Shift-Reduce Constituency Parsing with Dynamic Programming and POS Tag Lattice

Haitao Mi[†]
[†]T.J. Watson Research Center
IBM
hmi@us.ibm.com

Liang Huang[‡†]
[‡]Queens College & Graduate Center
City University of New York
liang.huang.sh@gmail.com

Abstract

We present the first dynamic programming (DP) algorithm for shift-reduce constituency parsing, which extends the DP idea of Huang and Sagae (2010) to context-free grammars. To alleviate the propagation of errors from part-of-speech tagging, we also extend the parser to take a tag lattice instead of a fixed tag sequence. Experiments on both English and Chinese treebanks show that our DP parser significantly improves parsing quality over non-DP baselines, and achieves the best accuracies among empirical linear-time parsers.

1 Introduction

Incremental parsing has gained popularity in both dependency (Nivre, 2004; Zhang and Clark, 2008) and constituency parsing (Zhu et al., 2013; Wang and Xue, 2014). However, the greedy or beam search algorithms used in these parsers can only explore a tiny fraction of trees among exponentially many candidates. To alleviate this problem, Huang and Sagae (2010) propose a dynamic programming (DP) algorithm, reducing the search space to a polynomial size by merging equivalent states. This idea has been extended by Kuhlmann et al. (2011) and Cohen et al. (2011) to other dependency parsing paradigms.

In constituency parsing, however, DP has not yet been applied to incremental parsing, and the bigger search space in constituency parsing suggests a potentially even bigger advantage by DP. However, with unary rules and more-than-binary branchings, constituency parsing presents challenges not found in dependency parsing that must be addressed before applying DP. Thus, we first present an odd-even shift-reduce constituency parser which always finishes in same number of steps, eliminating the complicated asynchronicity issue in previous work (Zhu et al., 2013; Wang and Xue, 2014), and then develop dynamic programming on top of that. Secondly, to alleviate the error propagation from POS tagging, we also extends the algorithm to take a tagging sausage lattice as input, which is a compromise between pipeline and joint approaches (Hatori et al., 2011; Li et al., 2011; Wang and Xue, 2014).

Our DP parser achieves state-of-the-art performances on both Chinese and English treebanks (at 90.8% on PTB and 83.9% on CTB, the latter being the highest in literature).

2 Odd-Even Shift-Reduce CFG Parser

One major challenge in constituency parsing is unary rules. Unlike dependency parsing where shift-reduce always finishes in $2n-1$ steps, existing incremental constituency parsers (Zhu et al., 2013; Wang and Xue, 2014) reach the goal state (full parse tree) in different steps due to different number of unary rules. So we propose a new, synchronized, "odd-even" system to reach the goal in the same $4n-2$ steps. A *state* is notated $p = \langle S, Q \rangle$, where S is a stack of trees ..., s_1, s_0, and Q is a queue of word-tag pairs. At even steps (when step index is even) we can choose one of the three standard actions

- sh: shift the head of Q, a word-tag pair (t, w), onto S as a singleton tree $t(w)$;
- $\mathsf{re}^x_{\curvearrowleft}$: combine the top two trees on the stack and replace them with a new tree $x(s_1, s_0)$, x being the root nonterminal, headed on s_0;
- $\mathsf{re}^x_{\curvearrowright}$: similar to $\mathsf{re}^x_{\curvearrowleft}$ but headed on s_1;

and at odd steps we can choose two new actions:

Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the ACL, pages 1030–1035,
Denver, Colorado, May 31 – June 5, 2015.

$$\text{input}\ \ (t_1, w_1)\ ...\ (t_n, w_n)$$

$$\text{axiom}\ 0 : \langle \epsilon, (t_1, w_1)\ ...\ (t_n, w_n) \rangle : 0$$

$$\textsf{sh}\quad \frac{l : \langle S, (t,w)|Q \rangle : c}{l{+}1 : \langle S|t(w), Q \rangle : c{+}c_{\textsf{sh}}}\ l \text{ is even}$$

$$\textsf{re}^x_{\curvearrowleft}\quad \frac{l : \langle S|s_1|s_0, Q \rangle : c}{l{+}1 : \langle S|x(s_1, s_0), Q \rangle : c{+}c_{\textsf{re}^x_{\curvearrowleft}}}\ l \text{ is even}$$

$$\textsf{un}^x\quad \frac{l : \langle S|s_0, Q \rangle : c}{l{+}1 : \langle S|x(s_0), Q \rangle : c{+}c_{\textsf{un}^x}}\ l \text{ is odd}$$

$$\textsf{st}\quad \frac{l : \langle S|s_0, Q \rangle : c}{l{+}1 : \langle S|s_0, Q \rangle : c{+}c_{\textsf{st}}}\ l \text{ is odd}$$

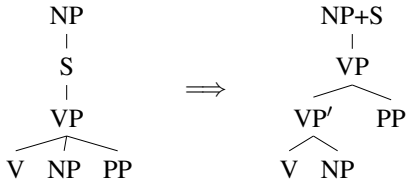
$$\text{goal}\ \ 2(2n-1) : \langle s_0, \epsilon \rangle : c$$

Figure 1: Shift-reduce system, omitting $\textsf{re}^x_{\curvearrowright}$. c is the model score, and $c_{\textsf{sh}}$, $c_{\textsf{re}^x_{\curvearrowleft}}$, etc. are the action scores.

- $\textsf{un}^x$: replace s_0 with a new tree $x(s_0)$ with x being the root nonterminal;
- $\textsf{st}$: no action.

Figure 1 shows the deductive system. Note that we alternate between standard shift-reduce actions in even steps and unary actions ($\textsf{un}^x$ or $\textsf{st}$) in odd steps, and the first action must be $\textsf{sh}$, followed by a $\textsf{un}^x$ or $\textsf{st}$, and followed by another $\textsf{sh}$. Continuing this procedure, we can always achieve the goal in $2(2n-1)$ steps.

In practice, we have larger than two-way rules and multi-level unary rules, so we binarize them and collapse multi-level unary rules into one level, for example,



Following Huang and Sagae (2010), we represent **feature templates** as functions $\mathbf{f}(\cdot, \cdot)$ on stack S and queue Q. Table 1 shows the 43 feature templates we use in this paper, all adopted from Zhu et al. (2013). They are combinations of the 32 **atomic features** $\tilde{\mathbf{f}}(S, Q)$ (e.g. s_0.t and s_0.c denote the head tag and

$$\textsf{sh}\quad \frac{l : \langle S, (t,w)|Q \rangle : (c, v)}{l{+}1 : \langle S|t(w), Q \rangle : (c{+}c_{\textsf{sh}}, 0)}\ l \text{ is even}$$

$$\textsf{re}^x_{\curvearrowleft}\quad \frac{\overset{\text{state } p\text{:}}{l' : \langle S'|s'_1, Q' \rangle : (c', v')}\quad \overset{\text{state } q\text{:}}{l : \langle S|s_1|s_0, Q \rangle : (c, v)}}{l{+}1 : \langle S'|x(s'_1, s_0), Q \rangle : (c'{+}v{+}\delta, v'{+}v{+}\delta)}$$
$$l \text{ and } l' \text{ are even}, p \in \pi(q)$$

$$\textsf{un}^x\ \frac{l : \langle S|s_0, Q \rangle : (c, v)}{l{+}1 : \langle S|x(s_0), Q \rangle : (c{+}c_{\textsf{un}^x}, v + c_{\textsf{un}^x})}\ l \text{ is odd}$$

Figure 2: DP shift-reduce, omitting $\textsf{re}^x_{\curvearrowright}$ and $\textsf{st}$. c and v are prefix and inside scores, and $\delta = c_{\textsf{sh}}(p) + c_{\textsf{re}^x_{\curvearrowleft}}(q)$. State equivalence is defined below in Section 3.

syntactic category of tree s_0, resp., and s_0.lc.w is the head word of its leftmost child).

3 Dynamic Programming

The key idea towards DP is the merging of equivalent states, after which the stacks are organized in a "graph-structured stack" (GSS)(Tomita, 1988). Following Huang and Sagae (2010), "equivalent states" $\sim$ in a same beam are defined by the atomic features $\tilde{\mathbf{f}}(S, Q)$ and the span of s_0:

$$\langle S, Q \rangle \sim \langle S', Q' \rangle$$
$$\Leftrightarrow \tilde{\mathbf{f}}(S, Q) = \tilde{\mathbf{f}}(S', Q') \text{ and } s_0.span = s'_0.span.$$

Similarly, for each state p, $\pi(p)$ is a set of **predictor states**, each of which can be combined with p in a $\textsf{re}^x_{\curvearrowleft}$ or $\textsf{re}^x_{\curvearrowright}$ action. For each action, we have different operations on $\pi(p)$. If a state p makes a $\textsf{sh}$ action and generates a state p', then $\pi(p') = \{p\}$. If two shifted states p' and p'' are equivalent, $p' \sim p''$, we merge $\pi(p')$ and $\pi(p'')$. If a state p makes a reduce ($\textsf{re}^x_{\curvearrowleft}$ or $\textsf{re}^x_{\curvearrowright}$) action, p tries to combine with every $p' \in \pi(p)$, and each combination generates a state r with $\pi(r) = \pi(p')$. If two reduced states are equivalent, we only keep one predictor states, as their predictor states are identical. If a state p fires an $\textsf{un}^x$ or a $\textsf{st}$ action resulting in a state u, we copy the predictor states $\pi(u) = \pi(p)$. Similar to reduce actions, if two resulting states after applying an $\textsf{un}^x$ or a $\textsf{st}$ action are equivalent, we only keep the best one with highest score (the recombined ones are only useful for searching k-best trees).

feature templates $\mathbf{f}(S, Q)$						
unigrams	s_0.t ∘ s_0.c	s_0.w ∘ s_0.c	s_1.t ∘ s_1.c	s_1.w ∘ s_1.c	s_2.t ∘ s_2.c	s_2.w ∘ s_2.c
	s_3.t ∘ s_3.c	q_0.w ∘ q_0.t	q_1.w ∘ q_1.t	q_2.w ∘ q_2.t	q_3.w ∘ q_3.t	s_0.lc.w ∘ s_0.lc.c
	s_0.rc.w ∘ s_0.rc.c	s_0.u.w ∘ s_0.u.c	s_1.lc.w ∘ s_1.lc.c	s_1.rc.w ∘ s_1.rc.c	s_1.u.w ∘ s_1.u.c	
bigrams	s_0.w ∘ s_1.w	s_0.w ∘ s_1.c	s_0.c ∘ s_1.w	s_0.c ∘ s_1.c	s_0.w ∘ q_0.w	s_0.w ∘ q_0.t
	s_0.c ∘ q_0.w	s_0.c ∘ q_0.t	q_0.w ∘ q_1.w	q_0.w ∘ q_1.t	q_0.t ∘ q_1.w	q_0.t ∘ q_1.t
	s_1.w ∘ q_0.w	s_1.w ∘ q_0.t	s_1.c ∘ q_0.w	s_1.c ∘ q_0.t		
	s_0.c ∘ s_0.lc.c ∘ s_0.rc.c ∘ s_1.c		s_0.c ∘ s_0.lc.c ∘ s_0.rc.c ∘ s_1.c			
trigrams	s_0.c ∘ s_1.c ∘ s_2.c		s_0.w ∘ s_1.c ∘ s_2.c		s_0.c ∘ s_1.w ∘ q_0.t	
	s_0.c ∘ s_1.c ∘ s_2.w		s_0.c ∘ s_1.c ∘ q_0.t		s_0.w ∘ s_1.c ∘ q_0.t	
	s_0.c ∘ s_1.w ∘ q_0.t		s_0.c ∘ s_1.c ∘ q_0.w			

Table 1: All feature templates (43 templates based on 32 atomic features), taken from Zhu et al. (2013). s_i.c, s_i.w and s_i.t denote the syntactic label, the head word, and the head tag of s_i. s_i.lc.w means the head word of the left child of s_i. s_i.u.w means the head word of the unary root s_i. q_i.w and q_i.t denote the word and the tag of q_i.

input $(T_1, w_1)...(T_n, w_n)$

axioms $0 : \langle \epsilon, (t, w_1)...(T_n, w_n)(\{\text{</s>}\}, \text{</s>})\rangle : 0, \forall\, t \in T_1$

$$\textsf{sh}\ \frac{l : \langle S,\ (t, w)|(T', w')|Q\rangle : (c, v)}{l{+}1 : \langle S|t(w),\ (t', w')|Q\rangle : (c{+}c_{\textsf{sh}}, 0)}\ \begin{array}{l} t' \in T', \\ l \text{ is even} \end{array}$$

Figure 3: Extended shift-reduce deductive system with tagging sausage lattice, only showing sh.

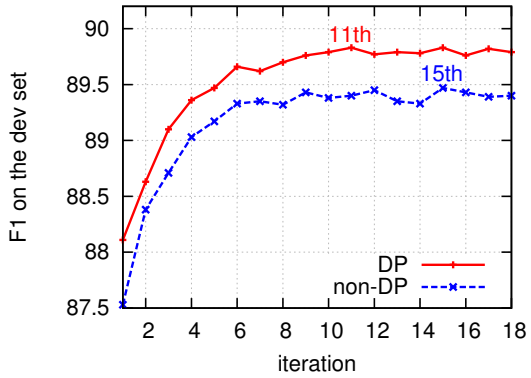


Figure 4: The learning curves of non-DP and DP parsers on the development set. DP achieves the best performance at 11th iteration with 89.8%, while non-DP gets its optimal iteration at 15th with a lower F1 89.5%.

In order to compute all the scores in GSS, for each state p, we calculate the prefix score, c, which is the total cost of the best action sequence from the initial state to the end of state p, and the inside score v, which is the score since the last shift (Figure 2).

The new mechanism beyond Huang and Sagae (2010) is the non-trivial dynamic programming treatment of unary actions ($\textsf{un}^x$ and st), which is not found in dependency parsing. Note that the score calculation is quite different from shift in the sense that unary actions are more like reduces.

4 Incorporating Tag Lattices

It is easy to extend our deductive system to take tagging sausage lattices as input. The key difference is that the tag t associated with each word in the input sequence becomes a set of tags T. Thus, in the sh action, we split the state with all the possible tags t' in the tagset T' for the **second** word on the queue. Figure 3 shows the deductive system, where we only change the sh action, input and axiom. For simplicity reasons we only present one word look ahead (we just need to know the tag of the first word on the queue), but in practice, we use a look ahead of 4 words ($q_0..q_3$, see Table 1), so each shift actually splits the tagset of the 5th word on the queue (q_4).

5 Experiments

We evaluate our parsers on both Penn English Treebank (PTB) and Chinese Treebank (CTB). For PTB, we use sections 02-21 as the training, section 24 as the dev set, and section 23 as the test. For CTB, we use the version of 5.1, articles 001-270 and 440-1151 as the training data, articles 301-325 as the dev set, and articles 271-300 as the test set.

Besides training with gold POS tags, we add k-best automatic tagging results to the training set using a MaxEnt model with ten-way jackknifing (Collins, 2000). And we automatically tag the dev and test sets with k-best tagging sequences us-

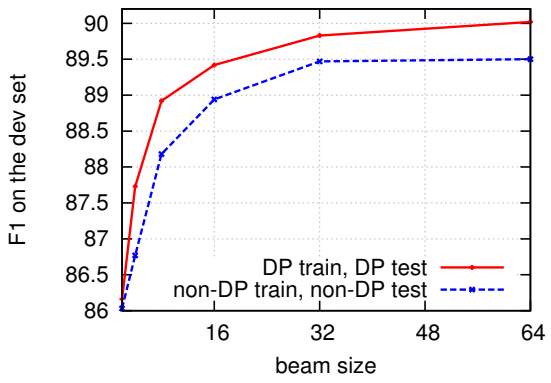


Figure 5: The F1 curves of non-DP and DP parsers (train and test consistently) on the dev set.

ing the MaxEnt POS tagger (at 97.1% accuracy on English, and 94.5% on Chinese) trained on the training set. We set k to 20 for English. And we run two sets of experiments, 1-best vs. 20-best, for Chinese to address the tagging issue. We train our parsers using "max-violation perceptron" (Huang et al., 2012) (which has been shown to converge much faster than "early-update" of Collins and Roark (2004)) with minibatch parallelization (Zhao and Huang, 2013) on the head-out binarized and unary-collapsed training set. We finally debinarize the trees to recover the collapsed unary rules.

We evaluate parser performance with EVALB including labeled precision (LP), labeled recall (LR), and bracketing F1. We use a beam size of 32, and pick the optimal iteration number based on the performances on the dev set.

Our baseline is the shift-reduce parser without state recombination (henceforth "non-DP"), and our dynamic programming parser (henceforth "DP") is the extension of the baseline.

5.1 Learning Curves and Search Quality

Figure 4 shows the learning curves on the PTB dev set. With a same beam width, DP parser achieves a better performance (89.8%, peaking at the 11th iteration) and converges faster than non-DP. Picking the optimal iterations for DP and non-DP models, we test each with various beam size, and plot the F1 curves in Figure 5. Again, DP is always better than non-DP, with 0.5% difference at beam of 64.

	LR	LP	F1	comp.
Collins (1999)	88.1	88.3	88.2	$O(n^5)$
Charniak (2000)	89.5	89.9	89.5	$O(n^5)$ †
Carreras (2008)	90.7	91.4	91.1	$O(n^4)$ ‡
Petrov (2007)	90.1	90.2	90.1	$O(n^3)$ †
Ratnaparkhi (1997)	86.3	87.5	86.9	
Sagae (2006)	87.8	88.1	87.9	$O(n)$
Zhu (2013)	90.2	90.7	90.4	
non-DP	90.3	90.4	90.3	$O(n)$
DP	**90.7**	**90.9**	**90.8**	

Table 2: Final Results on English (PTB) test set (sec23). †The empirical complexities for Charniak and Petrov are $O(n^{2.5})$ and $O(n^{2.4})$, resp., ‡but Carreras is exact $O(n^4)$.

	LR	LP	F1	POS
Charniak (2000)	79.6	82.1	80.8	-
Petrov (2007)	81.9	84.8	83.3	-
Zhu (2013)	82.1	84.3	83.2	-
Wang (2014) (1-best POS)	80.3	80.0	80.1	94.0
Wang (2014) (joint)	82.9	84.2	83.6	95.5
non-DP (1-best POS)	80.7	80.5	80.6	94.5
non-DP (20-best POS)	83.3	83.2	83.2	95.5
DP (20-best POS)	**83.6**	**84.2**	**83.9**	**95.6**

Table 3: Results on Chinese (CTB) 5.1 test set.

5.2 Final Results on English

Table 2 shows the final results on the PTB test set. The last column shows the empirical time complexity. Our baseline parser achieves a competitive score, which is higher than Berkeley even with a linear time complexity, and is comparable to Zhu et al. (2013). Our DP parser improves the F1 score by 0.5 points over the non-DP, and achieves the best F1 score among empirical linear-time parsers.

5.3 Sausage Lattice Parsing

To alleviate the propagation of errors from POS tagging, we run sausage lattice parsing on both Chinese and English, where Chinese tagging accuracy significantly lag behind English.

Table 3 shows the F1 score and POS tagging accuracy of all parsing models on the Chinese 5.1 test set. Our MaxEnt POS tagger achieves an accuracy of 94.5% on 1-best outputs, and an oracle score of 97.1% on 20-best results. The average number of

tags for each word in the 20-best list is 1.1.

The joint tagging and parsing approach of Wang and Xue (2014) improves the F1 score from 80.1% to 83.6% (see lines 4 and 5). We instead use sausage lattices, a much cheaper way. The non-DP (1-best POS) and non-DP (20-best POS) lines show the effectiveness of using sausage lattices (+1.1 for tagging and +2.6 for parsing). As Wang and Xue (2014) is a non-DP model, it is comparable to our non-DP results. With the help of 20-best tagging lattices, we achieve the same tagging accuracy at 95.5%, but still 0.4 worse on the F1 score than the joint model. It suggests that we need a larger k to catch up the gap. But our DP model boosts the performance further to the best score at 83.9% with a similar set of features.

The last two lines (non-DP and DP) in Table 2 show our English lattice parsing results. So we run another baseline with the non-DP English parser on 1-best POS tags, and the baseline achieves a tagging accuracy at 97.11 and an F1 score at 90.1. Comparing to the tagging accuracy (97.15) and F1 score (90.3) of our non-DP lattice parser, sausage lattice parsing doesn't help the tagging accuracy, but helps parsing a little by 0.2 points. The statistics show that 2 percent of POS tags in the lattice parsing result are different from the baseline, and those differences lead to a slight improvement on parsing.

6 Conclusions

In this paper, we present a dynamic programming algorithm based on graph-structured stack (GSS) for shift-reduce constituency parsing, and extend the algorithm to take tagging sausage lattices as input. Experiments on both English and Chinese treebanks show that our DP parser outperforms almost all other parsers except of Carreras et al. (2008), which runs in a much higher time complexity.

Acknowledgment

We thank the anonymous reviewers for comments. Haitao Mi is supported by DARPA HR0011-12-C-0015 (BOLT), and Liang Huang is supported by DARPA FA8750-13-2-0041 (DEFT), NSF IIS-1449278, and a Google Faculty Research Award. The views and findings in this paper are those of the authors and are not endorsed by the DARPA.

References

Xavier Carreras, Michael Collins, and Terry Koo. 2008. Tag, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proceedings of CoNLL 2008*.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of NAACL*.

Shay B. Cohen, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Exact inference for generative probabilistic non-projective dependency parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of ACL*.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proceedings of ICML*, pages 175–182.

Jun Hatori, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2011. Incremental joint pos tagging and dependency parsing in chinese. In *IJCNLP*.

Liang Huang and Kenji Sagae. 2010. Dynamic programming for linear-time incremental parsing. In *Proceedings of ACL 2010*.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proceedings of NAACL*.

Marco Kuhlmann, Carlos Gmez-Rodrguez, and Giorgio Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of ACL*.

Zhenghua Li, Min Zhang, Wanxiang Che, Ting Liu, Wenliang Chen, and Haizhou Li. 2011. Joint models for chinese pos tagging and dependency parsing. In *Proceedings of EMNLP*, pages 1180–1191.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Incremental Parsing: Bringing Engineering and Cognition Together. Workshop at ACL-2004*, Barcelona.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proceedings of HLT-NAACL*.

Adwait Ratnaparkhi. 1997. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of EMNLP*, pages 1–10.

Kenji Sagae and Alon Lavie. 2006. A best-first probabilistic shift-reduce parser. In *Proceedings of ACL (poster)*.

Masaru Tomita. 1988. Graph-structured stack and natural language parsing. In *Proceedings of the 26th annual meeting on Association for Computational Linguistics*, pages 249–257, Morristown, NJ, USA. Association for Computational Linguistics.

Zhiguo Wang and Nianwen Xue. 2014. Joint pos tagging and transition-based constituent parsing in chinese with non-local features. In *Proceedings of ACL*.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of EMNLP*.

Kai Zhao and Liang Huang. 2013. Minibatch and parallelization for online large margin structured learning. In *Proceedings of NAACL 2013*.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Proceedings of ACL 2013*.