LTAG Dependency Parsing with Bidirectional Incremental Construction

Libin Shen
BBN Technologies
lshen@bbn.com

Aravind K. Joshi
University of Pennsylvania
joshi@cis.upenn.edu

Abstract

In this paper, we first introduce a new architecture for parsing, bidirectional incremental parsing. We propose a novel algorithm for incremental construction, which can be applied to many structure learning problems in NLP. We apply this algorithm to LTAG dependency parsing, and achieve significant improvement on accuracy over the previous best result on the same data set.

1 Introduction

The phrase "Bidirectional Incremental" may appear self-contradictory at first sight, since incremental parsing usually means left-to-right parsing in the context of conventional parsing. In this paper, we will extend the meaning of incremental parsing.

The idea of bidirectional parsing is related to the bidirectional sequential classification method described in (Shen et al., 2007). In that paper, a tagger assigns labels to words of highest confidence first, and then these labels in turn serve as the context of later labelling operations. The bidirectional tagger obtained the best results in literature on POS tagging on the standard PTB dataset.

We extend this method from labelling to structure learning, The search space of structure learning is much larger, so that it is appropriate to exploit confidence scores in search.

In this paper, we are interested in LTAG dependency parsing because TAG parsing is a well known problem of high computational complexity in regular parsing. In order to get a focus for the learning algorithm, we work on a variant of LTAG based parsing in which we learn the word dependency relations encoded in LTAG derivations instead of the full-fledged trees.

1.1 Parsing

Two types of parsing strategies are popular in natural language parsing, which are chart parsing and incremental parsing.

Suppose the input sentence is $w_1 w_2 ... w_n$. Let cell $[i, j]$ represent $w_i w_{i+1} ... w_j$, a substring of the sentence. As far as CFG parsing is concerned, a chart parser computes the possible structures over all possible cells $[i, j]$, where $1 \leq i \leq j \leq n$. The order of computing on these $n(n+1)/2$ cells is based on some partial order $\preceq$, such that $[p_1, p_2] \preceq [q_1, q_2]$ if $q_1 \leq p_1 \leq p_2 \leq q_2$. In order to employ dynamic programming, one can only use a fragment of a hypothesis to represent the whole hypothesis, which is assumed to satisfy conditional independence assumption. It is well known that richer context representation gives rise to better parsing performance (Johnson, 1998). However, the need for tractability does not allow much internal information to be used to represent a hypothesis. The designs of hypotheses in (Collins, 1999; Charniak, 2000) show a delicate balance between expressiveness and tractability, which play an important role in natural language parsing.

Some recent work on incremental parsing (Collins and Roark, 2004; Shen and Joshi, 2005) showed another way to handle this problem. In these incremental parsers, tree structures are used to represent the left context. In this way, one can access the whole tree to collect rich context information at the expense of being limited to beam search, which only maintains k-best results at each

Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing, pages 495–504,
Honolulu, October 2008.

step. Compared to chart parsing, incremental parsing searches for the analyses for only $2n-1$ cells, $[1,1],[2,2],[1,2],..,[i,i],[1,i],..,[1,n]$, incrementally, while complex structures are used for the analyses for each cell, which satisfy conditional independence under a much weaker assumption.

In this paper, we call this particular approach **left-to-right** incremental parsing, since one can also search from right to left incrementally in a similar way. A major problem of the left-to-right approach is that one can only utilize the structural information on the left side but not the right side.

1.2 Parsing as Bidirectional Construction

A natural way to handle this problem is to employ bidirectional search, which means we can dynamically search the space in two directions. So we expand the idea of incremental parsing by introducing greedy search. Specifically, we look for the hypotheses over the cell $[1,n]$ by building analyses over $2n-1$ cells $[a_{i,1},a_{i,2}], i=1,..,2n-1$ step by step, where $[a_{2n-1,1},a_{2n-1,2}]=[1,n]$. Furthermore, for any $[a_{i,1},a_{i,2}]$

- $a_{i,1}=a_{i,2}$, or
- $\exists j,k,$ such that $[a_{i,1},a_{i,2}]=[a_{j,1},a_{k,2}]$, where $j<i$, $k<i$ and $a_{j,2}+1=a_{k,1}$.

It is easy to show that the set $\{[a_{i,1},a_{i,2}] \mid 1 \leq i \leq 2n-1\}$ forms a *tree* relation, which means that each cell except the last one will be used to build another cell just once. In this framework, we can begin with several starting points in a sentence and search in any direction. So left-to-right parsing is only a special case of **incremental** parsing defined in this way. We still use complex structures to represent the partial analyses, so as to employ both top-down and bottom-up information as in (Collins and Roark, 2004; Shen and Joshi, 2005). Furthermore, we can utilize the rich context on both sides of the partial results.

Similar to bidirectional labelling in (Shen et al., 2007), there are two learning tasking in this model. First, we need to learn which cell we should choose. At each step, we can select only one path. Secondly, we need to learn which operation we should take for a given cell. We maintain k-best candidates for each cell instead of only one, which differentiates this model from normal greedy search. So our model is more robust. Furthermore, we need to find an effective way to iterate between these two tasks.

Instead of giving an algorithm specially designed for parsing, we generalize the problem for graphs. A sentence can be viewed as a graph in which words are viewed as vertices and neighboring words are connected with an arc. In Sections 2 and 3, we will propose decoding and training algorithms respectively for graph-based incremental construction, which can be applied to many structure learning problems in NLP.

We will apply this algorithm to dependency parsing of Lexicalized Tree Adjoining Grammar (Joshi and Schabes, 1997). Specifically, we will train and evaluate an LTAG dependency parser over the LTAG treebank described in Shen et al. (2008). We report the experimental results on PTB section 23 of the LTAG treebank. The accuracy on LTAG dependency is 90.5%, which is 1.2 points over 89.3%, the previous best result (Shen and Joshi, 2005) on the same data set.

It should be noted that PTB-based bracketed labelling is not an appropriate evaluation metric here, since the experiments are on an LTAG treebank. The derived trees in the LTAG treebank are different from the CFG trees in PTB. Hence, we do not use metrics such as labeled precision and labeled recall for evaluation.

2 Graph-based Incremental Construction

2.1 Idea and Data Structures

Now we define the problem formally. We will use dependency parsing as an example to illustrate the idea.

We are given a connected graph $G(V,E)$ whose hidden structure is U, where $V=\{v_i\}$, $E \subseteq V \times V$ is a symmetric relation, and $U=\{u_k\}$ is composed of a set of elements that vary with applications. As far as dependency parsing is concerned, the input graph is simply a chain of vertices, where $E(v_{i-1},v_i)$, and its hidden structure is $\{u_k=(v_{s_k},v_{e_k},b_k)\}$, where vertex v_{e_k} depends on vertex v_{s_k} with label b_k.

A graph-based incremental construction algorithm looks for the hidden structure in a bottom-up

style.

Let x_i and x_j be two sets of connected vertexes in V, where $x_i \cap x_j = \phi$ and they are directly connected via an edge in E. Let y^{xi} be a hypothesized hidden structure of x_i, and y^{xj} a hypothesized hidden structure of x_j.

Suppose we choose to combine y^{xi} and y^{xj} with an operation r to build a hypothesized hidden structure for $x_k = x_i \cup x_j$. We say the process of construction is **incremental** if the output of the operation, $y^{xk} = r(x_i, x_j, y^{xi}, y^{xj}) \supseteq y^{xi} \cup y^{xj}$ for all the possible x_i, x_j, y^{xi}, y^{xj} and operation r. As far as dependency parsing is concerned, incrementality means that we cannot remove any links coming from the substructures.

Once y^{xk} is built, we can no longer use y^{xi} or y^{xj} as a building block. It is easy to see that left to right incremental construction is a special case of our approach. So the question is how we decide the order of construction as well as the type of operation r. For example, in the very first step of dependency parsing, we need to decide which two words are to be combined as well as the dependency label to be used.

This problem is solved statistically, based on the features defined on the substructures involved in the operation and their context. Suppose we are given the weights of these features, we will show in the next section how these parameters guide us to build a set of hypothesized hidden structures with beam search. In Section 3, we will present a Perceptron like algorithm (Collins, 2002; Daumé III and Marcu, 2005) to obtain the parameters.

Now we introduce the data structure to be used in our algorithms.

A **fragment** is a *connected* sub-graph of $G(V, E)$. Each fragment x is associated with a set of hypothesized hidden structures, or **fragment hypotheses** for short: $Y^x = \{y_1^x, ..., y_k^x\}$. Each y^x is a possible fragment hypothesis of x.

It is easy to see that an operation to combine two fragments may depend on the fragments in the context, i.e. fragments directly connected to one of the operands. So we introduce the **dependency** relation over fragments. Suppose there is a dependency relation $D \subseteq F \times F$, where $F \subseteq 2^V$ is the set of all fragments in graph G. $D(x_i, x_j)$ means that any operation on a fragment hypothesis of x_i depends on the features in the fragment hypothesis of x_j, and vice versa.

We are especially interested in the following two dependency relations.

- *level-0 dependency*: $D_0(x_i, x_j) \iff i = j$.
- *level-1 dependency*: $D_1(x_i, x_j) \iff x_i$ and x_j are directly connected in G.

Level-0 dependency means that the features of a hypothesis for a vertex x_i do not depend on the hypotheses for other vertices. Level-1 dependency means that the features depend on the hypotheses of nearby vertices only.

The learning algorithm for level-0 dependency is similar to the guided learning algorithm for labelling as described in (Shen et al., 2007). Level-1 dependency requires more data structures to maintain the hypotheses with dependency relations among them. However, we do not get into the details of level-1 formalism in this papers for two reasons. One is the limit of page space and depth of a conference paper. On the other hand, our experiments show that the parsing performance with level-1 dependency is close to what level-0 dependency could provides. Interested readers could refer to (Shen, 2006) for detailed description of the learning algorithms for level-1 dependency.

2.2 Algorithms

Algorithm 1 shows the procedure of building hypotheses incrementally on a given graph $G(V, E)$. Parameter k is used to set the beam width of search. Weight vector $\mathbf{w}$ is used to compute score of an operation.

We have two sets, H and Q, to maintain hypotheses. Hypotheses in H are selected in beam search, and hypotheses in Q are candidate hypotheses for the next step of search in various directions.

We first initiate the hypotheses for each vertex, and put them into set H. For example, in dependency parsing, the initial value is a set of possible POS tags for each single word. Then we use a queue Q to collect all the possible hypotheses over the initial hypotheses H.

Whenever Q is not empty, we search for the hypothesis with the highest score according to a given weight vector $\mathbf{w}$. Suppose we find (x, y). We select

Algorithm 1 Incremental Construction

Require: graph $G(V, E)$;
Require: beam width k;
Require: weight vector $\mathbf{w}$;
1: $H \leftarrow init_H()$;
2: $Q \leftarrow init_Q(H)$;
3: **repeat**
4: $(x', y') \leftarrow arg\max_{(x,y)\in Q} score(y)$;
5: $H \leftarrow update_H(H, x')$;
6: $Q \leftarrow update_Q(Q, H, x')$;
7: **until** $(Q = \phi)$

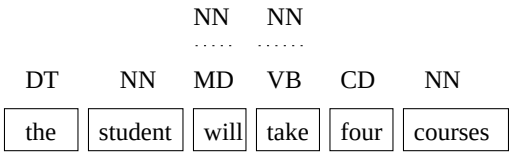

Figure 1: After initialization

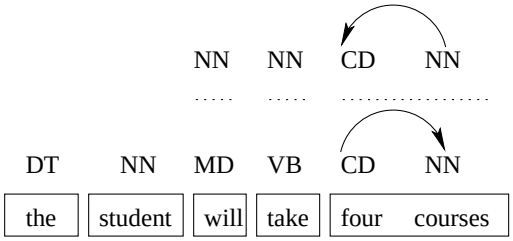

Figure 2: Step 1

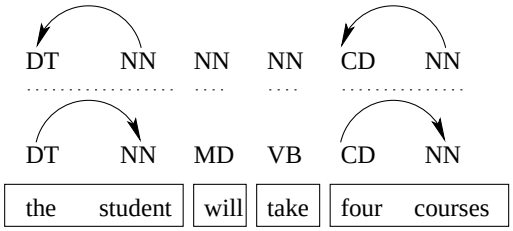

Figure 3: Step 2

top k-best hypotheses for segment x from Q and use them to update H. Then we remove from Q all the hypotheses for segments that have overlap with segment x. In the end, we build new candidate hypotheses with the updated selected hypothesis set H, and add them to Q.

2.3 An Example

We use an example of dependency parsing to illustrate the incremental construction algorithm first.

Suppose the input sentence is *the student will take four courses*. We are also given the candidate POS tags for each word. So the graph is just a linear structure in this case. We use level-0 dependency and set beam width to two.

We use boxes to represent fragments. The dependency links are from the parent to the child.

Figure 1 shows the result after initialization. Figure 2 shows the result after the first step, combining the fragments of *four* and *courses*. Figure 3 shows the result after the second step, combining *the* and *student*, and figure 4 shows the result after the third step, combining *take* and *four courses*. Due to limited space, we skip the rest operations.

2.4 Description

Now we will explain the functions in Algorithm 1 one by one.

- $init_H()$ initiates hypotheses for each vertex. Here we set the initial fragment hypotheses, $Y^{xi} = \{y_1^{xi}, ..., y_k^{xi}\}$, where $x_i = \{v_i\}$ contains only one vertex.

- $init_Q(H)$ initiates the queue of candidate operations over the current hypotheses H. Supposed there exist segments x_i and x_j which are directly connected in G. We apply all possible operations to all fragment hypotheses for x_j and x_j, and add the result hypotheses in Q. For example, we generate (x, y) with some operation r, where segment x is $x_i \cup x_j$.

  All the candidate operations are organized with respect to the segments. For each segment, we maintain top k candidates according to their scores.

- $update_H(H, x)$ is used to update hypotheses in H. First, we remove from H all the hypotheses whose corresponding segment is a sub-set of x. Then, we add into H the top k hypotheses for segment x.

- $update_Q(Q, H, x)$ is also designed to complete two tasks. First, we remove from Q all the hypotheses whose corresponding segment has overlap with segment x. Then, we add new candidate hypotheses depending on x in a way

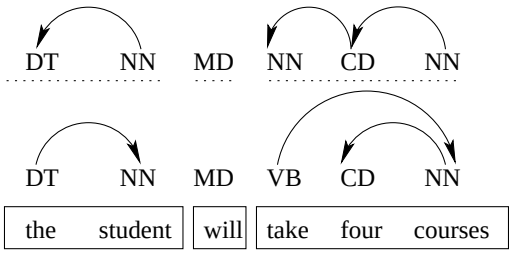

Figure 4: Step 3

Algorithm 2 Parameter Optimization

```
1: w ← 0;
2: for (round r = 0; r < R; r++) do
3:    load graph G_r(V, E), gold standard H_r;
4:    initiate H and Q;
5:    repeat
6:       (x', y') ← arg max_{(x,y)∈Q} score(y);
7:       if (y' is compatible with H_r) then
8:          update H and Q;
9:       else
10:         ỹ ← positive(Q, x');
11:         promote(w, ỹ);
12:         demote(w, y');
13:         update Q with w;
14:      end if
15:   until (Q = φ)
16: end for
```

similar to the $init_Q(H)$ function. For each segment, we maintain the top k candidates for each segment.

3 Parameter Optimization

In the previous section, we described an algorithm for graph-based incremental construction for a given weight vector **w**. In Algorithm 2, we present a Perceptron like algorithm to obtain the weight vector for the training data.

For each given training sample (G_r, H_r), where H_r is the gold standard hidden structure of graph G_r, we first initiate cut T, hypotheses H^T and candidate queue Q by calling $init_H$ and $init_Q$ as in Algorithm 1.

Then we use the gold standard H_r to guide the search. We select candidate (x', y') which has the highest operation score in Q. If y' is compatible with H_r, we update H and Q by calling $update_H$ and $update_Q$ as in Algorithm 1. If y' is incompatible with H_r, we treat y' as a negative sample, and search for a positive sample $\tilde{y}$ in Q with $positive(Q, x')$.

If there exists a hypothesis $\tilde{y}^{x'}$ for fragment x' which is compatible with H_r, then $positive(Q, x')$ returns $\tilde{y}^{x'}$. Otherwise $positive(Q, x')$ returns the candidate hypothesis which is compatible with H_r and has the highest operation score in Q.

Then we update the weight vector **w** with $\tilde{y}$ and y'. At the end, we update the candidate Q by using the new weights **w**.

In order to improve the performance, we use *Perceptron with margin* in the training (Krauth and Mézard, 1987). The margin is proportional to the loss of the hypothesis. Furthermore, we use averaged weights (Collins, 2002; Freund and Schapire, 1999) in Algorithm 1.

4 LTAG Dependency Parsing

We apply the new algorithm to LTAG dependency parsing on an LTAG Treebank (Shen et al., 2008) extracted from Penn Treebank (Marcus et al., 1994) and Proposition Bank (Palmer et al., 2005). Penn Treebank was previously used to train and evaluate various dependency parsers (Yamada and Matsumoto, 2003; McDonald et al., 2005). In these works, Magerman's rules are used to pick the head at each level according to the syntactic labels in a local context.

The dependency relation encoded in the LTAG Treebank reveals deeper information for the following two reasons. First, the LTAG architecture itself reveals deeper dependency. Furthermore, the PTB was reconciled with the Propbank in the LTAG Treebank extraction (Shen et al., 2008).

We are especially interested in the two types of structures in the LTAG Treebank, predicate adjunction and predicate coordination. They are used to encode dependency relations which are unavailable in other approaches. On the other hand, these structures turn out to be a big problem for the general representation of dependency relations, including adjunction and coordination. We will show that the algorithm proposed here provides a nice solution for this problem.

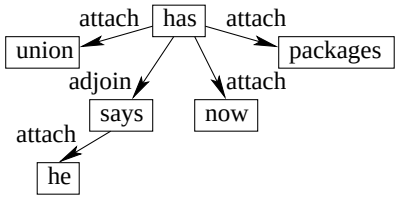

Figure 5: Predicate Adjunction

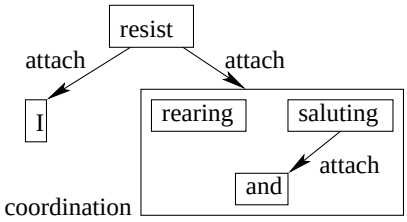

Figure 6: Predicate Coordination

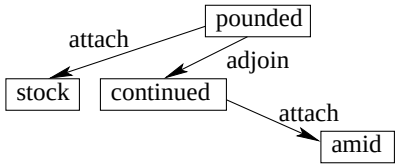

Figure 7: Non-projective Adjunction

4.1 Representation of the LTAG Treebank

In the LTAG Treebank (Shen et al., 2008), each word is associated with a spinal template, which represents the projection from the lexical item to the root. Templates are linked together to form a *derivation* tree. The topology of the derivation tree shows a type of dependency relation, which we call **LTAG dependency** here.

There are three types of operations in the LTAG Treebank, which are attachment, adjunction, and coordination. Attachment is used to represent both substitution and sister adjunction in the traditional LTAG. So it is similar to the dependency relation in other approaches.

The LTAG dependency can be a non-projective relation thanks to the operation of adjunction. In the LTAG Treebank, raising verbs and passive ECM verbs are represented as auxiliary trees to be adjoined. In addition, adjunction is used to handle many cases of discontinuous arguments in Propbank. For example, in the following sentence, ARG1 of *says* in Propbank is discontinuous, which is *First Union now has packages for seven customer groups*.

- *First Union, he says, now has packages for seven customer groups.*

In the LTAG Treebank, the subtree for *he says* adjoins onto the node of *has*, which is the root of the derivation tree, as shown in Figure 5.

Another special aspect of the LTAG Treebank is the representation of predicate coordination. Figure 6 is the representation of the following sentence.

- *I couldn't resist rearing up on my soggy loafers and saluting.*

The coordination between *rearing* and *saluting* is represented explicitly with a coord-structure, and this coord-structure attaches to *resist*. It is shown in (Shen et al., 2008) that coord-structures could encode the ambiguity of argument sharing, which can be non-projective also.

4.2 Incremental Construction

We build LTAG derivation trees incrementally. A hypothesis of a fragment is represented with a partial derivation tree. When the fragment hypotheses of two nearby fragments combine, the partial derivation trees are combined into one.

It is trivial to combine two partial derivation trees with attachment. We simply attach the root of one tree to some node on the other tree which is *visible* to this root node. Adjunction is similar to attachment, except that an adjoined subtree may be *visible* from the other side of the derivation tree. For example, in sentence

- *The stock of UAL Corp. continued to be pounded amid signs that British Airways ...*

continued adjoins onto *pounded*, and *amid* attaches to *continued* from the other side of the derivation tree (*pounded* is between *continued* and *amid*), as shown in Figure 7.

The predicate coordination is decomposed into a set of operations to meet the need for incremental processing. Suppose a coordinated structure attaches to the parent node on the left side. We build this structure incrementally by attaching the first

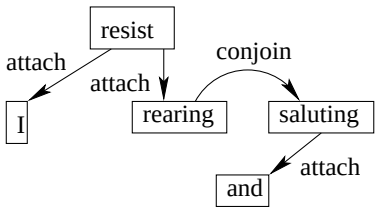

Figure 8: Conjunction

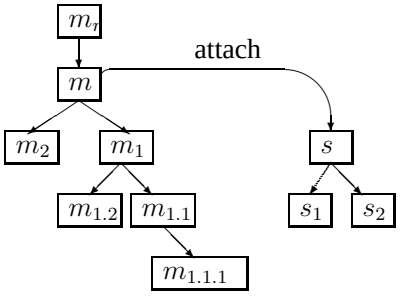

Figure 9: Representation of nodes

conjunct to the parent and conjoining other conjuncts to first one. In this way, we do not need to force the coordination to be built before the attachment. Either can be executed first. A sample is shown in Figure 8.

4.3 Features

In this section, we will describe the features used in LTAG dependency parsing. An operation is represented by a 4-tuple

- $op = (type, dir, pos_{left}, pos_{right})$,

where $type \in \{attach, adjoin, conjoin\}$ and dir is used to represent the direction of the operation. pos_{left} and pos_{right} are the POS tags of the two operands.

Features are defined on POS tags and lexical items of the nodes in the context. In order to represent the features, we use m for the main-node of the operation, s for the sub-node, m_r for the parent of the main-node, $m_1..m_i$ for the children of m, and $s_1..s_j$ for the children of s, as shown in Figure 9. The index always starts from the side where the operation takes place. We use the Gorn addresses to represent the nodes in the subtrees rooted on m and s.

Furthermore, we use l_k and r_k to represent the nodes in the left and right context of the flat sentence. We use h_l and h_r to represent the head of the hypothesis trees on the left and right context respectively. Let x be a node. We use $x.p$ to represent the POS tag of node x, and $x.w$ to represent the lexical item of node x.

Table 1 show the features used in LTAG dependency parsing. There are seven classes of features. The first three classes of features are those defined on only one operand, on both operands, and on the siblings respectively. If gold standard POS tags are used as input, we define features on the POS tags in the context. If level-1 dependency is used, we define features on the root node of the hypothesis partial derivation trees in the neighborhood.

Half check and full check features are designed for grammatical check. For example, in Figure 9, node s attaches onto node m from left. Then nothing can attach onto s from the right side. The children of the right side of s are fixed, so we use the half check features to check the completeness of the children of the right half for s. Furthermore, we notice that all the rightmost descendants of s and the leftmost descendants of m at each level become unavailable for any further operation. So their children are fixed after this operation. All these nodes are in the form of $m_{1.1...1}$ or $s_{1.1...1}$. We use full check features to check the children from both sides for these nodes.

In the discussion above, we ignored adjunction and conjunction. We need to slightly refine the conditions of checking. Due to the limit of space, we skip these cases.

5 Experiments

We use the same data set as in (Shen and Joshi, 2005). We use Sec. 2-21 of the LTAG Treebank for training, Sec. 22 for feature selection, and Sec. 23 for test. Table 2 shows the comparison of different models. Beam size is set to five in our experiments. With level-0 dependency, our system achieves an accuracy of 90.3% at the speed of 4.25 sentences a second on a Xeon 3G Hz processor with JDK 1.5. With level-1 dependency, the parser achieves 90.5% at 3.59 sentences a second. Level-1 dependency does not provide much improvement due to the fact that level-0 features provide most of the useful information for this specific application.

It is interesting to compare our system with other dependency parsers. The accuracy on LTAG depen-

category	description	templates
one operand	Features defined on only one operand. For each template tp, $[type, dir, tp]$ is used as a feature.	$(m.p)$, $(m.w)$, $(m.p, m.w)$, $(s.p)$, $(s.w)$, $(s.p, s.w)$
two operands	Features defined on both operands. For each template tp, $[op, tp]$ is used as a feature. In addition, $[op]$ is also used as a feature.	$(m.w)$, $(s.w)$, $(m.w, s.w)$
siblings	Features defined on the children of the main nodes. For each template tp, $[op, tp]$, $[op, m.w, tp]$, $[op, m_r.p, tp]$ and $[op, m_r.p, m.w, tp]$ are used as features.	$(m_1.p)$, $(m_1.p, m_2.p)$, .., $(m_1.p, m_2.p, .., m_i.p)$
POS context	In the case that gold standard POS tags are used as input, features are defined on the POS tags of the context. For each template tp, $[op, tp]$ is used as a feature.	$(l_2.p)$, $(l_1.p)$, $(r_1.p)$, $(r_2.p)$, $(l_2.p, l_1.p)$, $(l_1.p, r_1.p)$, $(r_1.p, r_2.p)$
tree context	In the case that level-1 dependency is employed, features are defined on the trees in the context. For each template tp, $[op, tp]$ is used as a feature.	$(h_l.p)$, $(h_r.p)$
half check	Suppose $s_1, ..., s_k$ are all the children of s which are between s and m in the flat sentence. For each template tp, $[tp]$ is used as a feature.	$(s.p, s_1.p, s_2.p, .., s_k.p)$, $(m.p, s.p, s_1.p, s_2.p, .., s_k.p)$ and $(s.w, s.p, s_1.p, s_2.p, .., s_k.p)$, $(s.w, m.p, s.p, s_1.p, s_2.p, .., s_k.p)$ if $s.w$ is a verb
full check	Let x_1, x_2, .., x_k be the children of x, and x_r the parent of x. For any $x = m_{1.1...1}$ or $s_{1.1...1}$, template tp, $[tp(x)]$ is used as a feature.	$(x.p, x_1.p, x_2.p, .., x_k.p)$, $(x_r.p, x.p, x_1.p, x_2.p, .., x_k.p)$ and $(x.w, x.p, x_1.p, x_2.p, .., x_k.p)$, $(x.w, x_r.p, x.p, x_1.p, x_2.p, .., x_k.p)$ if $x.w$ is a verb

Table 1: Features defined on the context of operation

model	accuracy%
Shen and Joshi, 2005	89.3
level-0 dependency	90.3
level-1 dependency	90.5

Table 2: Experiments on Sec. 23 of the LTAG Treebank

dency is comparable to the numbers of the previous best systems on dependency extracted from PTB with Magerman's rules, for example, 90.3% in (Yamada and Matsumoto, 2003) and 90.9% in (McDonald et al., 2005). However, their experiments are on the PTB, while ours is on the LTAG corpus.

It should be noted that it is more difficult to learn LTAG dependencies. Theoretically, the LTAG dependencies reveal deeper relations. Adjunction can lead to non-projective dependencies, and the dependencies defined on predicate adjunction are linguistically more motivated, as shown in the examples in Figure 5 and 7. The explicit representation of predicate coordination also provides deeper relations. For example, in Figure 6, the LTAG dependency contains $resist \rightarrow rearing$ and $resist \rightarrow saluting$, while the Magerman's dependency only contains $resist \rightarrow rearing$. The explicit representation of predicate coordination will help to solve for the dependencies for shared arguments.

6 Discussion

In our approach, each fragment in the graph is associated with a hidden structure, which means that we cannot reduce it to a labelling task. Therefore, the problem of interest to us is different from previous

work on graphical models, such as CRF (Lafferty et al., 2001) and MMMN (Taskar et al., 2003).

McAllester et al. (2004) introduced Case-Factor Diagram (CFD) to transform a graph based construction problem to a labeling problem. However, adjunction, prediction coordination, and long distance dependencies in LTAG dependency parsing make it difficult to implement. Our approach provides a novel alternative to CFD.

Our learning algorithm stems from Perceptron training in (Collins, 2002). Variants of this method have been successfully used in many NLP tasks, like shallow processing (Daumé III and Marcu, 2005), parsing (Collins and Roark, 2004; Shen and Joshi, 2005) and word alignment (Moore, 2005). Theoretical justification for those algorithms can be applied to our training algorithm in a similar way.

In our algorithm, dependency is defined on complicated hidden structures instead of on a graph. Thus long distance dependency in a graph becomes local in hidden structures, which is desirable from linguistic considerations.

The search strategy of our bidirectional dependency parser is similar to that of the bidirectional CFG parser in (Satta and Stock, 1994; Ageno and Rodrguez, 2001; Kay, 1989). A unique contribution of this paper is that selection of path and decisions about action are trained simultaneously with discriminative learning. In this way, we can employ context information more effectively.

7 Conclusion

In this paper, we introduced bidirectional incremental parsing, a new architecture of parsing. We proposed a novel algorithm for graph-based incremental construction, and applied this algorithm to LTAG dependency parsing, revealing deep relations, which are unavailable in other approaches and difficult to learn. We evaluated the parser on an LTAG Treebank. Experimental results showed significant improvement over the previous best system. Incremental construction can be applied to other structure learning problems of high computational complexity, for example, such as machine translation and semantic parsing.

References

A. Ageno and H. Rodrguez. 2001. Probabilistic modelling of island-driven parsing. In *International Workshop on Parsing Technologies*.

E. Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics*.

M. Collins and B. Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL)*.

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

M. Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference of Empirical Methods in Natural Language Processing*.

H. Daumé III and D. Marcu. 2005. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proceedings of the 22nd International Conference on Machine Learning*.

Y. Freund and R. E. Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296.

M. Johnson. 1998. PCFG Models of Linguistic Tree Representations. *Computational Linguistics*, 24(4).

A. K. Joshi and Y. Schabes. 1997. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 69 – 124. Springer-Verlag.

M. Kay. 1989. Head-driven parsing. In *Proceedings of Workshop on Parsing Technologies*.

W. Krauth and M. Mézard. 1987. Learning algorithms with optimal stability in neural networks. *Journal of Physics A*, 20:745–752.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmentation and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. 1994. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

D. McAllester, M. Collins, and F. Pereira. 2004. Case-factor diagrams for structured probabilistic modeling. In *UAI 2004*.

R. McDonald, K. Crammer, and F. Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43th Annual Meeting of the Association for Computational Linguistics (ACL)*.

R. Moore. 2005. A discriminative framework for bilingual word alignment. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*.

M. Palmer, D. Gildea, and P. Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1).

G. Satta and O. Stock. 1994. Bi-Directional Context-Free Grammar Parsing for Natural Language Processing. *Artificial Intelligence*, 69(1-2).

L. Shen and A. K. Joshi. 2005. Incremental LTAG Parsing. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*.

L. Shen, G. Satta, and A. K. Joshi. 2007. Guided Learning for Bidirectional Sequence Classification. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)*.

L. Shen, L. Champollion, and A. K. Joshi. 2008. LTAG-spinal and the Treebank: a new resource for incremental, dependency and semantic parsing. *Language Resources and Evaluation*, 42(1):1–19.

L. Shen. 2006. *Statistical LTAG Parsing*. Ph.D. thesis, University of Pennsylvania.

B. Taskar, C. Guestrin, and D. Koller. 2003. Max-margin markov networks. In *Proceedings of the 17th Annual Conference Neural Information Processing Systems*.

H. Yamada and Y. Matsumoto. 2003. Statistical dependency analysis with Support Vector Machines. In *IWPT 2003*.