Optimal Shift-Reduce Constituent Parsing with Structured Perceptron

Le Quang Thang
Hanoi University of Science
and Technology
{lelightwin@gmail.com}

Hiroshi Noji and **Yusuke Miyao**
National Institute of Informatics,
The Graduate University for
Advanced Studies
{noji,yusuke}@nii.ac.jp

Abstract

We present a constituent shift-reduce parser with a structured perceptron that finds the optimal parse in a practical runtime. The key ideas are new feature templates that facilitate state merging of dynamic programming and A* search. Our system achieves 91.1 F1 on a standard English experiment, a level which cannot be reached by other beam-based systems even with large beam sizes.[1]

1 Introduction

A parsing system comprises two components: a scoring model for a tree and a search algorithm. In shift-reduce parsing, the focus of most previous studies has been the former, typically by enriching feature templates, while the search quality has often been taken less seriously. For example, the current state-of-the-art parsers for constituency (Zhu et al., 2013; Wang and Xue, 2014) and dependency (Bohnet et al., 2013) both employ beam search with a constant beam size, which may suffer from severe search errors. This is contrary to ordinary PCFG parsing which, while it often uses some approximations, has nearly optimal quality (Petrov and Klein, 2007).

In this paper, we instead investigate the question of whether we can obtain a practical shift-reduce parser with state-of-the-art accuracy by focusing on optimal search quality like PCFG parsing. We base our system on best-first search for shift-reduce parsing formulated in Zhao et al. (2013), but it differs from their approach in two points. First, we focus on constituent parsing while they use dependency grammar. Second, and more crucially, they use a locally trained MaxEnt model, which is simple but not strong, while we explore a structured perceptron, the current state-of-the-art in shift-reduce parsing (Zhu et al., 2013).

As we will see, this model change makes search quite hard, which motivates us to invent new feature templates as well as to improve the search algorithm. In existing parsers, features are commonly exploited from the parsing history, such as the top k elements on the stack. However, such features are expensive in terms of search efficiency. Instead of relying on features primarily from the stack, our features mostly come from the span of the top few nodes, an idea inspired by the recent empirical success in CRF parsing (Hall et al., 2014). We show that these span features also fit quite well in the shift-reduce system and lead to state-of-the-art accuracy. We further improve search with new A* heuristics that make optimal search for shift-reduce parsers with a structured perceptron tractable for the first time.

The primary contribution of this paper is to demonstrate the effectiveness and the practicality of optimal search for shift-reduce parsing, especially when combined with appropriate features and efficient search. In English Penn Treebank experiments, our parser achieves an F1 score of 91.1 on test set at a speed of 13.6 sentences per second. This score is in excess of that of a beam-based system with larger beam size and same speed.

2 Background and Related Work

2.1 Shift-Reduce Constituent Parsing

We first introduce the shift-reduce algorithm for constituent structures. For space reasons, our exposition is rather informal; See Zhang and Clark (2009) for details. A shift-reduce parser parses a sentence through transitions between *states*, each of which consists of two data structures of a stack and a queue. The stack preserves intermediate parse results, while the queue saves unprocessed tokens. At each step, a parser selects an action,

[1]The open source software of our system is available at https://github.com/mynlp/optsr.

Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pages 1534–1544,
Beijing, China, July 26-31, 2015. ©2015 Association for Computational Linguistics

which changes the current state into the new one. For example, SHIFT pops the front word from the queue and pushes it onto the stack, while REDUCE(X) combines the top two elements on the stack into their parent.[2] For example, if the top two elements on the stack are DT and NN, REDUCE(NP) combines these by applying the CFG rule NP → DT NN.

Unary Action The actions above are essentially the same as those in shift-reduce dependency parsing (Nivre, 2008), but a special action for constituent parsing UNARY(X) complicates the system and search. For example, if the top element on the stack is NN, UNARY(NP) changes it to NP by applying the rule NP → NN. In particular, this causes inconsistency in the numbers of actions between derivations (Zhu et al., 2013), which makes it hard to apply the existing best first search for dependency grammar to our system. We revisit this problem in Section 3.1.

Model The model of a shift-reduce parser gives a score to each derivation, i.e., an action sequence $\mathbf{a} = (a_1, \cdots, a_{|\mathbf{a}|})$, in which each a_i is a shift or reduce action. Let $\mathbf{p} = (p_1, \cdots, p_{|\mathbf{a}|})$ be the sequence of states, where p_i is the state after applying a_i to p_{i-1}. p_0 is the initial state for input sentence $\mathbf{w}$. Then, the score for a derivation $\Phi(\mathbf{a})$ is calculated as the total score of every action:

$$\Phi(\mathbf{a}) = \sum_{1 \le i \le |\mathbf{a}|} \phi(a_i, p_{i-1}). \quad (1)$$

There are two well-known models, in which the crucial difference is in training criteria. The MaxEnt model is trained *locally* to select the correct action at each step. It assigns a probability for each action a_i as

$$P(a_i|p_{i-1}) \propto \exp(\theta^\intercal f(a_i, p_{i-1})), \quad (2)$$

where θ and $f(a, p)$ are weight and feature vectors, respectively. Note that the probability of an action sequence $\mathbf{a}$ under this model is the product of local probabilities, though we can cast the total score in summation form (1) by using the log of (2) as a local score $\phi(a_i, p_{i-1})$.

The structured perceptron is instead trained *globally* to select the correct action sequence given an input sentence. It does not use probability and the local score is just $\phi(a_i, p_{i-1}) = \theta^\intercal f(a_i, p_{i-1})$. In practice, this global model is much stronger than the local MaxEnt model. However, training this model without any approximation is hard, and the common practice is to rely on well-known heuristics such as an early update with beam search (Collins and Roark, 2004). We are not aware of any previous study that succeeded in training a structured perceptron for parsing without approximation. We will show how this becomes possible in Section 3.

[2]Many existing constituent parsers use two kinds of reduce actions for selecting the direction of its head child while we do not distinguish these two. In our English experiments, we found no ambiguity for head selection in our binarized grammar (See Section 4).

2.2 Previous Best-First Shift-Reduce Parsing

The basic idea behind best-first search (BFS) for shift-reduce parsing is assuming each parser state as a node on a graph and then searching for the *minimal cost* path from a start state (node) to the final state. This is the idea of Sagae and Lavie (2006), and it was later refined by Zhao et al. (2013). BFS gives a *priority* to each state, and a state with the highest priority (lowest cost) is always processed first. BFS guarantees that the first found goal is the best (*optimality*) if the *superiority* condition is satisfied: a state never has a lower cost than the costs of its previous states.

Though the found parse is guaranteed to be optimal, in practice, current BFS-based systems are not stronger than other systems with approximate search (Zhu et al., 2013; Wang and Xue, 2014) since all existing systems are based on the MaxEnt model. With this model, the speriority can easily be accomplished by using the negative log of (2), which is always positive and becomes smaller with higher probability. We focus instead on the structured perceptron, but achieving superiority with this model is not trivial. We resolve this problem in Section 3.1.

In addition to the mathematical convenience, the MaxEnt model itself helps search. Sagae and Lavie ascribe the empirical success of their BFS to the sparseness of the distribution over subsequent actions in the MaxEnt model. In other words, BFS is very efficient when only a few actions have dominant probabilities in each step, and the MaxEnt model facilitates this with its exponential operation (2). Unfortunately, this is not the case in our global structured perceptron because the score of each action is just the sum of the feature weights. Resolving this search difficulty is the central problem of this paper; we illustrate this problem in Section 4 and resolve it in Section 5.

2.3 Hypergraph Search of Zhao et al. (2013)

The worst time complexity of BFS in Sagae and Lavie (2006) is exponential. For dependency parsing, Zhao et al. (2013) reduce it to polynomial by converting the search graph into a hypergraph by using the state merging technique of Huang and Sagae (2010). This hypergraph search is the basis of our parser, so we will briefly review it here.

The algorithm is closely related to agenda-based best-first parsing algorithms for PCFGs (Klein and Manning, 2001; Pauls and Klein, 2009). As in those algorithms, it maintains two data structures: a chart C that preserves *processed* states as well as a priority queue (agenda) Q. The difference is in the basic items processed in C and Q. In PCFG parsing, they are *spans*. Each span abstracts many *derivations* on that span and the chart maps a span to the best (lowest cost) derivation found so far. In shift-reduce parsing, the basic items are not spans but *states*, i.e., partial representations of the stack.[3] We denote $p = \langle i, j, s_d...s_0 \rangle$ where s_i is the i-th top subtree on the stack and s_0 spans i to j. We extract features from $s_d...s_0$. Note that d is constant and a state usually does not contain full information about a derivation. In fact, it only keeps *atomic features*, the minimal information on the stack necessary to recover the full features and packs many derivations. The chart maps a state to the current best derivation. For example, if we extract features only from the root symbol of s_0, each state looks the same as a span of PCFGs.

Differently from the original shift-reduce algorithm, during this search, reduce actions are defined between two states p and q. The basic operation of the algorithm is to pop the best (top) state p from the queue, push it into the chart, and then enqueue every state that can be obtained by a reduce action between p and other states in the chart or a *shift* action from p. The left states $\mathcal{L}(p)$ and right states $\mathcal{R}(p)$ are important concepts. $\mathcal{L}(p)$ is a set of states in the chart, with which p can reduce from the right side. Formally,

$$\mathcal{L}(\langle i, j, s_d...s_0 \rangle) = \\ \{\langle h, i, s'_d...s'_0 \rangle | \forall k \in [1, d], f_k(s'_{k-1}) = f_k(s_k)\},$$

where $f_k(\cdot)$ returns atomic features on the k-th top node. See Figure 4 for how they look like in constituent parsing. $\mathcal{R}(p)$ is defined similarly; p can reduce $q \in \mathcal{R}(p)$ from the left side. When p is popped, it searches for every $\mathcal{L}(p)$ and $\mathcal{R}(p)$ in the chart and tries to expand the current derivation.

The priority for each state is a pair (c, v). c is the prefix cost that is the total cost to reach that state, while v is the inside cost, a cost to build the top node s_0. The top state in the queue has the lowest prefix cost, or the lowest inside cost if the two prefix costs are the same.

3 Best-First Shift-Reduce Constituent Parsing with Structured Perceptron

This section describes our basic parsing system, i.e., shift-reduce constituent parsing with BFS and the structured perceptron. We have to solve two problems. The first is how to achieve BFS with the structured perceptron, and the second is how to apply that BFS to constituent parsing. Interestingly, the solution to the first problem makes the second problem relatively trivial.

3.1 Superiority of Structured Perceptron

We must design each priority of a state to satisfy the superiority condition. $\phi(a_i, p_{i-1}) = \theta^\intercal f(a_i, p_{i-1})$ is the usual local *score* employed in structured perceptrons (Huang and Sagae, 2010) but we cannot use it as a local *cost* for two reasons. First, in our system, the best parse should have the lowest cost; it is opposite in the ordinary setting (Collins, 2002). We can resolve this conflict by changing the direction of structured perceptron training so that the best parse has the lowest score.[4] Second, each $\phi(a_i, p_{i-1})$ can take a negative value but the cost should always be positive. This is in contrast to the MaxEnt model in which the negative log probability is always positive. Our strategy is to add a constant offset δ to every local cost. If δ is large enough so that every score is positive, the superiority condition is satisfied.[5]

Unary Merging Though this technique solves the problem with the structured perceptron for a simpler shift-reduce system, say for dependency grammar, the existence of unary actions, as mentioned in Section 2.1, requires additional effort in order to apply it to constituent parsing. In particular, constituent parsing takes different numbers of

[3] Although Zhao et al. (2013) explained that the items in Q are derivations (not states), we can implement Q as a set of states by keeping backpointers in a starndard way.

[4] This is easily accomplished by inverting all signs of the update equations.

[5] To find this value, we train our system using beam search with several beam sizes, choosing the maximum value of the action score during training.

$$\textsc{Sh} \quad \frac{\overset{\text{state } p\text{:}}{\langle \_, j, s_d ... s_0 \rangle : (c, \_)}}{\langle j, j+1, s_{d-1} ... s_0 | t_j(w_j) \rangle : (c + c_{\mathsf{sh}}(p), c_{\mathsf{sh}}(p))} \; j < n$$

$$\textsc{Shu(X)} \quad \frac{\overset{\text{state } p\text{:}}{\langle \_, j, s_d ... s_0 \rangle : (c, \_)}}{\langle j, j+1, s_{d-1} ... s_0 | \mathrm{X}(t_j(w_j)) \rangle : (c + c_{\mathsf{shu(X)}}(p), c_{\mathsf{shu(X)}}(p))} \; j < n$$

$$\textsc{Re(X)} \quad \frac{\overset{\text{state } q\text{:}}{\langle k, i, s'_d ... s'_0 \rangle : (c', v')} \quad \overset{\text{state } p\text{:}}{\langle i, j, s_d ... s_0 \rangle : (c, v)}}{\langle k, j, s'_d ... s'_1 | \mathrm{X}(s'_0, s_0) \rangle : (c' + v + c_{\mathsf{re(X)}}(p), v' + v + c_{\mathsf{re(X)}}(p))} \; q \in \mathcal{L}(p)$$

$$\textsc{Reu(Y, X)} \quad \frac{\overset{\text{state } q\text{:}}{\langle k, i, s'_d ... s'_0 \rangle : (c', v')} \quad \overset{\text{state } p\text{:}}{\langle i, j, s_d ... s_0 \rangle : (c, v)}}{\langle k, j, s'_d ... s'_1 | \mathrm{Y}(\mathrm{X}(s'_0, s_0)) \rangle : (c' + v + c_{\mathsf{reu(Y, X)}}(p), v' + v + c_{\mathsf{reu(Y, X)}}(p))} \; q \in \mathcal{L}(p)$$

Figure 1: The deductive system of our best-first shift-reduce constituent parsing explaining how the prefix cost and inside cost are calculated. FIN is omitted. | on the stack means an append operation and $a(b)$ means a subtree $a \rightarrow b$. t_j is the POS tag of j-th token while w_j is the surface form. $c_a(p)$ is the cost for an action a of which features are extracted from p. Each $c_a(p)$ implicitly includes an offset δ.

actions for each derivation, which means that the scores of two final states may contain different offset values. The existing modification to alleviate this inconsistency (Zhu et al., 2013) cannot be applied here because it is designed for beam search.

We instead develop a new transition system, in which the number of actions to reach the final state is always $2n$ (n is the length of sentence). The basic idea is merging a unary action into each shift or reduce action. Our system uses five actions:

- SH: original shift action;
- SHU(X): shift a node, then immediately apply a unary rule to that node;
- RE(X): original reduce action;
- REU(Y, X): do reduce to X first, then immediately apply an unary rule Y $\rightarrow$ X to it;
- FIN: finish the process.

Though the system cannot perform consecutive unary actions, in practice it can generate any unary chains as long as those in the training corpus by collapsing a chain into one rule. We preprocess the corpus in this way along with binarization (See Section 4).

Note that this system is quite similar to the transition system for dependency parsing. The only changes are that we have several varieties of shift and reduce actions. This modification also makes it easy to apply an algorithm developed for dependency parsing to constituent parsing, such as dynamic programming with beam search (Huang and Sagae, 2010), which has not been applied into constituent parsing until quite recently (Mi and Huang, 2015) (See Section 7).

Algorithm 1 BFS for Constituent Parsing; Only differences from Zhao et al. (2013)

```
1: procedure SHIFT(x, Q)
2:     TRYADD(sh(x), Q)
3:     for y ∈ shu(x) do
4:         TRYADD(y, Q)
5: procedure REDUCE(A, B, Q)
6:     for (x, y) ∈ A × B do
7:         for z ∈ re(x, y) ∪ reu(x, y) do
8:             TRYADD(z, Q)
```

3.2 BFS with Dynamic Programming

Now applying BFS of Zhao et al. (2013) for dependency parsing into constituent parsing is not hard. Figure 1 shows the deductive system of dynamic programming, which is much similar to that in dependency parsing. One important change is that we include a cost for a shift (SH or SHU) action in the prefix cost in a shift step, not a reduce step as in Zhao et al. (2013), since it is unknown whether the top node s_0 of a state p is instantiated with SH or SHU. This modification keeps the correctness of the algorithm and has been employed in another system (Kuhlmann et al., 2011).

The algorithm is also slightly changed. We show only the difference from Zhao et al. (2013) (Algorithm 1) in Algorithm 1. $\mathsf{shu}(x)$ is a function which returns the set of states that can be arrived at by possible SHU rules applied to the state x. $\mathsf{re}(x, y)$ and $\mathsf{reu}(x, y)$ are similar, and they return the set of states arrived at through one of RE or REU actions. As a speed up, we can apply a lazy expansion technique (we do so in our experiment).

Model	F1	Speed (Sent./s.)
SP (reduced features)	88.9	0.8
ME (reduced features)	85.1	4.8
ME (full features)	86.3	2.5

Table 1: Results of BFS systems with dynamic programming for the Penn Treebank development set with different models and features. SP = the structured perceptron; ME = the MaxEnt.

Another difference is in training. The previous best-first shift-reduce parsers are all trained in the same way as a parser with greedy search since the model is local MaxEnt. In our case, we can use structured perceptron training with exact search (Collins, 2002); that is, at each iteration for each sentence, we find the current argmin derivation with BFS, then update the parameters if it differs from the gold derivation. Note that at the beginning of training, BFS is inefficient due to the initial flat parameters. We use a heuristic to speed up this process: For a few iterations (five, in our case), we train the model with beam search and an early update (Collins and Roark, 2004). We find that this approximation does not affect the performance, while it greatly reduces the training time.

4 Evaluation of Best-First Shift-Reduce Constituent Parsing

This section evaluates the empirical performance of our best-first constituent parser that we built in the previous section. As mentioned in Section 2.2, the previous empirical success of best-first shift-reduce parsers might be due to the sparsity property of the MaxEnt model, which may not hold true in the structured perceptron. We investigate the validity of this assumption by comparing two systems, a locally trained MaxEnt model and a globally trained structured perceptron.

Setting We follow the standard practice and train each model on section 2-21 of the WSJ Penn Treebank (Marcus et al., 1993), which is binarized using the algorithm in Zhang and Clark (2009) with the head rule of Collins (1999). We report the F1 scores for the development set of section 22. The Stanford POS tagger is used for part-of-speech tagging.[6] We used the EVALB program to evaluate parsing performance.[7] Every experiment reported here was performed on hardware equipped with an Intel Corei5 2.5GHz processor and 16GB of RAM.

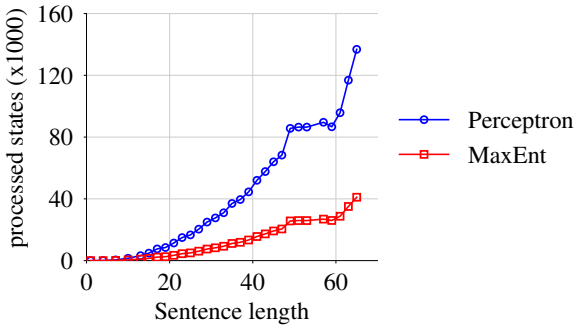


Figure 2: Comparison of the average number of the processed states of the structured perceptron with those of the MaxEnt model.

Feature We borrow the feature templates from Sagae and Lavie (2006). However, we found the full feature templates make training and decoding of the structured perceptron much slower, and instead developed simplified templates by removing some, e.g., that access to the child information on the second top node on the stack.[8]

Result Table 1 summarizes the results that indicate our assumption is true. The structured perceptron has the best score even though we restrict the features. However, its parsing speed is much slower than that of the local MaxEnt model. To see the difference in search behaviors between the two models, Figure 2 plots the number of processed (popped) states during search.

Discussion This result may seem somewhat depressing. We have devised a new method that enables optimal search for the structured perceptron, but it cannot handle even modestly large feature templates. As we will see below, the time complexity of the system depends on the used features. We have tried features from Sagae and Lavie (2006), but their features are no longer state-of-the-art. For example, Zhu et al. (2013) report higher scores by using beam search with much richer feature templates, though, as we have examined, it seems implausible to apply such features to our system. In the following, we find a practical solution for improving both parse accuracy and search efficiency in our system. We will see that our new features not only make BFS tractable, but also lead to comparable or even superior accuracy relative to the current mainstream features. When

[6] http://nlp.stanford.edu/software/tagger.shtml
[7] http://nlp.cs.nyu.edu/evalb
[8] The other features that we removed are features 9–14 defined in Figure 1 of Sagae and Lavie (2006).

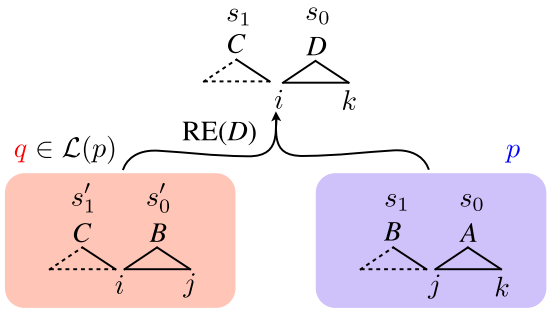


Figure 3: A snippet of the hypergraph for the system that simulates a simple PCFG. p is the popped state, which is being expanded with a state of its left states $\mathcal{L}(p)$ using a reduce rule.

it is combined with A* search, the speed reaches a practical level.

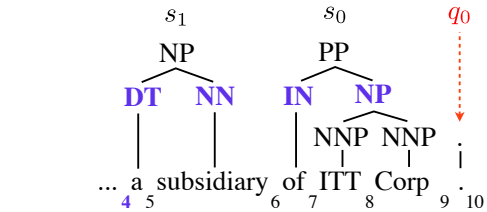
5 Improving Optimal Search Efficiency

5.1 Span Features

The worst time complexity of hypergraph search for shift-reduce parsing can be analyzed with the deduction rule of the reduce step. Figure 3 shows an example. In this case, the time complexity is $O(n^3 \cdot |G| \cdot |N|)$ since there are three indices (i, j, k) and four nonterminals (A, B, C, D), on which three comprise a rule. The extra factor $|N|$ compared with ordinary CKY parsing comes from the restriction that we extract features only from one state (Huang and Sagae, 2010).

Complexity increases when we add new atomic features to each state. For example, if we lexicalize this model by adding features that depend on the head indices of s_0 and/or s_1, it increases to $O(n^6 \cdot |G| \cdot |N|)$ since we have to maintain three head indices of *A*, *B*, and *C*. This is why Sagae and Lavie's features are too expensive for our system; they rely on head indices of s_0, s_1, s_2, s_3, the left and right children of s_0 and s_1, and so on, leading prohibitively huge complexity. Historically speaking, the success of shift-reduce approach in constituent parsing has been led by its success in dependency parsing (Nivre, 2008), in which the head is the primary element, and we suspect this is the reason why the current constituent shift-reduce parsers mainly rely on deeper stack elements and their heads.

The features we propose here are extracted from fundamentally different parts from these recent trends. Figure 4 explains how we extract atomic features from a state and Table 2 shows the full list of feature templates. Our system is unlexicalized;



s_0.bt = NN	s_0.ft = IN	s_0.lt = NNP	s_0.at = .
s_0.bw = subsidiary	s_0.fw = of	s_0.lw = Corp	s_0.aw = .
s_0.shape = xXX	s_0.c = PP	s_0.rule = PP → **IN NP**	
s_0.len = 3			

Figure 4: Atomic features of our system largely come from the span of a constituency. For each span (s_0 and s_1), we extract the surface form and POS tag of the preceding word (bw, bt), the first word (fw, ft), the last word (lw, lt), and the subsequent word (aw, at). shape is the same as that in Hall et al. (2014). Bold symbols are additional information from the system of Figure 3. The time complexity is $O(n^4 \cdot |G|^3 \cdot |N|)$.

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.c ∘ s_0.ft	s_0.c ∘ s_0.fw	s_0.c ∘ s_0.lt	s_0.c ∘ s_0.lw
s_0.c ∘ s_0.at	s_0.c ∘ s_0.aw	s_0.c ∘ s_0.ft ∘ s_0.lt	s_0.c ∘ s_0.ft ∘ s_0.lw
s_0.c ∘ s_0.fw ∘ s_0.lt	s_0.c ∘ s_0.fw ∘ s_0.lw	s_0.c ∘ s_0.len	s_0.c ∘ s_0.shape
s_0.rule	s_0.shape ∘ s_0.rule		
s_1.c ∘ s_1.ft	s_1.c ∘ s_1.fw	s_1.c ∘ s_1.lt	s_1.c ∘ s_1.lw
s_1.c ∘ s_1.bt	s_1.c ∘ s_1.bw	s_1.c ∘ s_1.ft ∘ s_1.lt	s_1.c ∘ s_1.ft ∘ s_1.lw
s_1.c ∘ s_1.fw ∘ s_1.lt	s_1.c ∘ s_1.fw ∘ s_1.lw	s_1.c ∘ s_1.len	s_1.c ∘ s_1.shape
s_1.rule	s_1.shape ∘ s_1.rule		
s_1.lw ∘ s_0.fw	s_0.ft ∘ s_1.lw	s_1.lt ∘ s_0.fw	s_1.lt ∘ s_0.ft
s_1.c ∘ s_0.fw	s_0.c ∘ s_1.fw	s_1.c ∘ s_0.lw	s_0.c ∘ s_1.lw
s_0.fw ∘ q_0.w	s_0.lw ∘ q_0.w	q_0.t ∘ s_0.fw	q_0.t ∘ s_0.lw
s_0.c ∘ q_0.w	s_0.c ∘ q_0.t	s_1.fw ∘ q_0.w	s_1.lw ∘ q_0.w
q_0.t ∘ s_1.fw	q_0.t ∘ s_1.lw	s_1.c ∘ q_0.w	s_1.c ∘ q_0.t
q_0.w ∘ q_1.w	q_0.t ∘ q_1.w	q_0.w ∘ q_1.t	q_0.t ∘ q_1.t
s_0.c ∘ s_1.c ∘ q_0.t	s_0.c ∘ s_1.c ∘ q_0.w	s_1.c ∘ q_0.t ∘ s_0.fw	s_1.c ∘ q_0.t ∘ s_0.lw
s_0.c ∘ q_0.t ∘ s_1.fw	s_0.c ∘ q_0.t ∘ s_1.fw		

Table 2: All feature templates in our span model. See Figure 4 for a description of each element. q_i is the i-th top token on the queue.

i.e., it does not use any head indices. This feature design is largely inspired by the recent empirical success of *span features* in CRF parsing (Hall et al., 2014). Their main finding is that the surface information on a subtree, such as the first or the last word of a span, has essentially the same amount of information as its head. For our system, such span features are much cheaper, so we expect they would facilitate our dynamic programming without sacrificing accuracy.

We customize their features for fitting in the shift-reduce framework. Unlike the usual setting of PCFG parsing, shift-reduce parsers receive a POS-tagged sentence as input, so we use both the POS tag and surface form for each word on the span. One difficult part is using features with an applied rule. We include this feature by memoriz-

ing the previously applied rule for each span (subtree). This is a bit costly, because it means we have to preserve labels of the left and right children for each node, which lead to an additional $|G|^2$ factor of complexity. However, we will see that this problem can be alleviated by our heuristic cost functions in A* search described below.

5.2 A* Search

We now explain our A* search, another key technique for speeding up our search. To our knowledge, this is the first work to successfully apply A* search to shift-reduce parsing.

A* parsing (Klein and Manning, 2003a) modifies the calculation of priority $\sigma(p_i)$ for state p_i. In BFS, it is basically the prefix cost, the sum of every local cost (Section 3.1), which we denote as β_{p_i}:

$$\beta_{p_i} = \sum_{1 \le j \le i} (\phi(a_j, p_{j-1}) + \delta).$$

In A* parsing, $\sigma(p_i) = \beta_{p_i} + h(p_i)$ where $h(p_i)$ is a heuristic cost. β_{p_i} corresponds to the Viterbi inside cost of PCFG parsing (Klein and Manning, 2003a) while $h(p_i)$ is the Viterbi outside cost, an approximation of the cost for the future best path (action sequence) from p_i.

$h(p_i)$ must be a lower bound of the *true* Viterbi outside cost. In PCFG parsing, this is often achieved with a technique called *projection*. Let G^* be a projected, or relaxed, grammar of the original G; then, a rule weight in the relaxed grammar w_{r^*} will become $w_{r^*} = \min_{r \in G: \pi(r) = r^*} w_r$, where $\pi(r)$ is a projection function which returns the set of rules that correspond to r in G^*.

In feature-based shift-reduce parsing, a rule weight corresponds to the sum of feature weights for an action a, that is, $\phi(a, p_i) = \theta^\intercal f(a, p_i)$. We calculate $h(p_i)$ with a relaxed feature function $\phi^*(a, p_i)$, which always returns a lower bound:

$$\phi^*(a, p_i) = \theta^{*\intercal} f(a, c_i) \le \theta^\intercal f(a, p_i) = \phi(a, p_i).$$

Note that we only have to modify the weight vector. If a relaxed weight satisfies $\theta^*(k) \le \theta(k)$ for all k, that projection is correct.

Our A* parsing is essentially hierarchical A* parsing (Pauls and Klein, 2009), and we calculate a heuristic cost $h(p)$ on the fly using another chart for the relaxed space when a new state p is pushed into the priority queue. Below we introduce two different projection methods, which are orthogonal and later combined hierarchically.

a	$\circ$	s_1.c	$\circ$	s_1.ft	θ	θ_{GP}	θ_{LF}
SH	$\circ$	VP	$\circ$	NN	10.53	-5.82	-5.82
SH	$\circ$	SBAR	$\circ$	NN	1.98	-5.82	-5.82
SH	$\circ$	NP	$\circ$	NN	**-5.82**	-5.82	-5.82
		...					
SH	$\circ$	VP	$\circ$	DT	3.25	1.12	-5.82
SH	$\circ$	SBAR	$\circ$	DT	**1.12**	1.12	-5.82
SH	$\circ$	NP	$\circ$	DT	1.98	1.12	-5.82
		...					

Table 3: Example of our feature projection. θ_{GP} is a weight vector with the GP, which collapses every c. θ_{LF} is with the LF, which collapses all elements in Table 4.

s_1.c	s_1.ft	s_1.fw	s_1.bt	s_1.bw
s_1.len	s_1.shape	s_1.rule	s_0.rule	

Table 4: List of feature elements ignored in the LF.

Grammar Projection (GP) Our first projection borrows the idea from the filter projection of Klein and Manning (2003a), in which the grammar symbols (nonterminals) are collapsed into a single label X. Our projection, however, does not collapse all the labels into X; instead, we utilize constituent labels in level 2 from Charniak et al. (2006), in which labels that tend to be head, such as S or VP are collapsed into HP and others are collapsed into MP. θ_{G} in Table 3 is an example of how feature weights are relaxed with this projection. Here we show each feature as a tuple including action name (a). Let π_{GP} be a feature projection function: e.g.,

$$((a \circ s_1.\text{c} \circ s_1.\text{ft}) = (\text{SH} \circ \text{VP} \circ \text{NN})) \\ \mapsto_{\pi_{\text{GP}}} ((a \circ s_1.\text{c} \circ s_1.\text{ft}) = (\text{SH} \circ \text{HP} \circ \text{NN})).$$

Formally, for k-th feature, the weight $\theta_{\text{GP}}(k)$ is determined by minimizing over the features collapsed by π_{GP}:

$$\theta_{\text{GP}}(k) = \min_{1 \le k' \le K: \pi_{\text{GP}}(g_{k'}) = g_k} \theta(k'),$$

where g_k is the value of the k-th feature.

Less-Feature Projection (LF) The basic idea of our second projection is to *ignore* some of the atomic features in a feature template so that we can reduce the time complexity for computing the heuristics. We apply this technique to the feature elements in Table 4. We can do so by not filling in the actual value in each feature template: e.g.,

$$((a \circ s_1.\text{c} \circ s_1.\text{ft}) = (\text{SH} \circ \text{VP} \circ \text{NN})) \\ \mapsto_{\pi_{\text{LF}}} ((a \circ s_1.\text{c} \circ s_1.\text{ft}) = (\text{SH} \circ s_1.\text{c} \circ s_1.\text{ft})).$$

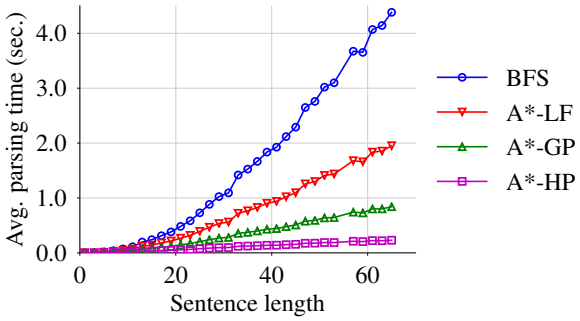


Figure 5: Comparison of parsing times between different A* heuristics.

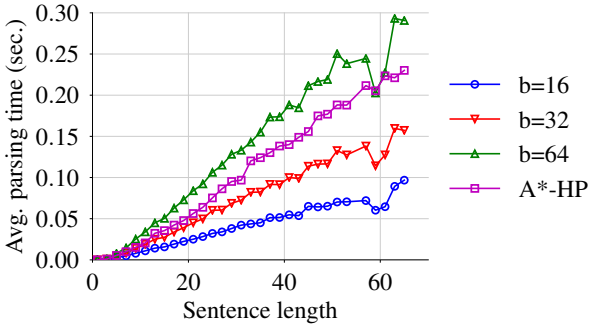


Figure 6: Comparison of parsing times between A* and beam search (with DP).

The elements in Table 4 are selected so that all bold elements in Figure 4 would be eliminated; the complexity is $O(n^3 \cdot |G| \cdot |N|)$. In practice, this is still expensive. However, we note that the effects of these two heuristics are complementary: The LF reduces complexity to a cubic time bound, while the GP greatly reduces the size of grammar $|G|$; We combine these two ideas below.

Hierarchical Projection (HP) The basic idea of this combined projection is to use the heuristics given by the GP to lead search of the LF. This is similar to the hierarchical A* for PCFGs with multilevel symbol refinements (Pauls and Klein, 2009). The difference is that their hierarchy is on the grammar symbols while our projection targets are features. When a state p is created, its heuristic score $h(p)$ is calculated with the LF, which requires search for the outside cost in the space of the LF, but its worst time complexity is cubic. The GP is used to guide this search. For each state p_{LF} in the space of the LF, the GP calculates the heuristic score. We will see that this combination works quite well in practice in the next section.

6 Experiment

We build our final system by combining the ideas in Section 5 and the system in Section 3. We also build beam-based systems with or without dynamic programming (DP) and with the ordinary or the new span features. All systems are trained with the structured perceptron. We use the early update for training beam-based systems.

Effect of A* heuristics Figure 5 shows the effects of A* heuristics. In terms of search quality, the LF is better; it prunes 92.5% of states compared to naive BFS, while the GP prunes 75%. However, the LF takes more time to calculate

Feaure	Z&C feature set			Span (this work)		
DP		✓			✓	
	F1	F1	Sent./s.	F1	F1	Sent./s.
b=16	89.1	90.1	34.6	88.6	89.9	31.9
b=32	89.6	89.9	20.0	89.3	90.2	17.0
b=64	89.7	90.2	10.6	89.6	90.2	9.1
A*	-	-	-	-	**90.7**	13.6
BFS	-	-	-	-	**90.7**	1.1

Table 5: Results for the Penn Treebank development set. Z&C = feature set of Zhang and Clark (2009). The speeds of non-DP and DP are the same, so we omit them from the comparison.

heuristics than the GP. The HP combines the advantages of both, achieving the best result.

Accuracy and Speed The F1 scores for the development set are summarized in Table 5. We can see that the systems with our new feature (span) perform surprisingly well, at a competitive level with the more expensive features of Zhang and Clark (2009) (Z&C). This is particularly true with DP; it sometimes outperforms Z&C, probably because our simple features facilitate state merging of DP, which expands search space. However, our main result that the system with optimal search gets a much higher score (90.7 F1) than beam-based systems with a larger beam size (90.2 F1) indicates that ordinary beam-based systems suffer from severe search errors even with the help of DP. Though our naive BFS is slow (1.12 sent./s.), A* search considerably improves parsing speed (13.6 sent./s.), and is faster than the beam-based system with a beam size of 64 (Figure 6).

Unary Merging We have not mentioned the effect of our unary merging (Section 3), but the result indicates it has almost the same effect as the previously proposed padding method (Zhu et al.,

Shift-reduce (closed)	LR	LP	F1	Sent./s.
Sagae (2005)†	86.0	86.1	86.0	3.7
Sagae (2006)†	88.1	87.8	87.9	2.2
Zhu (2013) (Z&C)	90.2	90.7	90.4	93.4
Span (b=64, DP)	90.2	90.6	90.4	8.4
Span (A*)	**90.9**	**91.2**	**91.1**	13.6
Other (closed)				
Berkeley (2007)	90.1	90.3	90.2	6.1
Stanford (2013) (RNN)	90.3	90.7	90.5	3.3
Hall (2014) (CRF)	89.0	89.5	89.3	0.7
External/Reranking				
Charniak (2005)	91.2	91.8	91.5	2.1
McClosky (2006)	92.2	92.6	92.4	1.2
Zhu (2013) +semi	91.1	91.5	91.3	47.6

Table 6: The final results for section 23 of the Penn Treebank. The systems with † are reported by authors running on different hardware. We divide baseline state-of-the-art systems into three categories: shift-reduce systems (Sagae and Lavie, 2005; Sagae and Lavie, 2006; Zhu et al., 2013), other chart-based systems (Petrov and Klein, 2007; Socher et al., 2013), and the systems with external semi supervised features or reranking (Charniak and Johnson, 2005; McClosky et al., 2006; Zhu et al., 2013).

2013). The score with the non-DP beam size = 16 and Z&C (89.1 F1) is the same as that reported in their paper (the features are the same).

Final Experiment Table 6 compares our parsing system with those of previous studies. When we look at closed settings, where no external resource other than the training Penn Treebank is used, our system outperforms all other systems including the Berkeley parser (Petrov and Klein, 2007) and the Stanford parser (Socher et al., 2013) in terms of F1. The parsing systems with external features or reranking outperform our system. However, it should be noted that our system could also be improved by external features. For example, the feature of type-level distributional similarity, such as Brown clustering (Brown et al., 1992), can be incorporated with our system without changing the theoretical runtime.

7 Related Work and Discussion

Though the framework is shift-reduce, we can notice that our system is strikingly similar to the CKY-based discriminative parser (Hall et al., 2014) because our features basically come from two nodes on the stack and their spans. From this viewpoint, it is interesting to see that our system outperforms theirs by a large margin (Figure 6). Identifying the source of this performance change is beyond the scope of this paper, but we believe this is an important question for future parsing research. For example, it is interesting to see whether there is any structural advantage for shift-reduce over CKY by comparing two systems with exactly the same feature set.

As shown in Section 4, the previous optimal parser on shift-reduce (Sagae and Lavie, 2006) was not so strong because of the locality of the model. Other optimal parsing systems are often based on relatively simple PCFGs, such as unlexicalized grammar (Klein and Manning, 2003b) or factored lexicalized grammar (Klein and Manning, 2003c) in which A* heuristics from the unlexicalized grammar guide search. However, those systems are not state-of-the-art probably due to the limited context captured with a simple PCFG. A recent trend has thus been extending the context of each rule (Petrov and Klein, 2007; Socher et al., 2013), but the resulting complex grammars make exact search intractable. In our system, the main source of information comes from spans as in CRF parsing. This is cheap yet strong, and leads to a fast and accurate parsing system with optimality.

Concurrently with this work, Mi and Huang (2015) have developed another dynamic programming for constituent shift-reduce parsing by keeping the step size for a sentence to $4n - 2$, instead of $2n$, with an un-unary (stay) action. Their final score is 90.8 F1 on WSJ. Though they only experiment with beam-search, it is possible to build BFS with their transition system as well.

8 Conclusions

To date, all practical shift-reduce parsers have relied on approximate search, which suffers from search errors but also allows to utilize *unlimited* features. The main result of this paper is to show another possibility of shift-reduce by proceeding in an opposite direction: By selecting features and improving search efficiency, a shift-reduce parser with provable search optimality is able to find very high quality parses in a practical runtime.

Acknowledgements

We would like to thank Katsuhiko Hayashi for answering our questions about dynamic programming on shift-reduce parsing.

References

Bernd Bohnet, Joakim Nivre, Igor M. Boguslavsky, Richárd Farkas, and Jan Hajič. 2013. Joint Morphological and Syntactic Analysis for Richly Inflected Languages. *Transactions of the Association for Computational Linguistics*, 1(Oct):429–440.

Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. 1992. Class-based N-gram Models of Natural Language. *Comput. Linguist.*, 18(4):467–479, December.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 173–180, Ann Arbor, Michigan, June. Association for Computational Linguistics.

Eugene Charniak, Mark Johnson, Micha Elsner, Joseph Austerweil, David Ellis, Isaac Haxton, Catherine Hill, R. Shrivaths, Jeremy Moore, Michael Pozar, and Theresa Vu. 2006. Multilevel Coarse-to-Fine PCFG Parsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 168–175, New York City, USA, June. Association for Computational Linguistics.

Michael Collins and Brian Roark. 2004. Incremental Parsing with the Perceptron Algorithm. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 111–118, Barcelona, Spain, July.

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

Michael Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 1–8. Association for Computational Linguistics, July.

David Hall, Greg Durrett, and Dan Klein. 2014. Less Grammar, More Features. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 228–237, Baltimore, Maryland, June. Association for Computational Linguistics.

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, Uppsala, Sweden, July. Association for Computational Linguistics.

Dan Klein and Christopher D. Manning. 2001. Parsing and Hypergraphs. In *Proceedings of the Seventh International Workshop on Parsing Technologies (IWPT-2001), 17-19 October 2001, Beijing, China*.

Dan Klein and Christopher D. Manning. 2003a. A* Parsing: Fast Exact Viterbi Parse Selection. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 40–47, Stroudsburg, PA, USA. Association for Computational Linguistics.

Dan Klein and Christopher D. Manning. 2003b. Accurate Unlexicalized Parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan, July. Association for Computational Linguistics.

Dan Klein and Christopher D. Manning. 2003c. Factored A^* Search for Models over Sequences and Trees. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 1246–1251.

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, Portland, Oregon, USA, June. Association for Computational Linguistics.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *COMPUTATIONAL LINGUISTICS*, 19(2):313–330.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Reranking and Self-Training for Parser Adaptation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 337–344, Sydney, Australia, July. Association for Computational Linguistics.

Haitao Mi and Liang Huang. 2015. Shift-Reduce Constituency Parsing with Dynamic Programming and POS Tag Lattice. In *Proceedings of the 2015 Conference on the North American Chapter of the Association for Computational Linguistics Human Language Technologies*, Denver, Colorado, May. Association for Computational Linguistics.

Joakim Nivre. 2008. Algorithms for Deterministic Incremental Dependency Parsing. *Computational Linguistics*, 34(4):513–553.

Adam Pauls and Dan Klein. 2009. Hierarchical Search for Parsing. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 557–565, Boulder, Colorado, June. Association for Computational Linguistics.

Slav Petrov and Dan Klein. 2007. Improved Inference for Unlexicalized Parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 404–411, Rochester, New York, April. Association for Computational Linguistics.

Kenji Sagae and Alon Lavie, 2005. *Proceedings of the Ninth International Workshop on Parsing Technology*, chapter A Classifier-Based Parser with Linear Run-Time Complexity, pages 125–132. Association for Computational Linguistics.

Kenji Sagae and Alon Lavie. 2006. A Best-First Probabilistic Shift-Reduce Parser. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 691–698, Sydney, Australia, July. Association for Computational Linguistics.

Richard Socher, John Bauer, Christopher D. Manning, and Ng Andrew Y. 2013. Parsing with Compositional Vector Grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 455–465, Sofia, Bulgaria, August. 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 the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 733–742, Baltimore, Maryland, June. Association for Computational Linguistics.

Yue Zhang and Stephen Clark. 2009. Transition-Based Parsing of the Chinese Treebank using a Global Discriminative Model. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 162–171, Paris, France, October. Association for Computational Linguistics.

Kai Zhao, James Cross, and Liang Huang. 2013. Optimal Incremental Parsing via Best-First Dynamic Programming. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 758–768, Seattle, Washington, USA, October. Association for Computational Linguistics.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and Accurate Shift-Reduce Constituent Parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 434–443, Sofia, Bulgaria, August. Association for Computational Linguistics.