t) = f_{C}(w_k, w_t) \text{ and } f_{re}(w_i, w_j) = 0. \]

The parameterization we arrive at emulates exactly the scoring model of \( f_{C} \).

We further claim that the arc-eager model is more expressive than not only the edge-factored GBDP, but also the arc-hybrid model in our paper.

**Lemma 2.** The arc-eager model contains the arc-hybrid model.

**Proof Sketch.** We leverage the fact that the arc-eager model divides the sh transition in the arc-hybrid model into two separate transitions, sh and ra. When we constrain the parameters \( f_{sh} = f_{ra} \) in the arc-eager model, the model hypothesis space becomes exactly the same as arc-hybrid’s.

The extra expressiveness of the arc-eager model comes from the scoring functions \( f_{sh} \) and \( f_{re} \) that capture structural contexts other than head-modifier relations. Unlike traditional higher-order graph-based parsing that directly models relations such as siblinghood (McDonald and Pereira, 2006) or grandparenthood (Carreras, 2007), however, the arguments in those two functions do not have any fixed type of structural interactions.

6 Experiments

**Data and Evaluation** We experimented with English and Chinese. For English, we used the Stanford Dependencies (de Marneffe and Manning, 2008) conversion (via the Stanford parser 3.3.0) of the Penn Treebank (Marcus et al., 1993, PTB). As is standard, we used §2-21 of the Wall Street Journal for training, §22 for development, and §23 for testing; POS tags were predicted using 10-way jackknifing with the Stanford max entropy tagger (Toutanova et al., 2003). For Chinese, we used the Penn Chinese Treebank 5.1 (Xue et al., 2002, CTB), with the same splits and head-finding rules for conversion to dependencies as Zhang and Clark (2008). We adopted the CTB’s gold-standard tokenization and POS tags. We report unlabeled attachment score (UAS) and sentence-level unlabeled exact match (UEM). Following prior work, all punctuation is excluded from evaluation. For each model, we initialized the network parameters with 5 different random seeds and report performance average and standard deviation.

**Implementation Details** Our model structures reproduce those of Kiperwasser and Goldberg (2016a). We use 2-layer bi-directional LSTMs with 256 hidden cell units. Inputs are concatenations of 28-dimensional randomly-initialized part-of-speech embeddings and 100-dimensional word vectors initialized from GloVe vectors (Pennington et al., 2014) (English) and pre-trained skipgram-model vectors (Mikolov et al., 2013) (Chinese). The concatenation of the bi-LSTM feature vectors is passed through a multi-layer perceptron (MLP) with 1 hidden layer which has 256 hidden units and activation function tanh. We set the dropout rate for the bi-LSTM (Gal and Ghahramani, 2016) and MLP (Srivastava et al., 2014) for each model according to development-set performance.6 All parameters except the word embed-
Main Results

We implement exact decoders for the arc-hybrid and arc-eager systems, and present the test performance of different model configurations in Table 2, comparing global models with local models. All models use the same decoder for testing as during the training process. Though no global decoder for the arc-standard system has been explored in this paper, its local models are listed for comparison. We also include an edge-factored graph-based model, which is conventionally trained globally. The edge-factored model scores bi-LSTM features for each head-modifier pair; a maximum spanning tree algorithm is used to find the tree with the highest sum of edge scores. For this model, we use Dozat and Manning’s (2017) biaffine scoring model, although in our case the model size is smaller.\(^7\)

Analogously to the dev-set results given in §2, on the test data, the minimal feature sets perform as well as larger ones in locally-trained models. And there exists a clear trend of global models outperforming local models for the two different transition systems on both datasets. This illustrates the effectiveness of exact decoding and global training. Of the three types of global models, the arc-eager arguably has the edge, an empirical finding resonating with our theoretical comparison of their model expressiveness.

Comparison with State-of-the-Art Models

Figure 2 compares our algorithms’ results with those of the state-of-the-art.\(^9\) Our models are competitive and an ensemble of 15 globally-trained models (5 models each for arc-eager DP, arc-hybrid DP and edge-factored) achieves 95.33 and 90.22 on PTB and CTB, respectively, reach-

---

\(^7\) See https://github.com/tzshi/dp-parser-emnlp17.

\(^8\) The same architecture and model size as other transition-based global models is used for fair comparison.

\(^9\) We exclude Choe and Charniak (2016), Kuncoro et al. (2017) and Liu and Zhang (2017), which convert constituent-based parses to dependency parses. They produce higher PTB UAS, but access more training information and do not directly apply to datasets without constituency annotation.
ing the highest reported UAS on the CTB dataset, and the second highest reported on the PTB dataset among dependency-based approaches.

7 Related Work Not Yet Mentioned

Approximate Optimal Decoding/Training Besides dynamic programming (Huang and Sagae, 2010; Kuhlmann et al., 2011), various other approaches have been proposed for approaching global training and exact decoding. Best-first and A* search (Klein and Manning, 2003; Sagae and Lavie, 2006; Sagae and Tsujii, 2007; Zhao et al., 2013; Thang et al., 2015; Lee et al., 2016) give optimality certificates when solutions are found, but have the same worst-case time complexity as the original search framework. Other common approaches to search a larger space at training or test time include beam search (Zhang and Clark, 2011), dynamic oracles (Goldberg and Nivre, 2012, 2013; Cross and Huang, 2016b) and error states (Vaswani and Sagae, 2016). Beam search records the $k$ best-scoring transition prefixes to delay local hard decisions, while the latter two leverage configurations deviating from the gold transition path during training to better simulate the test-time environment.

Neural Parsing Neural-network-based models are widely used in state-of-the-art dependency parsers (Henderson, 2003, 2004; Chen and Manning, 2014; Weiss et al., 2015; Andor et al., 2016; Dozat and Manning, 2017) because of their expressive representation power. Recently, Stern et al. (2017) have proposed minimal span-based features for constituency parsing.

Recurrent and recursive neural networks can be used to build representations that encode complete configuration information or the entire parse tree (Le and Zuidema, 2014; Dyer et al., 2015; Kiperwasser and Goldberg, 2016b), but these models cannot be readily combined with DP approaches, because their state spaces cannot be merged into smaller sets and thus remain exponentially large.

8 Concluding Remarks

In this paper, we have shown the following.

- The bi-LSTM-powered feature set $\{s_0, b_0\}$ is minimal yet highly effective for arc-hybrid and arc-eager transition-based parsing.
- Since DP algorithms for exact decoding (Huang and Sagae, 2010; Kuhlmann et al., 2011) have a run-time dependence on the number of positional features, using our mere two effective positional features results in a running time of $O(n^3)$, feasible for practice.

- Combining exact decoding with global training — which is also enabled by our minimal feature set — with an ensemble of parsers achieves 90.22 UAS on the Chinese Treebank and 95.33 UAS on the Penn Treebank: these are, to our knowledge, the best and second-best results to date on these data sets among “purely” dependency-based approaches.

There are many directions for further exploration. Two possibilities are to create even better training methods, and to find some way to extend our run-time improvements to other transition systems. It would also be interesting to further investigate relationships between graph-based and dependency-based parsing. In §5 we have mentioned important earlier work in this regard, and provided an update to those formal findings.

In our work, we have brought exact decoding, which was formerly the province solely of graph-based parsing, to the transition-based paradigm. We hope that the future will bring more inspiration from an integration of the two perspectives.

Acknowledgments: an author-reviewer success story We sincerely thank all the reviewers for their extraordinarily careful and helpful comments. Indeed, this paper originated as a short paper submission by TS&LL to ACL 2017, where an anonymous reviewer explained in the review comments how, among other things, the DP runtime could be improved from $O(n^4)$ to $O(n^3)$. In their author response, TS&LL invited the reviewer to co-author, suggesting that they ask the conference organizers to make the connection between anonymous reviewer and anonymous authors. All three of us are truly grateful to PC co-chair Regina Barzilay for implementing this idea, bringing us together!

We also thank Kai Sun for help with Chinese word vectors, and Xilun Chen, Yao Cheng, Dezong Deng, Juneki Hong, Jon Kleinberg, Ryan McDonald, Ashudeep Singh, and Kai Zhao for discussions and suggestions. TS and LL were supported in part by a Google focused research grant to Cornell University. LH was supported in part by NSF IIS-1656051, DARPA N66001-17-2-4030, and a Google Faculty Research Award.
References


