Fast(er) Exact Decoding and Global Training for Transition-Based Dependency Parsing via a Minimal Feature Set

Tianze Shi
Cornell University
tianze@cs.cornell.edu

Liang Huang
Oregon State University
liang.huang.sh@gmail.com

Lillian Lee
Cornell University
llee@cs.cornell.edu

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.

Publication venue: EMNLP 2017

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).[1]

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

[1] 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.

Sagae's and Kuhlmann et al.'s dynamic programming 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.[2]

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 subtrees 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.

Features	Arc-standard	Arc-hybrid	Arc-eager
$\{\overleftarrow{s}_2, \overleftarrow{s}_1, \overleftarrow{s}_0, \overleftarrow{b}_0\}$	$93.95_{\pm 0.12}$	$94.08_{\pm 0.13}$	$93.92_{\pm 0.04}$
$\{\overleftarrow{s}_1, \overleftarrow{s}_0, \overleftarrow{b}_0\}$	$94.13_{\pm 0.06}$	$94.08_{\pm 0.05}$	$93.91_{\pm 0.07}$
$\{\overleftarrow{s}_0, \overleftarrow{b}_0\}$	$54.47_{\pm 0.36}$	$94.03_{\pm 0.12}$	$93.92_{\pm 0.07}$
$\{\overleftarrow{b}_0\}$	$47.11_{\pm 0.44}$	$52.39_{\pm 0.23}$	$79.15_{\pm 0.06}$

Min positions	Arc-standard	Arc-hybrid	Arc-eager
K&G 2016a	-	4	-
C&H 2016a	3	-	-
our work	3	**2**	**2**

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.

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.

We employ the same model architecture as Kiperwasser and Goldberg (2016a). Specifically, we first use a bi-LSTM to encode an n-token sentence, treated as a sequence of per-token concatenations of word- and POS-tag embeddings, into a

[2] 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.

sequence of vectors $[\overleftrightarrow{w}_1, \ldots, \overleftrightarrow{w}_n]$, where each $\overleftrightarrow{w}_i$ 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 $\overleftrightarrow{s}_j$, defined as $\overleftrightarrow{w}_{i(s_j)}$ 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 $\overleftrightarrow{b}_j$, defined as $\overleftrightarrow{w}_{i(b_j)}$ 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 $\overleftrightarrow{s}_j$s and $\overleftrightarrow{b}_k$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 $\{\overleftrightarrow{s}_0, \overleftrightarrow{b}_0\}$ our *minimal* feature set with respect to arc-hybrid and arc-eager, and empirically confirm that Cross and Huang's $\{\overleftrightarrow{s}_0, \overleftrightarrow{s}_1, \overleftrightarrow{b}_0\}$ is minimal for arc-standard; see Table 1 for a summary.[3]

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 $\mathcal{S} = (C, T, c^s, C_\tau)$, where C is a nonempty set of configurations, each $t \in T : C \rightharpoonup C$ is a transition function between configurations, c^s is an initialization function that maps an input sentence to an initial configuration, and $C_\tau \subseteq C$ 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 $c \in C$, we are referring to a stack σ of partially parsed subtrees; a buffer β 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.[4] We write $m \overset{\frown}{\leftarrow} h$ to indicate that m left-modifies h, and $h \overset{\frown}{\rightarrow} m$ to indicate that m right-modifies h. For a sentence $w = w_1, ..., w_n$, the initial configuration is (σ_0, β_0, A_0), where σ_0 and A_0 are empty and $\beta_0 = [\texttt{ROOT}|w_1, ..., w_n]$; `ROOT` is a special node denoting the root of the parse tree[5] (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 ($\text{re}_{\overset{\frown}{\rightarrow}}$, reduce and attach a right modifier), and left-reduce ($\text{re}_{\overset{\frown}{\leftarrow}}$, reduce and attach a left modifier), are defined as:

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

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 $\text{re}_{\overset{\frown}{\rightarrow}}$ as arc-standard, but forces

[3] We tentatively conjecture that the following might explain the observed phenomena, but stress that we don't currently see a concrete way to test the following hypothesis. With $\{\overleftrightarrow{s}_0, \overleftrightarrow{b}_0\}$, in the arc-standard case, situations can arise where there are multiple possible transitions with missing information. In contrast, in the arc-hybrid case, there is only one possible transition with missing information (namely, $\text{re}_{\overset{\frown}{\rightarrow}}$, introduced in §3.1); perhaps $\overleftrightarrow{s}_1$ is therefore not so crucial for arc-hybrid in practice?

[4] For simplicity, we only present unlabeled parsing here. See Shi et al. (2017) for labeled-parsing results.

[5] 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 $\text{re}_{\curvearrowleft}$ transition. This contrasts with arc-standard, where the attachment of left and right modifiers can be interleaved on the stack.

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

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).

$$\text{sh}[(\sigma, b_0|\beta, A)] = (\sigma|b_0, \beta, A)$$
$$\text{re}_{\curvearrowleft}[(\sigma|s_0, b_0|\beta, A)] = (\sigma, b_0|\beta, A \cup \{(b_0, s_0)\})$$
(precondition: s_0 not attached to any word)
$$\text{ra}[(\sigma|s_0, b_0|\beta, A)] = (\sigma|s_0|b_0, \beta, A \cup \{(s_0, b_0)\})$$
$$\text{re}[(\sigma|s_0, \beta, A)] = (\sigma, \beta, A)$$
(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 $w = w_1, ..., w_n$, 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 $head(s_0) = w_i$, results in an ending configuration wherein $head(s_0) = w_j$ and $head(b_0) = w_k$. 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 $\{\overleftarrow{s}_0, \overleftarrow{b}_0\}$ 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 $\mathbf{t} = t_1, t_2, \ldots$, we use c_i to denote the configuration that results after applying t_i.

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

However, our use of minimal feature sets enables direct computation of an argmax over the entire space of transition sequences, $\arg\max_{\mathbf{t}} F(\mathbf{t})$, via dynamic programming, because our positions don't rely on any information "outside" the deduction rule indices, thus eliminating the need for ad-

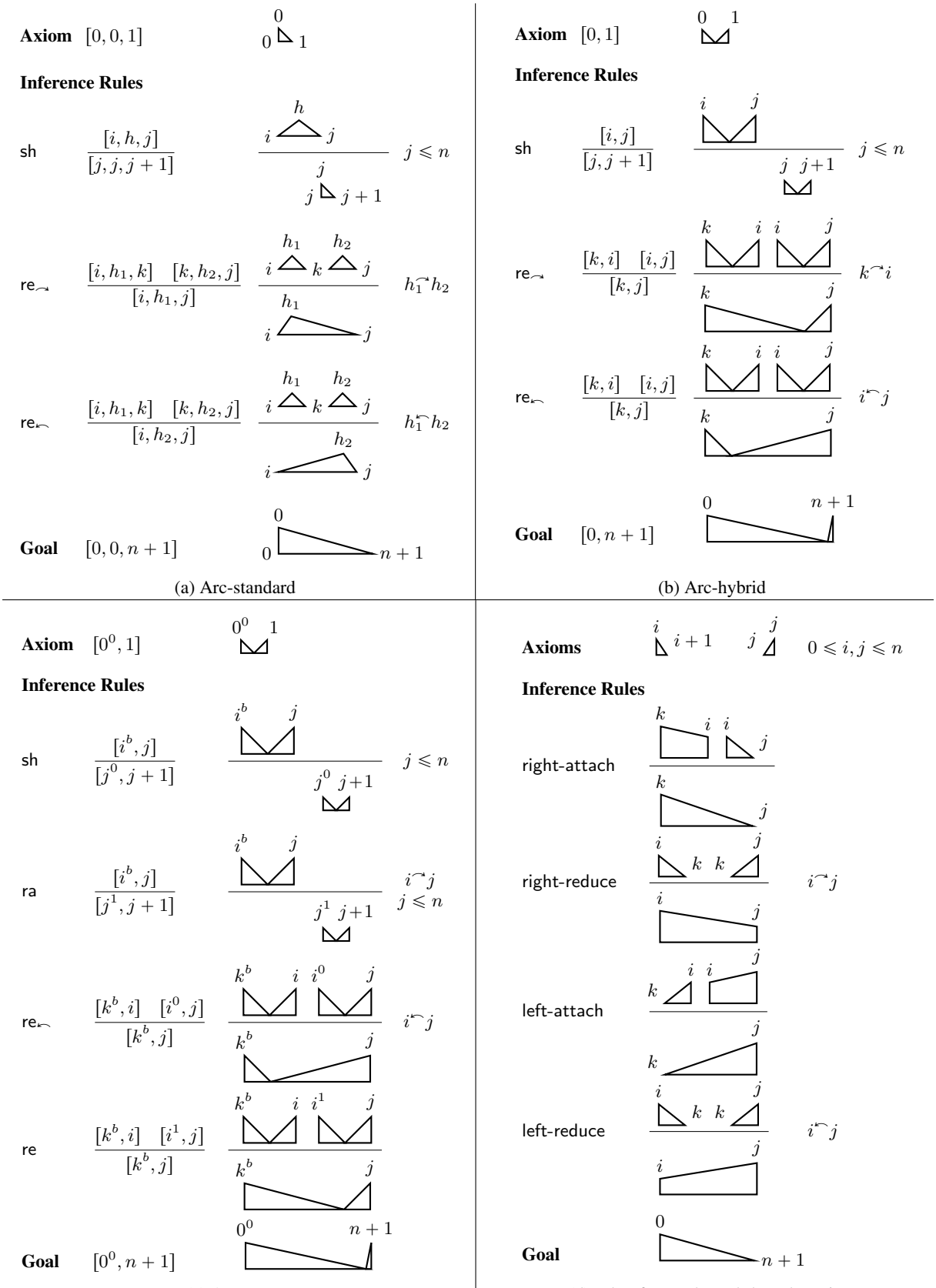

Figure 1: 1a-1c: Kuhlmann et al.'s inference rules for three transition systems, together with CKY-style visualizations of the local structures involved and, to their right, conditions for the rule to apply. 1d: the edge-factored graph-based parsing algorithm (Eisner and Satta, 1999) discussed in §5.

ditional 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:

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

where $\Delta = f_{\mathsf{sh}}(\overleftarrow{w}_k, \overleftarrow{w}_i) + f_{\mathsf{re}_{\curvearrowleft}}(\overleftarrow{w}_i, \overleftarrow{w}_j)$ — 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 each transition sequence globally. Formally, for a training sentence $w = w_1, \ldots, w_n$ with gold transition sequence $\mathbf{t}^{\text{gold}}$, our loss function is

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

where $cost(\mathbf{t}^{\text{gold}}, \mathbf{t})$ is a custom margin for taking $\mathbf{t}$ instead of $\mathbf{t}^{\text{gold}}$ — 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:

$$\frac{[k^b, i] : v_1 \quad [i^0, j] : v_2}{[k^b, j] : v_1 + v_2 + \Delta'}(\mathsf{re}_{\curvearrowleft})$$

where $\Delta' = \Delta + \mathbf{1}\left(head(w_i) \neq w_j\right)$. 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 $\{\overleftarrow{s}_0, \overleftarrow{b}_0\}$ ($O(n^2)$) 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 $f_G(\overleftarrow{h}, \overleftarrow{m})$, 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 $i \curvearrowleft j$ involves two deduction rules, and their contribution to the score of the final proof is $f_{\mathsf{sh}}(\overleftarrow{w}_k, \overleftarrow{w}_i) + f_{\mathsf{re}_{\curvearrowleft}}(\overleftarrow{w}_i, \overleftarrow{w}_j)$. We set $f_{\mathsf{sh}}(\overleftarrow{w}_k, \overleftarrow{w}_i) = 0$ and $f_{\mathsf{re}_{\curvearrowleft}}(\overleftarrow{w}_i, \overleftarrow{w}_j) = f_G(\overleftarrow{w}_j, \overleftarrow{w}_i)$. Similarly, for edges $k \curvearrowright i$ in the other direction, we set

Model	Training	Features	PTB		CTB	
			UAS (%)	UEM (%)	UAS (%)	UEM (%)
Arc-standard	Local	$\{\overleftarrow{s}_2, \overleftarrow{s}_1, \overleftarrow{s}_0, \overleftarrow{b}_0\}$	$93.95_{\pm 0.12}$	$52.29_{\pm 0.66}$	$88.01_{\pm 0.26}$	$36.87_{\pm 0.53}$
Arc-hybrid	Local	$\{\overleftarrow{s}_2, \overleftarrow{s}_1, \overleftarrow{s}_0, \overleftarrow{b}_0\}$	$93.89_{\pm 0.10}$	$50.82_{\pm 0.75}$	$87.87_{\pm 0.17}$	$35.47_{\pm 0.48}$
	Local	$\{\overleftarrow{s}_0, \overleftarrow{b}_0\}$	$93.80_{\pm 0.12}$	$49.66_{\pm 0.43}$	$87.78_{\pm 0.09}$	$35.09_{\pm 0.40}$
	Global	$\{\overleftarrow{s}_0, \overleftarrow{b}_0\}$	$94.43_{\pm 0.08}$	$53.03_{\pm 0.71}$	$88.38_{\pm 0.11}$	$36.59_{\pm 0.27}$
Arc-eager	Local	$\{\overleftarrow{s}_2, \overleftarrow{s}_1, \overleftarrow{s}_0, \overleftarrow{b}_0\}$	$93.80_{\pm 0.12}$	$49.66_{\pm 0.43}$	$87.49_{\pm 0.20}$	$33.15_{\pm 0.72}$
	Local	$\{\overleftarrow{s}_0, \overleftarrow{b}_0\}$	$93.77_{\pm 0.08}$	$49.71_{\pm 0.24}$	$87.33_{\pm 0.11}$	$34.17_{\pm 0.41}$
	Global	$\{\overleftarrow{s}_0, \overleftarrow{b}_0\}$	$\mathbf{94.53}_{\pm 0.05}$	$53.77_{\pm 0.46}$	$\mathbf{88.62}_{\pm 0.09}$	$\mathbf{37.75}_{\pm 0.87}$
Edge-factored	Global	$\{\overleftarrow{h}, \overleftarrow{m}\}$	$94.50_{\pm 0.13}$	$\mathbf{53.86}_{\pm 0.78}$	$88.25_{\pm 0.12}$	$36.42_{\pm 0.52}$

Table 2: 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.

$f_{\mathsf{ra}}(\overleftarrow{w}_k, \overleftarrow{w}_i) = f_G(\overleftarrow{w}_k, \overleftarrow{w}_i)$ and $f_{\mathsf{re}}(\overleftarrow{w}_i, \overleftarrow{w}_j) = 0$. The parameterization we arrive at emulates exactly the scoring model of f_G. □

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 $f_{\mathsf{sh}} = f_{\mathsf{ra}}$ 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 skip-gram-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-

[6]For bi-LSTM input and recurrent connections, we consider dropout rates in $\{0., 0.2\}$, and for MLP, $\{0., 0.4\}$.

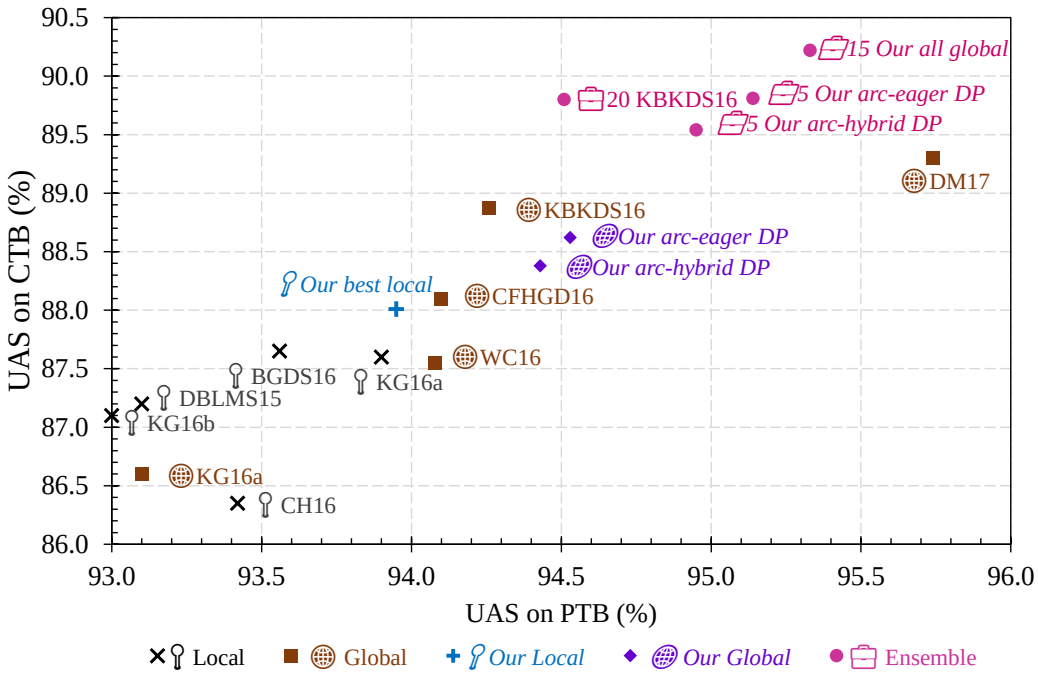

Figure 2: Comparing our UAS results with results from the literature. x-axis: PTB; y-axis: CTB. Most datapoint labels give author initials and publication year; citations are in the bibliography. Ensemble datapoints are annotated with ensemble size. Weiss et al. (2015) and Andor et al. (2016) achieve UAS of 94.26 and 94.61 on PTB with beam search, but did not report CTB results, and are therefore omitted.

dings are initialized uniformly (Glorot and Bengio, 2010). Approximately 1,000 tokens form a mini-batch for sub-gradient computation. We train each model for 20 epochs and perform model selection based on development UAS. The proposed structured loss function is optimized via Adam (Kingma and Ba, 2015). The neural network computation is based on the python interface to DyNet (Neubig et al., 2017), and the exact decoding algorithms are implemented in Cython.[7]

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.[8]

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 $\{\overrightarrow{s}_0, \overleftarrow{b}_0\}$ 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 run-time 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, Dezhong 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

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452, Berlin, Germany.

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A. Smith. 2016. Training with exploration improves a greedy stack LSTM parser. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 2005–2010.

Miguel Ballesteros and Joakim Nivre. 2013. Going to the roots of dependency parsing. *Computational Linguistics*, 39(1):5–13.

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 957–961.

Danqi Chen and Christopher D. Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 740–750, Doha, Qatar.

Hao Cheng, Hao Fang, Xiaodong He, Jianfeng Gao, and Li Deng. 2016. Bi-directional attention with agreement for dependency parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 2204–2214.

Do Kook Choe and Eugene Charniak. 2016. Parsing as language modeling. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 2331–2336, Austin, Texas.

John Cocke. 1969. Programming languages and their compilers: Preliminary notes. Technical report, Courant Institute of Mathematical Sciences, New York University.

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

James Cross and Liang Huang. 2016b. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1–11.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *Proceedings of the 5th International Conference on Learning Representations*.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343.

Jason Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics*, pages 340–345.

Jason Eisner and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 457–464.

Yarin Gal and Zoubin Ghahramani. 2016. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1019–1027.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, pages 249–256.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of the 24th International Conference on Computational Linguistics*, pages 959–976.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *Transactions of the Association for Computational Linguistics*, 1:403–414.

Carlos Gómez-Rodríguez, John Carroll, and David Weir. 2008. A deductive approach to dependency parsing. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technology*, pages 968–976.

Carlos Gómez-Rodríguez, John Carroll, and David Weir. 2011. Dependency parsing schemata and mildly non-projective dependency parsing. *Computational Linguistics*, 37(3):541–586.

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5-6):602–610.

James Henderson. 2003. Inducing history representations for broad coverage statistical parsing. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 24–31.

James Henderson. 2004. Discriminative training of a neural network statistical parser. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pages 95–102.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086.

Tadao Kasami. 1965. An efficient recognition and syntax-analysis algorithm for context-free languages. Technical report, Hawaii University Honolulu Department of Electrical Engineering.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the 4th International Conference on Learning Representations*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016a. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Eliyahu Kiperwasser and Yoav Goldberg. 2016b. Easy-first dependency parsing with hierarchical tree LSTMs. *Transactions of the Association for Computational Linguistics*, 4:445–461.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430.

Sandra Kübler, Ryan McDonald, and Joakim Nivre. 2009. *Dependency parsing*, volume 2 of *Synthesis Lectures on Human Language Technologies*. Morgan & Claypool Publishers.

Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 673–682.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. 2017. What do recurrent neural network grammars learn about syntax? In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1249–1258, Valencia, Spain.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, and Noah A. Smith. 2016. Distilling an ensemble of greedy dependency parsers into one MST parser. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1744–1753.

Phong Le and Willem Zuidema. 2014. The inside-outside recursive neural network model for dependency parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 729–739.

Kenton Lee, Mike Lewis, and Luke Zettlemoyer. 2016. Global neural CCG parsing with optimality guarantees. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 2366–2376.

Jiangming Liu and Yue Zhang. 2017. In-order transition-based constituent parsing. *Transactions of the Association for Computational Linguistics*. To appear.

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Marie-Catherine de Marneffe and Christopher D. Manning. 2008. Stanford typed dependencies manual. Technical report, Stanford University.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, pages 81–88.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017. DyNet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies*, pages 149–160.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543.

Fernando C. N. Pereira and David H. D. Warren. 1983. Parsing as deduction. In *Proceedings of the 21st Annual Meeting on Association for Computational Linguistics*, pages 137–144.

Kenji Sagae and Alon Lavie. 2006. A best-first probabilistic shift-reduce parser. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 691–698.

Kenji Sagae and Jun'ichi Tsujii. 2007. Dependency parsing and domain adaptation with LR models and parser ensembles. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1044–1050.

Tianze Shi, Felix G. Wu, Xilun Chen, and Yao Cheng. 2017. Combining global models for parsing Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 31–39, Vancouver, Canada.

Stuart M. Shieber, Yves Schabes, and Fernando C. N. Pereira. 1995. Principles and implementation of deductive parsing. *The Journal of Logic Programming*, 24(1):3–36.

Noah A. Smith. 2011. *Linguistic Structure Prediction*, volume 13 of *Synthesis Lectures on Human Language Technologies*. Morgan & Claypool Publishers.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Mitchell Stern, Jacob Andreas, and Dan Klein. 2017. A minimal span-based neural constituent parser. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. To appear.

Ben Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. 2005. Learning structured prediction models: A large margin approach. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 896–903.

Quang Le Thang, Hiroshi Noji, and Yusuke Miyao. 2015. Optimal shift-reduce constituent parsing with structured perceptron. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1534–1544.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 173–180.

Ashish Vaswani and Kenji Sagae. 2016. Efficient structured inference for transition-based parsing with neural networks and error states. *Transactions of the Association for Computational Linguistics*, 4:183–196.

Wenhui Wang and Baobao Chang. 2016. Graph-based dependency parsing with bidirectional LSTM. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2306–2315.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 323–333, Beijing, China.

Nianwen Xue, Fu-Dong Chiou, and Martha Palmer. 2002. Building a large-scale annotated Chinese corpus. In *Proceedings of the 19th International Conference on Computational Linguistics*.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the 8th International Workshop on Parsing Technologies*, pages 195–206.

Daniel H. Younger. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189 – 208.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 562–571.

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

Kai Zhao, James Cross, and Liang Huang. 2013. Optimal incremental parsing via best-first dynamic programming. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 758–768.