Optimal Incremental Parsing via Best-First Dynamic Programming*

Kai Zhao[1] **James Cross**[1] **Liang Huang**[1,2]

[1]Graduate Center
City University of New York
365 Fifth Avenue, New York, NY 10016
{kzhao,jcross}@gc.cuny.edu

[2]Queens College
City University of New York
6530 Kissena Blvd, Queens, NY 11367
huang@cs.qc.cuny.edu

Abstract

We present the first provably optimal polynomial time dynamic programming (DP) algorithm for best-first shift-reduce parsing, which applies the DP idea of Huang and Sagae (2010) to the best-first parser of Sagae and Lavie (2006) in a non-trivial way, reducing the complexity of the latter from exponential to polynomial. We prove the correctness of our algorithm rigorously. Experiments confirm that DP leads to a significant speedup on a probablistic best-first shift-reduce parser, and makes exact search under such a model tractable for the first time.

1 Introduction

Best-first parsing, such as A* parsing, makes constituent parsing efficient, especially for bottom-up CKY style parsing (Caraballo and Charniak, 1998; Klein and Manning, 2003; Pauls and Klein, 2009). Traditional CKY parsing performs cubic time exact search over an exponentially large space. Best-first parsing significantly speeds up by always preferring to explore states with higher probabilities.

In terms of incremental parsing, Sagae and Lavie (2006) is the first work to extend best-first search to shift-reduce constituent parsing. Unlike other very fast greedy parsers that produce suboptimal results, this best-first parser still guarantees optimality but requires exponential time for very long sentences in the worst case, which is intractable in practice. Because it needs to explore an exponentially large space in the worst case, a bounded priority queue becomes necessary to ensure limited parsing time.

*This work is mainly supported by DARPA FA8750-13-2-0041 (DEFT), a Google Faculty Research Award, and a PSC-CUNY Award. In addition, we thank Kenji Sagae and the anonymous reviewers for their constructive comments.

On the other hand, Huang and Sagae (2010) explore the idea of dynamic programming, which is originated in bottom-up constituent parsing algorithms like Earley (1970), but in a beam-based non best-first parser. In each beam step, they enable state merging in a style similar to the dynamic programming in bottom-up constituent parsing, based on an equivalence relation defined upon feature values. Although in theory they successfully reduced the underlying deductive system to polynomial time complexity, their merging method is limited in that the state merging is only between two states in the same beam step. This significantly reduces the number of possible merges, because: 1) there are only a very limited number of states in the beam at the same time; 2) a lot of states in the beam with different steps cannot be merged.

We instead propose to combine the idea of dynamic programming with the best-first search framework, and apply it in shift-reduce dependency parsing. We merge states with the same features set globally to further reduce the number of possible states in the search graph. Thus, our DP best-first algorithm is significantly faster than non-DP best-first parsing, and, more importantly, it has a polynomial time complexity even in the worst case.

We make the following contributions:

- *theoretically*, we formally prove that our DP best-first parsing reaches optimality with polynomial time complexity. This is the first time that exact search under such a probabilistic model becomes tractable.

- more interestingly, we reveal that our dynamic programming over shift-reduce parsing is in parallel with the bottom-up parsers, except that we have an extra order constraint given by the shift action to enforce left to right generation of

$$\text{input} \quad w_0 \dots w_{n-1}$$

$$\text{axiom} \quad 0 : \langle 0,\ \epsilon \rangle : 0$$

$$\textsf{sh} \quad \frac{\ell : \langle j,\ S \rangle : c}{\ell + 1 : \langle j+1,\ S|w_j \rangle : c + sc_{\textsf{sh}}(j, S)} \ j < n$$

$$\textsf{re}_{\curvearrowleft} \quad \frac{\ell : \langle j,\ S|s_1|s_0 \rangle : c}{\ell + 1 : \langle j,\ S|s_1{}^{\curvearrowleft}s_0 \rangle : c + sc_{\textsf{re}_{\curvearrowleft}}(j, S|s_1|s_0)}$$

$$\textsf{re}_{\curvearrowright} \quad \frac{\ell : \langle j,\ S|s_1|s_0 \rangle : c}{\ell + 1 : \langle j,\ S|s_1{}^{\curvearrowright}s_0 \rangle : c + sc_{\textsf{re}_{\curvearrowright}}(j, S|s_1|s_0)}$$

Figure 1: Deductive system of basic non-DP shift-reduce parsing. Here ℓ is the step index (for beam search), S is the stack, c is the score of the precedent, and $sc_a(x)$ is the score of action a from derivation x. See Figure 2 for the DP version.

partial trees, which is analogous to Earley.

- *practically*, our DP best-first parser is only ∼2 times slower than a pure greedy parser, but is guaranteed to reach optimality. In particular, it is ∼20 times faster than a non-DP best-first parser. With inexact search of bounded priority queue size, DP best-first search can reach optimality with a significantly smaller priority queue size bound, compared to non-DP best-first parser.

Our system is based on a MaxEnt model to meet the requirement from best-first search. We observe that this locally trained model is not as strong as global models like structured perceptron. With that being said, our algorithm shows its own merits in both theory and practice. To find a better model for best-first search would be an interesting topic for future work.

2 Shift-Reduce and Best-First Parsing

In this section we review the basics of shift-reduce parsing, beam search, and the best-first shift-reduce parsing algorithm of Sagae and Lavie (2006).

2.1 Shift-Reduce Parsing and Beam Search

Due to space constraints we will assume some basic familiarity with shift-reduce parsing; see Nivre (2008) for details. Basically, shift-reduce parsing (Aho and Ullman, 1972) performs a left-to-right scan of the input sentence, and at each step, chooses either to *shift* the next word onto the stack, or to *reduce*, i.e., combine the top two trees on stack, either with left as the root or right as the root. This scheme is often called "arc-standard" in the literature (Nivre, 2008), and is the basis of several state-of-the-art parsers, e.g. Huang and Sagae (2010). See Figure 1 for the deductive system of shift-reduce dependency parsing.

To improve on strictly greedy search, shift-reduce parsing is often enhanced with beam search (Zhang and Clark, 2008), where b derivations develop in parallel. At each step we extend the derivations in the current beam by applying each of the three actions, and then choose the best b resulting derivations for the next step.

2.2 Best-First Shift-Reduce Parsing

Sagae and Lavie (2006) present the parsing problem as a search problem over a DAG, in which each parser derivation is denoted as a node, and an edge from node x to node y exists if and only if the corresponding derivation y can be generated from derivation x by applying one action.

The best-first parsing algorithm is an application of the Dijkstra algorithm over the DAG above, where the score of each derivation is the priority. Dijkstra algorithm requires the priority to satisfy the *superiority* property, which means a descendant derivation should never have a higher score than its ancestors. This requirement can be easily satisfied if we use a generative scoring model like PCFG. However, in practice we use a MaxEnt model. And we use the negative log probability as the score to satisfy the superiority:

$$x \prec y \Leftrightarrow x.score < y.score,$$

where the order $x \prec y$ means derivation x has a higher priority than y.[1]

The vanilla best-first parsing algorithm inherits the optimality directly from Dijkstra algorithm. However, it explores exponentially many derivations to reach the goal configuration in the worst case. We propose a new method that has polynomial time complexity even in the worst case.

[1]For simplicity we ignore the case when two derivations have the same score. In practice we can choose either one of the two derivations when they have the same score.

3 Dynamic Programming for Best-First Shift-Reduce Parsing

3.1 Dynamic Programming Notations

The key innovation of this paper is to extend best-first parsing with the "state-merging" method of dynamic programming described in Huang and Sagae (2010). We start with describing a parsing configuration as a non-DP *derivation*:

$$\langle i, j, ...s_2 s_1 s_0 \rangle,$$

where $...s_2 s_1 s_0$ is the stack of partial trees, $[i..j]$ is the span of the top tree s_0, and $s_1 s_2...$ are the remainder of the trees on the stack.

The notation $\mathbf{f}_k(s_k)$ is used to indicate the features used by the parser from the tree s_k on the stack. Note that the parser only extracts features from the top $d+1$ trees on the stack.

Following Huang and Sagae (2010), $\widetilde{\mathbf{f}}(x)$ of a derivation x is called *atomic features*, defined as the *smallest set* of features s.t.

$$\begin{aligned} &\widetilde{\mathbf{f}}(i, j, ...s_2 s_1 s_0) = \widetilde{\mathbf{f}}(i, j, ...s'_2 s'_1 s'_0) \\ \Leftrightarrow\ &\mathbf{f}_k(s_k) = \mathbf{f}_k(s'_k), \forall k \in [0, d]. \end{aligned}$$

The atomic feature function $\widetilde{\mathbf{f}}(\cdot)$ defines an equivalence relation $\sim$ in the space of derivations $\mathcal{D}$:

$$\begin{aligned} &\langle i, j, ...s_2 s_1 s_0 \rangle \sim \langle i, j, ...s'_2 s'_1 s'_0 \rangle \\ \Leftrightarrow\ &\widetilde{\mathbf{f}}(i, j, ...s_2 s_1 s_0) = \widetilde{\mathbf{f}}(i, j, ...s'_2 s'_1 s'_0) \end{aligned}$$

This implies that any derivations with the same atomic features are in the same *equivalence class*, and their behaviors are similar in shift and reduce. We call each equivalence class a DP *state*. More formally we define the space of all states $\mathcal{S}$ as:

$$\mathcal{S} \triangleq \mathcal{D}/\sim.$$

Since only the top $d+1$ trees on the stack are used in atomic features, we only need to remember the necessary information and write the state as:

$$\langle i, j, s_d...s_0 \rangle.$$

We denote a derivation x's state as $[x]_\sim$. In the rest of this paper, we always denote derivations with letters x, y, and z, and denote states with letters p, q, and r.

The deductive system for dynamic programming best-first parsing is adapted from Huang and Sagae (2010). (See the left of Figure 2.) The difference is that we do not distinguish the step index of a state.

This deductive system describes transitions between states. However, in practice we use one state's best derivation found so far to represent the state. For each state p, we calculate the prefix score, $p.pre$, which is the score of the derivation to reach this state, and the inside score, $p.ins$, which is the score of p's top tree $p.s_0$. In addition we denote the shift score of state p as $p.sh \triangleq sc_{\mathsf{sh}}(p)$, and the reduce score of state p as $p.re \triangleq sc_{\mathsf{re}}(p)$. Similarly we have the prefix score, inside score, shift score, and reduce score for a derivation.

With this deductive system we extend the concept of reducible states with the following definitions:

The set of all states with which a state p can legally reduce from the right is denoted $\mathcal{L}(p)$, or *left states*. (see Figure 3 (a)) We call any state $q \in \mathcal{L}(p)$ a *left state* of p. Thus each element of this set would have the following form:

$$\begin{aligned} \mathcal{L}(\langle i, j, s_d...s_0 \rangle) \triangleq \{ &\langle h, i, s'_d...s'_0 \rangle \mid \\ &\mathbf{f}_k(s'_{k-1}) = \mathbf{f}_k(s_k),\ \forall k \in [1, d] \} \end{aligned} \quad (1)$$

in which the span of the "left" state's top tree ends where that of the "right" state's top tree begins, and $\mathbf{f}_k(s_k) = \mathbf{f}_k(s'_{k-1})$ for all $k \in [1, d]$.

Similarly, the set of all states with which a state p can legally reduce from the left is denoted $\mathcal{R}(p)$, or *right states*. (see Figure 3 (a)) For two states p, q,

$$p \in \mathcal{L}(q) \Leftrightarrow q \in \mathcal{R}(p)$$

3.2 Algorithm 1

We constrain the searching time with a polynomial bound by transforming the original search graph with exponentially many derivations into a graph with polynomial number of states.

In Algorithm 1, we maintain a chart C and a priority queue Q, both of which are based on hash tables.

Chart C can be formally defined as a function mapping from the space of states to the space of derivations:

$$C : \mathcal{S} \to \mathcal{D}.$$

In practice, we use the atomic features $\widetilde{\mathbf{f}}(p)$ as the signature of state p, since all derivations in the same state share the same atomic features.

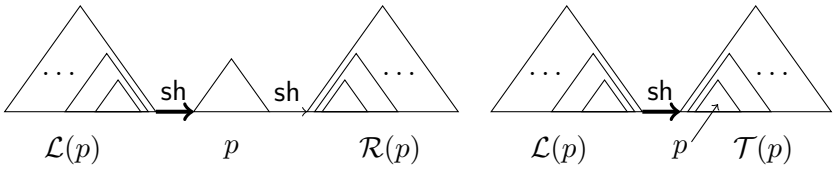
$$\textsf{sh} \quad \frac{\overset{\text{state } p\text{:}}{\langle \_, j, s_d ... s_0 \rangle : (c, \_)}}{\langle j, j+1, s_{d-1} ... s_0, w_j \rangle : (c + \xi, 0)} \; j < n$$

$$\textsf{re}_{\curvearrowleft} \frac{\overset{\text{state } q\text{:}}{\langle k, i, s'_d ... s'_0 \rangle : (c', v')} \quad \overset{\text{state } p\text{:}}{\langle i, j, s_d ... s_0 \rangle : (\_, v)}}{\langle k, j, s'_d ... s'_1, s'_0 \frown s_0 \rangle : (c' + v + \delta,\ v' + v + \delta)} \; q \in \mathcal{L}(p)$$

$$\textsc{Pred} \quad \frac{\langle \_, j, A \to \alpha . B \beta \rangle : (c, \_)}{\langle j, j, B \to . \gamma \rangle : (c + s,\ s)} \; (B \to \gamma) \in G$$

$$\textsc{Comp} \frac{\langle k, i, A \to \alpha . B \beta \rangle : (c', v') \quad \langle i, j, B \rangle : (\_, v)}{\langle k, j, A \to \alpha B . \beta \rangle : (c' + v,\ v' + v)}$$

Figure 2: Deductive systems for dynamic programming shift-reduce parsing (Huang and Sagae, 2010) (left, omitting $\textsf{re}_{\curvearrowright}$ case), compared to weighted Earley parsing (Stolcke, 1995) (right). Here $\xi = sc_{\textsf{sh}}(p)$, $\delta = sc_{\textsf{sh}}(q) + sc_{\textsf{re}_{\curvearrowleft}}(p)$, $s = sc(B \to \gamma)$, G is the set of CFG rules, $\langle i, j, B \rangle$ is a surrogate for any $\langle i, j, B \to \gamma . \rangle$, and $\_$ is a wildcard that matches anything.



(a) $\mathcal{L}(p)$ and $\mathcal{R}(p)$ (b) $\mathcal{T}(p) = \mathcal{R}(\mathcal{L}(p))$

Figure 3: Illustrations of left states $\mathcal{L}(p)$, right states $\mathcal{R}(p)$, and left corner states $\mathcal{T}(p)$. (a) Left states $\mathcal{L}(p)$ is the set of states that can be reduced with p so that $p.s_0$ will be the right child of the top tree of the result state. Right states $\mathcal{R}(p)$ is the set of states that can be reduced with p so that $p.s_0$ will be the left child of the top tree of the result state. (b) Left corner states $\mathcal{T}(p)$ is the set of states that have the same reducibility as shifted state p, i.e., $\forall p' \in \mathcal{L}(p)$, we have $\forall q \in \mathcal{T}(p), q \in \mathcal{R}(p')$. In both (a) and (b), thick sh arrow means shifts from multiple states; thin sh arrow means shift from a single state.

We use $C[p]$ to retrieve the derivation in C that is associated with state p. We sometimes abuse this notation to say $C[x]$ to retrieve the derivation associated with signature $\widetilde{\mathbf{f}}(x)$ for derivation x. This is fine since we know derivation x's state immediately from the signature. We say state $p \in C$ if $\widetilde{\mathbf{f}}(p)$ is associated with some derivation in C. A derivation $x \in C$ if $C[x] = x$. Chart C supports operation PUSH, denoted as $C[x] \leftarrow x$, which associate a signature $\widetilde{\mathbf{f}}(x)$ with derivation x.

Priority queue Q is defined similarly as C, except that it supports the operation POP that pops the highest priority item.

Following Stolcke (1995) and Nederhof (2003), we use the prefix score and the inside score as the priority in Q:

$$x \prec y \Leftrightarrow x.pre < y.pre \text{ or } (x.pre = y.pre \text{ and } x.ins < y.ins), \quad (2)$$

Note that, for simplicity, we again ignore the special case when two derivations have the same prefix score and inside score. In practice for this case we can pick either one of them. This will not affect the correctness of our optimality proof in Section 5.1.

In the DP best-first parsing algorithm, once a derivation x is popped from the priority queue Q, as usual we try to expand it with shift and reduce. Note that both left and right reduces are between the derivation x of state $p = [x]_\sim$ and an in-chart derivation y of left state $q = [y]_\sim \in \mathcal{L}(p)$ (Line 10 of Algorithm 1), as shown in the deductive system (Figure 2). We call this kind of reduction *left expansion*.

We further expand derivation x of state p with some in-chart derivation z of state r s.t. $p \in \mathcal{L}(r)$, i.e., $r \in \mathcal{R}(p)$ as in Figure 3 (a). (see Line 11 of Algorithm 1.) Derivation z is in the chart because it is the descendant of some other derivation that has been explored before x. We call this kind of reduction *right expansion*.

Our reduction with $\mathcal{L}$ and $\mathcal{R}$ is inspired by Nederhof (2003) and Knuth (1977) algorithm, which will be discussed in Section 4.

Algorithm 1 Best-First DP Shift-Reduce Parsing.

Let $\mathcal{L}_C(x) \triangleq C[\mathcal{L}([x]_\sim)]$ be in-chart derivations of $[x]_\sim$'s left states

Let $\mathcal{R}_C(x) \triangleq C[\mathcal{R}(p)]$ be in-chart derivations of $[x]_\sim$'s right states

1: **function** PARSE($w_0 \dots w_{n-1}$)
2: $C \leftarrow \emptyset$ ▷ empty chart
3: $Q \leftarrow \{\text{INIT}\}$ ▷ initial priority queue
4: **while** $Q \neq \emptyset$ **do**
5: $x \leftarrow \text{POP}(Q)$
6: **if** GOAL(x) **then return** x ▷ found best parse
7: **if** $[x]_\sim \notin C$ **then**
8: $C[x] \leftarrow x$ ▷ add x to chart
9: SHIFT(x, Q)
10: REDUCE($\mathcal{L}_C(x), \{x\}, Q$) ▷ left expansion
11: REDUCE($\{x\}, \mathcal{R}_C(x), Q$) ▷ right expansion
12: **procedure** SHIFT(x, Q)
13: TRYADD($\mathsf{sh}(x)$, Q) ▷ shift
14: **procedure** REDUCE(A, B, Q)
15: **for** $(x, y) \in A \times B$ **do** ▷ try all possible pairs
16: TRYADD($\mathsf{re}_{\curvearrowleft}(x, y)$, Q) ▷ left reduce
17: TRYADD($\mathsf{re}_{\curvearrowright}(x, y)$, Q) ▷ right reduce
18: **function** TRYADD(x, Q)
19: **if** $[x]_\sim \notin Q$ **or** $x \prec Q[x]$ **then**
20: $Q[x] \leftarrow x$ ▷ insert x into Q or update $Q[x]$

3.3 Algorithm 2: Lazy Expansion

We further improve DP best-first parsing with lazy expansion.

In Algorithm 2 we only show the parts that are different from Algorithm 1.

Assume a *shifted* derivation x of state p is a direct descendant from derivation x' of state p', then $p \in \mathcal{R}(p')$, and we have:

$$\forall ys.t.\ [y]_\sim = q \in \text{REDUCE}(\{p'\}, \mathcal{R}(p')), x \prec y$$

which is proved in Section 5.1.

More formally, we can conclude that

$$\forall ys.t.\ [y]_\sim = q \in \text{REDUCE}(\mathcal{L}(p), \mathcal{T}(p)), x \prec y$$

where $\mathcal{T}(p)$ is the *left corner states* of shifted state p, defined as

$$\mathcal{T}(\langle i, i{+}1, s_d ... s_0 \rangle) \triangleq \{\langle i, h, s'_d ... s'_0 \rangle \mid \mathbf{f}_k(s'_k) = \mathbf{f}_k(s_k),\ \forall k \in [1, d]\}$$

which represents the set of all states that have the same reducibility as a shifted state p. In other words,

$$\mathcal{T}(p) = \mathcal{R}(\mathcal{L}(p)),$$

Algorithm 2 Lazy Expansion of Algorithm 1.

Let $\mathcal{T}_C(x) \triangleq C[\mathcal{T}([x]_\sim)]$ be in-chart derivations of $[x]_\sim$'s left-corner states

1: **function** PARSE($w_0 \dots w_{n-1}$)
2: $C \leftarrow \emptyset$ ▷ empty chart
3: $Q \leftarrow \{\text{INIT}\}$ ▷ initial priority queue
4: **while** $Q \neq \emptyset$ **do**
5: $x \leftarrow \text{POP}(Q)$
6: **if** GOAL(x) **then return** x ▷ found best parse
7: **if** $[x]_\sim \notin C$ **then**
8: $C[x] \leftarrow x$ ▷ add x to chart
9: SHIFT(x, Q)
10: REDUCE($x.\mathit{lefts}, \{x\}, Q$) ▷ left expansion
11: **else if** $x.\mathit{action}$ **is** sh **then**
12: REDUCE($x.\mathit{lefts}, \mathcal{T}_C(x), Q$) ▷ right expan.
13: **procedure** SHIFT(x, Q)
14: $y \leftarrow \mathsf{sh}(x)$
15: $y.\mathit{lefts} \leftarrow \{x\}$ ▷ initialize *lefts*
16: TRYADD(y, Q)
17: **function** TRYADD(x, Q)
18: **if** $[x]_\sim \in Q$ **then**
19: **if** $x.\mathit{action}$ **is** sh **then** ▷ maintain *lefts*
20: $y \leftarrow Q[x]$
21: **if** $x \prec y$ **then** $Q[x] \leftarrow x$
22: $Q[x].\mathit{lefts} \leftarrow y.\mathit{lefts} \cup x.\mathit{lefts}$
23: **else if** $x \prec Q[x]$ **then**
24: $Q[x] \leftarrow x$
25: **else** ▷ $x \notin Q$
26: $Q[x] \leftarrow x$

which is illustrated in Figure 3 (a). Intuitively, $\mathcal{T}(p)$ is the set of states that have p's top tree, $p.s_0$, which contains only one node, as the left corner.

Based on this observation, we can safely delay the REDUCE($\{x\}, \mathcal{R}_C(x)$) operation (Line 11 in Algorithm 1), until the derivation x of a shifted state is popped out from Q. This helps us eliminate unnecessary right expansion.

We can delay even more derivations by extending the concept of left corner states to reduced states. Note that for any two states p, q, if q's top tree $q.s_0$ has p's top tree $p.s_0$ as left corner, and p, q share the same left states, then derivations of p should always have higher priority than derivations of q. We can further delay the generation of q's derivations until p's derivations are popped out.[2]

[2]We did not implement this idea in experiments due to its complexity.

4 Comparison with Best-First CKY and Best-First Earley

4.1 Best-First CKY and Knuth Algorithm

Vanilla CKY parsing can be viewed as searching over a hypergraph(Klein and Manning, 2005), where a hyperedge points from two nodes x, y to one node z, if x, y can form a new partial tree represented by z. Best-first CKY performs best-first search over the hypergraph, which is a special application of the Knuth Algorithm (Knuth, 1977).

Non-DP best-first shift-reduce parsing can be viewed as searching over a graph. In this graph, a node represents a derivation. A node points to all its possible descendants generated from shift and left and right reduces. This graph is actually a tree with exponentially many nodes.

DP best-first parsing enables state merging on the previous graph. Now the nodes in the hypergraph are not derivations, but equivalence classes of derivations, i.e., states. The number of nodes in the hypergraph is no longer always exponentially many, but depends on the equivalence function, which is the atomic feature function $\widetilde{\mathbf{f}}(\cdot)$ in our algorithms.

DP best-first shift-reduce parsing is still a special case of the Knuth algorithm. However, it is more difficult than best-first CKY parsing, because of the extra topological order constraints from shift actions.

4.2 Best-First Earley

DP best-first shift-reduce parsing is analogous to weighted Earley (Earley, 1970; Stolcke, 1995), because: 1) in Earley the PRED rule generates states similar to shifted states in shift-reduce parsing; and, 2) a newly completed state also needs to check all possible left expansions and right expansions, similar to a state popped from the priority queue in Algorithm 1. (see Figure 2)

Our Algorithm 2 exploits lazy expansion, which reduces unnecessary expansions, and should be more efficient than pure Earley.

5 Optimality and Polynomial Complexity

5.1 Proof of Optimality

We define a *best derivation* of state $[x]_\sim$ as a derivation x such that $\forall y \in [x]_\sim, x \preceq y$.

Note that each state has a unique feature signature. We want to prove that Algorithm 1 actually fills the chart by assigning a best derivation to its state. Without loss of generality, we assume Algorithm 1 fills C with derivations in the following order:

$$x_0, x_1, x_2, \ldots, x_m$$

where x_0 is the initial derivation, x_m is the first goal derivation in the sequence, and $C[x_i] = x_i, 0 \leq i \leq m$. Denote the status of chart right after x_k being filled as C_k. Specially, we define $C_{-1} = \emptyset$

However, we do not have superiority as in non-DP best-first parsing. Because we use a pair of prefix score and inside score, (pre, ins), as priority (Equation 2) in the deductive system (Figure 2). We have the following property as an alternative for superiority:

Lemma 1. *After derivation x_k has been filled into chart, $\forall x$ s.t. $x \in Q$, and x is a best derivation of state $[x]_\sim$, then x's descendants can not have a higher priority than x_k.*

Proof. Note that when x_k pops out, x is still in Q, so $x_k \preceq x$. Assume z is x's direct descendant.

- If $z = \mathsf{sh}(x)$ or $z = \mathsf{re}(x, \_)$, based on the deductive system, $x \prec z$, so $x_k \preceq x \prec z$.
- If $z = \mathsf{re}(y, x)$, $y \in \mathcal{L}(x)$, assume $z \prec x_k$.

  $$z.pre = y.pre + y.sh + x.ins + x.re$$

  We can construct a new derivation $x' \sim x$ by appending x's top tree, $x.s_0$ to y's stack, and

  $$x'.pre = y.pre + y.sh + x.ins < z.pre$$

  So $x' \prec z \prec x_k \preceq x$, which contradicts that x is a best derivation of its state.

With induction we can easily show that any descendants of x can not have a higher priority than x_k. □

We can now derive:

Theorem 1 (Stepwise Completeness and Optimality)**.** *For any k, $0 \leq k \leq m$, we have the following two properties:*

$$\forall x \prec x_k, [x]_\sim \in C_{k-1} \quad \text{(Stepwise Completeness)}$$
$$\forall x \sim x_k, x_k \preceq x \quad \text{(Stepwise Optimality)}$$

Proof. We prove by induction on k.

1. For $k = 0$, these two properties trivially hold.

2. Assume this theorem holds for $k = 2, ..., i-1$. For $k = i$, we have:

a) *[Proof for Stepwise Completeness]*

(Proof by Contradiction) Assume $\exists x \prec x_i$ s.t. $[x]_\sim \notin C_{i-1}$.Without loss of generality we take a best derivation of state $[x]_\sim$ as x. x must be derived from other best derivations only. Consider this derivation transition hypergraph, which starts at initial derivation $x_0 \in C_{i-1}$, and ends at $x \notin C_{i-1}$.

There must be a best derivation x' in this transition hypergraph, s.t. all best parent derivation(s) of x' are in C_{i-1}, but not x'.

If x' is a reduced derivation, assume x''s best parent derivations are $y \in C_{i-1}, z \in C_{i-1}$. Because y and z are best derivations, and they are in C_{i-1}, from Stepwise Optimality on $k = 1, ..., i-1$, $y, z \in \{x_0, x_1, \ldots, x_{i-1}\}$. From Line 7-11 in Algorithm 1, x' must have been pushed into Q when the latter of y, z is popped.

If x' is a shifted derivation, similarly x' must have been pushed into Q.

As $x' \notin C_{i-1}$, x' must still be in Q when x_i is popped. However, from Lemma 1, none of x''s descendants can have a higher priority than x_i, which contradicts $x \prec x_i$.

b) *[Proof for Stepwise Optimality]*

(Proof by Contradiction) Assume $\exists x \sim x_i$ s.t. $x \prec x_i$. From Stepwise Completeness on $k = 1, ..., i$, $x \in C_{i-1}$, which means the state $[x_i]_\sim$ has already been assigned to x, contradicting the premise that x_i is pushed into chart.

□

Both of the two properties have very intuitive meanings. Stepwise Optimality means Algorithm 1 only fills chart with a best derivation for each state. Stepwise Completeness means every state that has its best derivation better than best derivation p_i must have been filled before p_i, this guarantees that the global best goal derivation is captured by Algorithm 1.

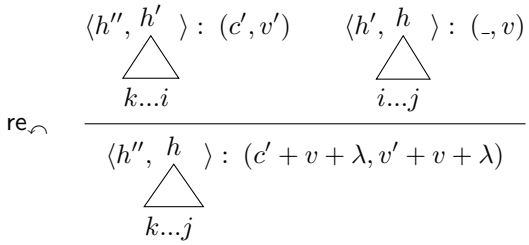


Figure 4: Example of shift-reduce with dynamic programming: simulating an edge-factored model. GSS is implicit here, and $\mathsf{re}_{\curvearrowleft}$ case omitted. Here $\lambda = sc_{\mathsf{sh}}(h'', h') + sc_{\mathsf{re}_{\curvearrowleft}}(h', h)$.

More formally we have:

Theorem 2 (Optimality of Algorithm 1)**.** *The first goal derivation popped off the priority queue is the optimal parse.*

Proof. (Proof by Contradiction.) Assume $\exists x$, x is the a goal derivation and $x \prec x_m$. Based on Stepwise Completeness of Theorem 1, $x \in C_{m-1}$, thus x has already been popped out, which contradicts that x_m is the first popped out goal derivation. □

Furthermore, we can see our lazy expansion version, i.e., Algorithm 2, is also optimal. The key observation is that we delay the reduction of derivation x' and a derivation of right states $\mathcal{R}([x']_\sim)$ (Line 11 of Algorithm 1), until shifted derivation, $x = \mathsf{sh}(x')$, is popped out (Line 11 of Algorithm 2). However, this delayed reduction will not generate any derivation y, s.t. $y \prec x$, because, based on our deductive system (Figure 2), for any such kind of reduced derivations y, $y.pre = x'.pre + x'.sh + y.re + y.ins$, while $x.pre = x'.pre + x'.sh$.

5.2 Analysis of Time and Space Complexity

Following Huang and Sagae (2010) we present the complexity analysis for our DP best-first parsing.

Theorem 3. *Dynamic programming best-first parsing runs in worst-case polynomial time and space, as long as the atomic features function satisfies:*

- **bounded***: $\forall$ derivation x, $|\widetilde{\mathbf{f}}(x)|$ is bounded by a constant.*
- **monotonic***:*

– **horizontal**: $\forall k,\ \mathbf{f}_k(s) = \mathbf{f}_k(t) \Rightarrow \mathbf{f}_{k+1}(s) = \mathbf{f}_{k+1}(t)$, *for all possible trees* s, t.
– **vertical**: $\forall k,\ \mathbf{f}_k(s^\frown s') = \mathbf{f}_k(t^\frown t') \Rightarrow \mathbf{f}_k(s) = \mathbf{f}_k(t)$ *and* $\mathbf{f}_k(s^\frown s') = \mathbf{f}_k(t^\frown t') \Rightarrow \mathbf{f}_k(s') = \mathbf{f}_k(t')$, *for all possible trees* s, s', t, t'.

In the above theorem, boundness means we can only extract finite information from a derivation, so that the atomic feature function $\widetilde{\mathbf{f}}(\cdot)$ can only distinguish a finite number of different states. Monotonicity requires the feature representation $\mathbf{f}_k$ subsumes $\mathbf{f}_{k+1}$. This is necessary because we use the features as signature to match all possible left states and right states (Equation 1). Note that we add the vertical monotonicity condition following the suggestion from Kuhlmann et al. (2011), which fixes a flaw in the original theorem of Huang and Sagae (2010).

We use the edge-factored model (Eisner, 1996; McDonald et al., 2005) with dynamic programming described in Figure 4 as a concrete example for complexity analysis. In the edge-factored model the feature set consists of only combinations of information from the roots of the two top trees s_1, s_0, and the queue. So the atomic feature function is

$$\widetilde{\mathbf{f}}(p) = (i, j, h(p.s_1), h(p.s_0))$$

where $h(s)$ returns the head word index of tree s.

The deductive system for the edge-factored model is in Figure 4. The time complexity for this deductive system is $O(n^6)$, because we have three head indexes and three span indexes as free variables in the exploration. Compared to the work of Huang and Sagae (2010), we reduce the time complexity from $O(n^7)$ to $O(n^6)$ because we do not need to keep track of the number of the steps for a state.

6 Experiments

In experiments we compare our DP best-first parsing with non-DP best-first parsing, pure greedy parsing, and beam parser of Huang and Sagae (2010).

Our underlying MaxEnt model is trained on the Penn Treebank (PTB) following the standard split: Sections 02-21 as the training set and Section 22 as the held-out set. We collect gold actions at different parsing configurations as positive examples from gold parses in PTB to train the MaxEnt model. We use the feature set of Huang and Sagae (2010).

	model score	accuracy	# states	time
greedy	-1.4303	90.08%	125.8	0.0055
beam*	-1.3302	90.60%	869.6	0.0331
non-DP	-1.3269	90.70%	$4,194.4$	0.2622
DP	-1.3269	90.70%	243.2	0.0132

Table 1: Dynamic programming best-first parsing reach optimality faster. *: for beam search we use beam size of 8. (All above results are averaged over the held-out set.)

Furthermore, we reimplemented the beam parser with DP of Huang and Sagae (2010) for comparison. The result of our implementation is consistent with theirs. We reach 92.39% accuracy with structured perceptron. However, in experiments we still use MaxEnt to make the comparison fair.

To compare the performance we measure two sets of criteria: 1) the internal criteria consist of the model score of the parsing result, and the number of states explored; 2) the external criteria consist of the unlabeled accuracy of the parsing result, and the parsing time.

We perform our experiments on a computer with two 3.1GHz 8-core CPUs (16 processors in total) and 64GB RAM. Our implementation is in Python.

6.1 Search Quality & Speed

We first compare DP best-first parsing algorithm with pure greedy parsing and non-DP best-first parsing without any extra constraints.

The results are shown in Table 1. Best-first parsing reaches an accuracy of 90.70% in the held-out set. Since that the MaxEnt model is locally trained, this accuracy is not as high as the best shift-reduce parsers available now. However, this is sufficient for our comparison, because we aim at improving the search quality and efficiency of parsing.

Compared to greedy parsing, DP best-first parsing reaches a significantly higher accuracy, with $\sim$2 times more parsing time. Given the extra time in maintaining priority queue, this is consistent with the internal criteria: DP best-first parsing reaches a significantly higher model score, which is actually optimal, exploring twice as many as states.

On the other hand, non-DP best-first parsing also achieves the optimal model score and accuracy.

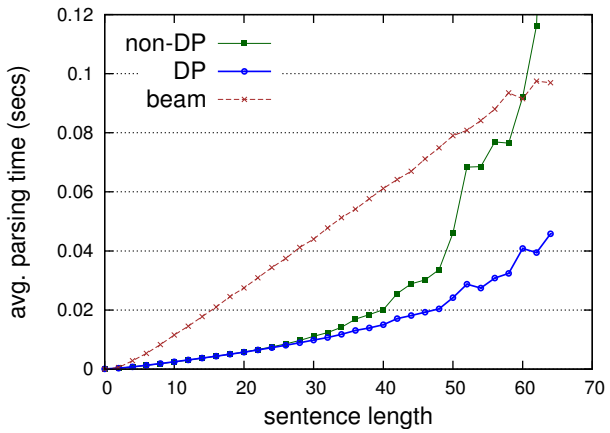


Figure 5: DP best-first significantly reduces parsing time. Beam parser (beam size 8) guarantees linear parsing time. Non-DP best-first parser is fast for short sentences, but the time grows exponentially with sentence length. DP best-first parser is as fast as non-DP for short sentences, but the time grows significantly slower.

However, it explores $\sim$17 times more states than DP, with an unbearable average time.

Furthermore, on average our DP best-first parsing is significantly faster than the beam parser, because most sentences are short.

Figure 5 explains the inefficiency of non-DP best-first parsing. As the time complexity grows exponentially with the sentence length, non-DP best-first parsing takes an extremely long time for long sentences. DP best-first search has a polynomial time bound, which grows significantly slower.

In general DP best-first parsing manages to reach optimality in tractable time with exact search. To further investigate the potential of this DP best-first parsing, we perform inexact search experiments with bounded priority queue.

6.2 Parsing with Bounded Priority Queue

Bounded priority queue is a very practical choice when we want to parse with only limited memory.

We bound the priority queue size at 1, 2, 5, 10, 20, 50, 100, 500, and 1000, and once the priority queue size exceeds the bound, we discard the worst one in the priority queue. The performances of non-DP best-first parsing and DP best-first parsing are illustrated in Figure 6 (a) (b).

Firstly, in Figure 6 (a), our DP best-first parsing reaches the optimal model score with bound 50, while non-DP best-first parsing fails even with bound 1000. Also, in average with bound 1000, compared to non-DP, DP best-first only needs to explore less than half of the number of states.

Secondly, for external criteria in Figure 6 (b), both algorithms reach accuracy of 90.70% in the end. In speed, with bound 1000, DP best-first takes $\sim 1/3$ time of non-DP to parse a sentence in average.

Lastly, we also compare to beam parser with beam size 1, 2, 4, 8. Figure 6 (a) shows that beam parser fails to reach the optimality, while exploring significantly more states. On the other hand, beam parser also fails to reach an accuracy as high as best-first parsers. (see Figure 6 (b))

6.3 Simulating the Edge-Factored Model

We further explore the potential of DP best-first parsing with the edge-factored model.

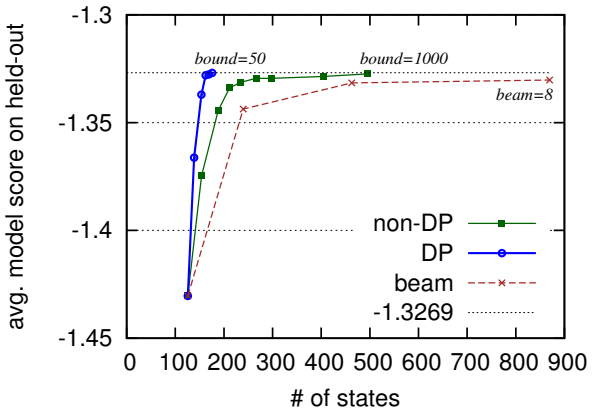
The simplified feature set of the edge-factored model reduces the number of possible states, which means more state-merging in the search graph. We expect more significant improvement from our DP best-first parsing in speed and number of explored states.

Experiment results confirms this. In Figure 6 (c) (d), curves of DP best-first diverge from non-DP faster than standard model (Figure 6 (a) (b)).

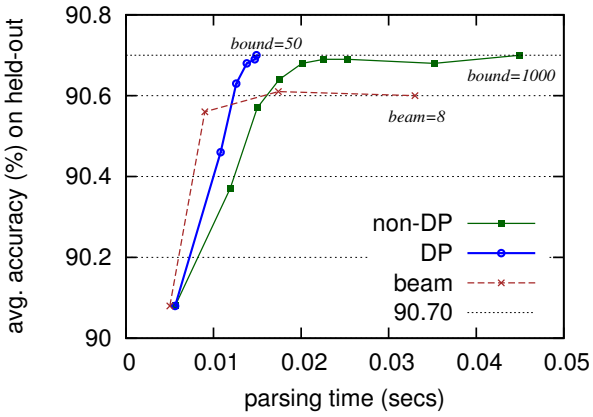
7 Conclusions and Future Work

We have presented a dynamic programming algorithm for best-first shift-reduce parsing which is guaranteed to return the optimal solution in polynomial time. This algorithm is related to best-first Earley parsing, and is more sophisticated than best-first CKY. Experiments have shown convincingly that our algorithm leads to significant speedup over the non-dynamic programming baseline, and makes exact search tractable for the first-time under this model.

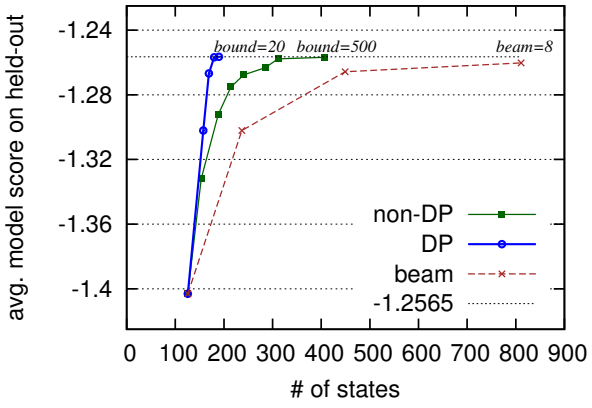
For future work we would like to improve the performance of the probabilistic models that is required by the best-first search. We are also interested in exploring A* heuristics to further speed up our DP best-first parsing.



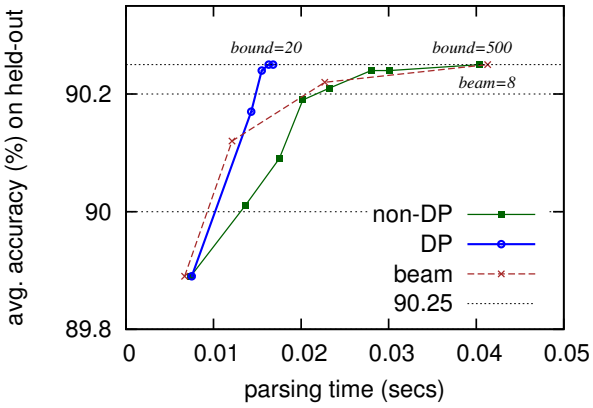
(a) search quality vs. # of states



(b) parsing accuracy vs. time



(c) search quality vs. # of states (edge-factored)



(d) parsing accuracy vs. time (edge-factored)

Figure 6: Parsing performance comparison between DP and non-DP. (a) (b) Standard model with features of Huang and Sagae (2010). (c) (d) Simulating edge-factored model with reduced feature set based on McDonald et al. (2005). Note that to implement bounded priority queue we use two priority queues to keep track of the worst elements, which introduces extra overhead, so that our bounded parser is slower than the unbounded version for large priority queue size bound.

References

Alfred V. Aho and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation, and Compiling*, volume I: Parsing of *Series in Automatic Computation*. Prentice Hall, Englewood Cliffs, New Jersey.

Sharon A Caraballo and Eugene Charniak. 1998. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24(2):275–298.

Jay Earley. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

Jason Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of COLING*.

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

Dan Klein and Christopher D Manning. 2003. A* parsing: Fast exact Viterbi parse selection. In *Proceedings of HLT-NAACL*.

Dan Klein and Christopher D Manning. 2005. Parsing and hypergraphs. In *New developments in parsing technology*, pages 351–372. Springer.

Donald Knuth. 1977. A generalization of Dijkstra's algorithm. *Information Processing Letters*, 6(1).

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

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd ACL*.

Mark-Jan Nederhof. 2003. Weighted deductive parsing and Knuth's algorithm. *Computational Linguistics*, pages 135–143.

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. Association for Computational Linguistics.

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

Andreas Stolcke. 1995. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201.

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*.