Incremental LTAG Parsing

Libin Shen and **Aravind K. Joshi**
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104, USA
{libin,joshi}@linc.cis.upenn.edu

Abstract

We present a very efficient statistical incremental parser for LTAG-spinal, a variant of LTAG. The parser supports the full *adjoining* operation, dynamic predicate coordination, and non-projective dependencies, with a formalism of provably stronger generative capacity as compared to CFG. Using gold standard POS tags as input, on section 23 of the PTB, the parser achieves an f-score of 89.3% for syntactic dependency defined on LTAG derivation trees, which are deeper than the dependencies extracted from PTB alone with head rules (for example, in Magerman's style).

1 Introduction

Lexicalized Tree Adjoining Grammar (LTAG) is a formalism motived by both linguistic and computational perspectives (for a relatively recent review, see (Joshi and Schabes, 1997)). Because of the introduction of the *adjoining* operation, the TAG formalism is provably stronger than Context Free Grammar (CFG) both in the weak and the strong generative power. The TAG formalism provides linguistically attractive analysis of natural language (Frank, 2002). Recent psycholinguistic experiments (Sturt and Lombardo, 2005) demonstrate that the *adjoining* operation of LTAG is required for eager incremental processing.

Vijay-Shanker and Joshi (1985) introduced the first TAG parser in a CYK-like algorithm. Because of the adjoining operation, the time complexity of LTAG parsing is as large as $O(n^6)$, compared with $O(n^3)$ of CFG parsing, where n is the length of the sentence to be parsed. Many LTAG parsers were proposed, such as the head-driven Earley style parser (Lavelli and Satta, 1991) and the head-corner parser (van Noord, 1994). The high time complexity prevents LTAG parsing from real-time applications.

In this paper, we work on LTAG-spinal (Shen and Joshi, 2005), an interesting subset of LTAG, which preserves almost all of the strong generative power of LTAG, and it is both weakly and strongly more powerful than CFG [1]. We will present a statistical incremental parsing for LTAG-spinal. As far as we know, this parser is the first comprehensive attempt of efficient statistical parsing with a formal grammar with provably stronger generative power than CFG, supporting the full *adjoining* operation, dynamic predicate coordination, as well as non-projective dependencies [2].

2 LTAG-spinal and the Treebank

We first briefly describe the LTAG-spinal formalism and the LTAG-spinal treebank to be used in this paper. More details are reported in (Shen and Joshi, 2005).

In LTAG-spinal, we have two different kinds of elementary trees, *initial* trees and *auxiliary* trees, which are the same as in LTAG. However, as the name implies, an initial tree in LTAG-spinal only contains the *spine* from the root to the anchor, and an auxiliary tree only contains the spine and the foot node directly connected to a node on the spine.

Three types of operations are used to connect the elementary trees into a derivation tree, which are *attachment*, *adjunction* and *conjunction*. We show LTAG-spinal elementary trees and operations with an example in Figure 1.

In Figure 1, each arc is associated with a character which represents the type of operation. We use **T** for a*t*tach, **A** for *a*djoin, and **C** for *c*onjoin.

---

[1] Further formal results are described in (Shen and Joshi, 2005). There is also some relationship of LTAG-spinal to the spinal form context-free tree grammar, as in (Fujiyoshi and Kasai, 2000)

[2] In (Riezler et al., 2002), the MaxEnt model was used to rerank the K-best parses generated by a rule-based LFG parser.
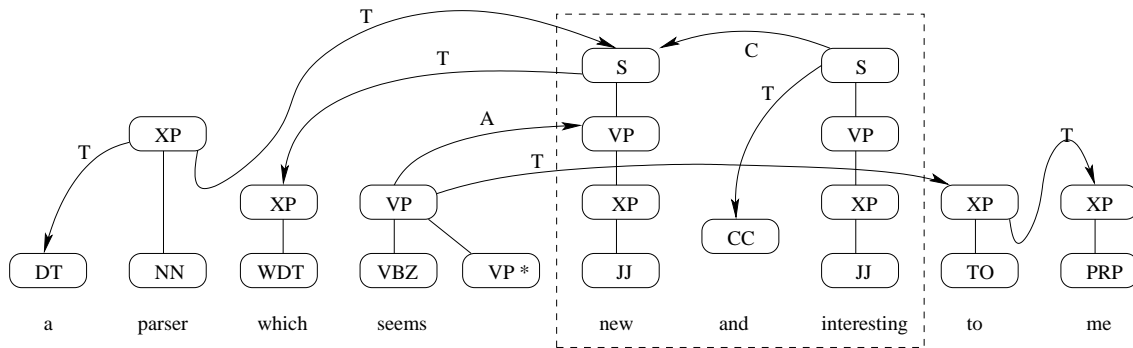
Figure 1: An example in LTAG-spinal. **A**=adjoin, **T**=attach, **C**=conjoin.

Attachment in LTAG-spinal is similar to *sister adjunction* (Chiang, 2000) in Tree Insertion Grammar (TIG) (Schabes and Waters, 1995). It represents a combination of *substitution* and *sister adjunction*. The *attachment* operation is designed to encode the ambiguity of an *argument* and an *adjunct*.

Adjunction inserts part of the spine and the foot node of an auxiliary tree into to the spine of another tree. The *adjunction* operation can effectively do wrapping, which distinguishes itself from *sister adjunction*. It is not difficult to see that adjunction only happens on the spine of a tree. This property will be exploited in the incremental parser.

Conjunction is similar to what was originally proposed in (Sarkar and Joshi, 1996). However, in LTAG-spinal, the conjunction operation is much easier to handle, since we only conjoin spinal elementary trees and we do not need to enumerate *contraction sets* for conjunction. In our formalization, *conjunction* can be treated as a special *adjunction*, however, this is beyond the scope of this paper.

We use the LTAG-spinal treebank described in (Shen and Joshi, 2005), which was extracted from the Penn Treebank (PTB) (Marcus et al., 1994) with Propbank (Palmer et al., 2005) annotations.

2.1 Relation to Traditional LTAG

LTAG-spinal preserves most of the strong generative power of LTAG. It can be shown that LTAG-spinal with adjoining restrictions (Joshi and Schabes, 1997) has stronger generative capacity as compared to CFG. For example, there exists an LTAG-spinal grammar that generates $\{a^n b^n c d^n e^n \mid n > 0\}$, which is not a context-free language.

A spinal elementary tree is smaller than a tradi-

tion LTAG elementary tree which contains all the substitution nodes of the arguments. In the LTAG-spinal formalism, both arguments and adjuncts are expected to be directly attached or adjoined onto a spine. In this sense, LTAG-spinal roughly satisfies *the fundamental TAG hypothesis: Every syntactic dependency is expressed locally within a single elementary tree* (Frank, 2002). The only difference is that, in LTAG-spinal, syntactic dependencies are represented via direct or local connections.

To better understand the meaning of this difference, we relate it to Frank's (2002) model for how the LTAG elementary trees are constructed. In Frank's model, all the elementary trees are built via *Marge* and *Move* operations, starting with a *local lexical array*. The resulting LTAG elementary trees are then combined with adjunction and substitution to build a derivation tree.

Thus, in a sense, the LTAG-spinal grammar opens a door to a *parallel* mechanism of building the elementary trees and the derivation tree. The spinal templates in LTAG-spinal only contain the path of projection from the anchor to the top node. A spinal template plus the root nodes of the subtrees attached to this template can be viewed as a traditional LTAG elementary tree. More specifically, it encodes a set of possible elementary trees if we distinguish substitution from sister adjunction. Thus, the LTAG-spinal parsing model to be proposed in Section 3 can be viewed as a parser at the meta-grammar (Candito, 1998; Kinyon and Prolo, 2002) level for traditional LTAG. Derivation tree construction and full-size elementary tree filtering are processed in parallel. Researches in statistical CFG parsing (Ratnaparkhi, 1997; Collins, 1999) and psycholinguistics

(Shieber and Johnson, 1993) showed that this strategy is desirable for NLP.

Furthermore, the way that we split a traditional LTAG elementary tree along the spine is similar to the method with which Evans and Weir (1997) compiled the XTAG English Grammar into finite state automata. In their work, this method was designed to employ shared structure in a rule-based parser. But here we extend this technique to statistical LTAG parsing.

2.2 Relation to Propbank

In building the LTAG-spinal Treebank, the Propbank information is used in the treebank extraction. As reported in (Shen and Joshi, 2005), tree transformation on PTB are employed to make it more compatible with the Propbank annotations. It was shown that 8 simple patterns of the path from a predicate to an argument account for 95.5% of the total pred-arg pairs. Thus, our high-quality parsing output will be very useful for semantic role labeling.

Arguments in Propbank are not *obligatory* complements. Therefore, we cannot treat the Propbank arguments as the *arguments* in LTAG. The ambiguity of argument and adjunct is reflected in the similarity of substitution and sister adjunction. This is one of the reasons that we do not distinguish substitution and sister adjunction in LTAG-spinal.

3 Incremental Parsing

We are especially interested in incremental parsing for the following two reasons. Firstly, the left to right strategy used in incremental parsing gives rise to a drastic boost in speed. Furthermore, there is also a strong connection between incremental parsing and psycholinguistics, and this connection is also observed in the LTAG formalism (Ferreira, 2000; Sturt and Lombardo, 2005).

In recent years, there have been many interesting works on incremental or semi-incremental parsing. By semi-incremental we mean the parsers that allow several rounds of left to right scans instead of one. Both left-corner strategy (Ratnaparkhi, 1997; Roark, 2001; Prolo, 2003; Henderson, 2003; Collins and Roark, 2004) and head-corner strategy (Henderson, 2000; Yamada and Matsumoto, 2003) were employed in incremental parsing. The head-corner

approach is more natural to the LTAG formalism (Evans and Weir, 1997). In our approach, we use a stack of derivation treelets to represent the partial parsing result. Furthermore, the LTAG formalism allows us to handle non-projectivity dependencies, which cannot be generated by a CFG or a Dependency parser.

In fact, the idea of incremental parsing with LTAG is closely related to the work on Supertagging (Joshi and Srinivas, 1994). A supertager first assigns the correct LTAG elementary tree to each word. Then a Lightweight Dependency Analyzer (LDA) (Srinivas, 1997) composes the whole derivation tree with these elementary trees. We use incremental parsing to incorporate supertager and LDA dynamically.

The model of incremental LTAG parsing is also similar to Structured Language Modeling (SLM) in (Chelba and Jelinek, 2000). In SLM, the left context of history is represented with a stack of binary trees. At each step, one computes the likelihood of the current word, its tag and the operations over the new context trees.

3.1 Treatment of Coordination

Predicate coordination appears in about 1/6 of the sentences in PTB, therefore proper treatment of coordination, especially predicate coordination, is important to parsing of PTB.

Some recent results in psycholinguistic experiments (Sturt and Lombardo, 2005) showed a high degree of *eagerness* in building coordination structures which is absent in a bottom-up approach; A bottom-up parser waits for the second conjunct to be completed before combining the two conjuncts as for example in VP coordination, and then combine the coordinated VP with the subject of the left conjunct. Psycholinguistic results suggest that the right conjunct has to have access to the subject NP of the left conjunct. This can be achieved by first building the entire S on the left and then *adjoining* the right VP conjunct to the VP node of the left conjunct (Sturt and Lombardo, 2005).

We follow the strategy suggested by the psycholinguistic experiments, treating conjoining as a special adjoining operation.

3.2 The Parsing Algorithm

There are four different types of operations in our parser. Three of them are described in Section 2. The fourth operation is **generation**, which is used to generate a possible spine for a given word according to the context and the lexicon.

Our left to right parsing algorithm is a variant of the shift-reduce algorithm with beam-search. We use a stack of disconnected derivation treelets to represent the left context. When the parser reads a word, it first *generates* a list of possible spinal elementary trees for this word, For each elementary tree, we first push it into the stack. Then we recursively pop the top two treelets from the stack and push the combined tree into the stack until we choose not to combine the top two treelets with one of the three combination operations (we can also choose not to pop anything at the beginning). Then we shift to the next word. This model is called the **Flex Model** in this paper.

A potential problem with the Flex Model is that a single *LTAG derivation* tree can be generated by several *shift-reduce derivation* steps, which only differ in the order of operations. For example, we have three trees A, B and C. In LTAG derivation, A adjoins to B, and B adjoins to C. Then we have two different shift-reduce derivations, which are $(A \rightarrow (B \rightarrow C))$ and $((A \rightarrow B) \rightarrow C)$.

Now we introduce the **Eager Model**, an eager evaluation strategy. Any two elementary trees which are directly connected in the LTAG derivation tree are combined immediately when they can be combined in some context. Furthermore, they cannot be combined afterwards, if they miss the first chance. In the previous example, the parser will generate $((A \rightarrow B) \rightarrow C)$, while $(A \rightarrow (B \rightarrow C))$ is ruled out. Then for each LTAG derivation tree, there exists a unique left-to-right derivation.

The Eager Model is motived by the treatment of coordination in (Sturt and Lombardo, 2005), as we discussed in the previous section. For example, we have the following two sentences.
1. Quimby knows Tom likes Philly steak.
2. Quimby knows Tom likes Philly steak and Jerry likes pizza.
Suppose we are parsing these two sentences, and for each case the current word is *likes*, the fourth word. Now we have just the same local contexts for both cases. According to the Eager Model, the parser takes the same action according to the context, which is to combine the *knows* tree and the *likes* tree. For sentence 2, the second *likes* tree will be conjoined with the first *likes* tree later. This is compatible with the psycholinguistic preference.

In the following section, we will explain the parsing mechanism for the Eager Model with an example. The Flex Model is similar except that the order of operations is flexible to some extent.

3.3 An Example

Figure 2 shows the left to right parsing of the phrase *a parser which seems new and interesting to me* with the Eager Model.

In Figure 2, each arc is associated with a number and a character. The number represents the order of operation, and the character stands for the type of operation as in Figure 1. Furthermore we use **G** to represent Generate.

In step 1 and 2, two disconnected spines are generated for *a* and *parser*. The spine for *a* is attached to the spine for *parser* on the *NP* node in step 3.

In step 6, the spine for *new*, the first conjunct of the predicate coordination, is generated. Then the auxiliary tree for *seems* is adjoined to the spine for *new* at the node *VP*. the latter is further combined with *which*, and is attached to the tree for *parser*.

In step 13, the conjoin operation is used to combine the treelet anchored on *new* and the treelet anchored on *interesting*. Alignments between the two spines are built, through which argument sharing is implemented in an implicit and underspecified way.

In step 15, for the spine for *to*, the visible nodes of the conjoined treelet include nodes on some auxiliary trees adjoined on the left of the spines, like the root *VP* node for *seems*. In this way, a non-projective structure is generated, which is just the same as the wrapping adjoining in LTAG.

3.4 Machine Learning Algorithm

Many machine learning algorithms have been successfully applied to parsing, incremental parsing, or shallow parsing (Ratnaparkhi, 1997; Punyakanok and Roth, 2001; Lafferty et al., 2001; Taskar et al., 2003), which can be applied to our incremental parsing algorithm.
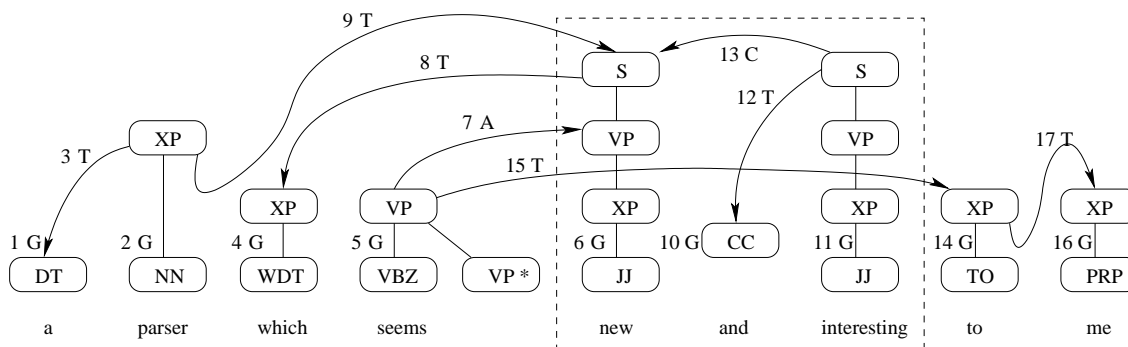
814

Figure 2: Incremental parsing with Eager Model. **A**=adjoin, **T**=attach, **C**=conjoin, **G**=generate

In this paper, we use the perceptron-like algorithm proposed in (Collins, 2002) which does not suffer from the label bias problem, and is fast in training. We also employ the voted perceptron algorithm (Freund and Schapire, 1999) and the *early update* technique as in (Collins and Roark, 2004).

3.5 Features

Features are defined in the format of (*operation, main spine, child spine, spine node, context*), where the *spine node* is the node on the *main spine* onto which the *child spine* is *attached* or *adjoined*. For *generate*, child spine and spine node are undefined, and for *conjoin* spine node is undefined. *context* describes the constituent label or lexical item associated with a certain node. The context of an operation includes the top two treelets involved in the operation as well as the two closest words on both sides of the current word.

- Context for *generate* :
  - The (-2, 2) window in the flat sentence.
  - The *visible* [3] spines on the topmost treelet.
- Context for *attach* and *adjoin* :
  - The (0, 2) window in the flat sentence.
  - The most recent spine previously attached or adjoined to the same location on the main spine.
  - The leftmost child spine attached to the child spines.
  - The spines that are *visible* before the operation and become *invisible* after the operation.
- Context for *conjoin* :
  - The (0, 2) window in the flat sentence.

- The leftmost child spine attached to the main spine, which is the first adjunct.
- The two leftmost children spines attached to the child spine, which is the current adjunct.

We have about 1.4M features extracted from the gold-standard parses, and about 600K features dynamically extracted from the generated parses in 10 rounds of training with the Eager Model.

4 Experiments and Analysis

We use the LTAG-spinal treebank reported in (Shen and Joshi, 2005). The LTAG-spinal parse for the 39434 sentences extracted from WSJ section 2-21 are used as the training data. Section 24 is used as the development data. Section 23 are used for test[4].

We use syntactic dependency for evaluation. It is worth mentioning that, for predicate coordination, we define the dependency on the parent of the coordination structure and each of the conjunct predicate. For example, in Figure 1, we have dependency relation on (parser, *new*) and (parser, *interesting*). Compared with other dependency parsers on PTB, the dependency defined on LTAG-spinal reveals deeper relations because of the treatment of traditional adjoining and predicate coordination described above.

In the community of parsing, *labeled recall* and *labeled precision* on phrase structures are often used for evaluation. However, in our experiments we cannot evaluate our parser with respect to the phrase structures in PTB. As shown in (Shen and Joshi, 2005), various irrecoverable tree transformations

---

[3]The details are presented in (Shen, 2005).

[4]The LTAG-spinal treebank contains 2401 out of 2416 sentences in section 23.
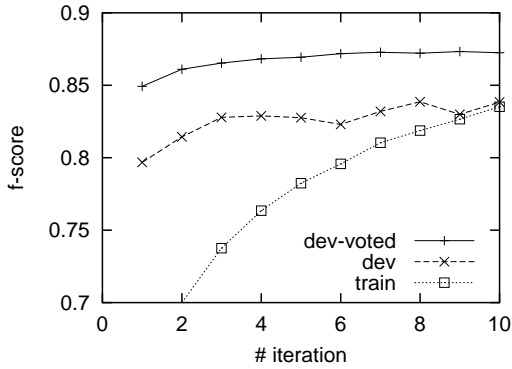
Figure 3: f-score of syntactic dependency on the *development* data with the Eager Model
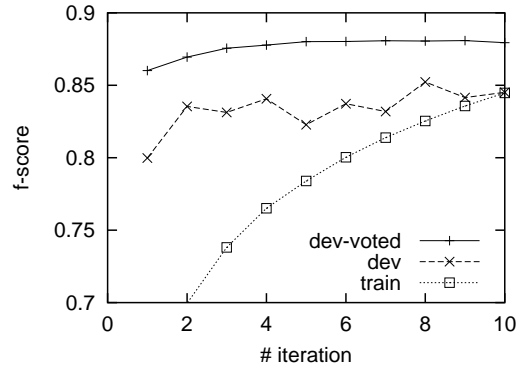

Figure 4: f-score of syntactic dependency on the *development* data with the Flex Model

were used to extract the LTAG-spinal treebank according the Propbank annotation on PTB. Therefore, we use syntactic dependency for evaluation.

4.1 Eager vs. Flex

We first train our incremental parser with Eager Model and Flex Model respectively. In the training, beam width is set to 10. Lexical features are limited to words appearing for at least 5 times in the training data. Figure 3 and Figure 4 show the learning curves on the training and the development data. The X axis represents the number of iterations of training, and the Y axis represents the f-score of dependency with respected to the LTAG derivation tree.

Since *early update* is used, the f-score on the training data is very low at the beginning. In both cases, the voted weights provide an f-score which is more than 3% higher. The voted results converge faster and are more stable. The result with Flex Model is 0.6% higher than the one with Eager Model, but the parsing time is much longer with Flex Model as we will show later.

We use the voted weights obtained after 10 rounds of iteration for the evaluation on the test data. We achieve an f-score of **88.7%** on dependency with the Eager Model, and **89.3%** with the Flex Model. The Flex Model achieves better performance because it allows the decision of operation to be delayed until there is enough context information.

4.2 K-Best Parsing

The next experiment is on K-best parsing. As a first attempt, we just use the same algorithm as in the pre-

Table 1: F-score of the oracle parse in the 10-best parses on the *development* data with the Eager Model

algorithm	f-score%
top (eager)	87.3
oracle (eager)	88.5
top (eager+combined parses)	87.4
oracle (eager+combined parses)	91.0

vious section, except that we study the oracle parse, or the best parse, among the top 10 parses. The f-score on the oracle in top 10 in the development data is 88.5%, while the f-score of the top candidate is 87.3%, as shown in Table 1. However, we are not satisfied with the score on oracle, which is not good enough for post-processing, i.e. parse reranking.

We notice that from a single partial derivation we can generate a large set of different partial derivations, just by combining the elementary tree of the next word. It is easy to see that these similar derivations may use up the search beam quickly, which is not good for parse search. Many of the new derivations share the same dependency structure. So we revised our learning procedure by combining derivations with the same dependency structure before each shift operation. We repeated the K-best parsing experiments by using **Combined Parses** as described above, and achieved significant improvement on the oracle, as shown in Table 1.

Figure 5 shows the f-score of the oracle on K-best parsing using combined parses on the test data. For each K-best oracle test, we set the beam width to K
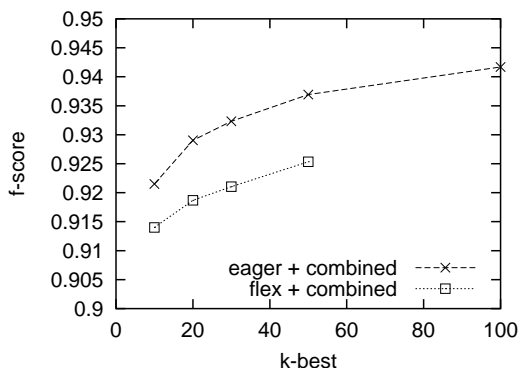
Figure 5: f-score of the oracle on the *test* data

Table 2: Speed of parsing on the *test* data set. Here cp? = whether the method of Combined Parses is used; sen/sec = sentence per second; top = top candidate given by the parser; oracle = oracle of the K-best parses where K equals the width of the beam.

model	cp?	beam	sen/sec	f-score%
single best				top
flex	no	10	0.37	89.3
eager	no	10	0.79	88.7
K-best				oracle
eager	yes	10	0.62	92.2
eager	yes	20	0.31	92.9
eager	yes	30	0.22	93.2
eager	yes	50	0.13	93.7
eager	yes	100	0.07	94.2

in parsing. The f-score of oracle in 100-best parsing is **94.2%** with the Eager Model + Combined Parses.

4.3 Speed of Parsing

Efficiency is important to the application of incremental parsing. This set of experiments is related to the speed of our parser on single best and K-best parsing with both the Eager Model and the Flex Model. All the experiments are performed on a Linux node with two 1.13GHz PIII CPUs and 2GB RAM. The parser is coded in Java.

Table 2 shows that the Eager Model is more than two times faster than the Flex Model, as we expected. The time spent on K-best parsing is proportional to the beam width.

5 Discussion and Future Work

The parser proposed in this paper is an incremental parser, so the accuracy on dependency is lower than that for chart parsers, for example like those reported in (Collins, 1999; Charniak, 2000). [5] However, it should be noted that the dependencies computed by our parser are *deeper* than those calculated by parsers working directly on PTB. This is due to the treatment of adjunction and coordination.

On the other hand, the LTAG-spinal treebank used in this paper shows a high degree of compatibility with the Propbank, as shown in (Shen and Joshi, 2005), so the LTAG derivations given by the parser are very useful for predicate-argument recognition. We plan to improve the parsing performance by reranking and extend our work to semantic parsing (Mooney, 2004).

Another interesting topic is whether this parser can be applied to languages which have various long-distance scrambling, as in German. It appears that by carefully modifying the definition of *visible* spines, we can represent scrambling structures, which at present can only be represented by Multi-Component TAG (Becker et al., 1991).

6 Conclusions

In this paper, we present an efficient incremental parser for LTAG-spinal, a variant of LTAG which is both linguistically and psycholinguistically motivated. As far as we know, the statistical incremental parser proposed in this paper is the first comprehensive attempt of efficient statistical parsing with a formal grammar with provably stronger generative power than CFG, supporting the *adjoining* operation, dynamic predicate coordination, as well as non-projective dependencies.

We have trained and tested our parser on the LTAG-spinal treebank, extracted from the Penn Treebank with Propbank annotation, Using gold standard POS tags as part of the input, the parser achieves an f-score of 89.3% for syntactic dependency on section 23 of PTB. Because of the treatment of adjunction and predicate coordination, These dependencies, which are defined on LTAG-spinal derivation trees, are deeper than the dependencies extracted from PTB alone with head rules.

---

[5] We plan to work on a chart parser for LTAG-spinal.

References

T. Becker, A. K. Joshi, and O. Rambow. 1991. Long distance scrambling and Tree Adjoining Grammars. In *EACL 1991*.

M. Candito. 1998. Building parallel ltag for french and italian. In *ACL-COLING 1998*.

E. Charniak. 2000. A maximum-entropy-inspired parser. In *NAACL 2000*.

C. Chelba and F. Jelinek. 2000. Structured language modeling. *Computer Speech and Language*, 14(4):283–332.

D. Chiang. 2000. Statistical Parsing with an Automatically-Extracted Tree Adjoining Grammar. In *ACL 2000*.

M. Collins and B. Roark. 2004. Incremental parsing with the perceptron algorithm. In *ACL 2004*.

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 *EMNLP 2002*.

R. Evans and D. Weir. 1997. Automaton-based parsing for lexicalized grammars. In *IWPT 1997*.

F. Ferreira. 2000. Syntax in language production: An approach using tree-adjoining grammars. In L. Wheeldon, editor, *Aspects of Language Production*. MIT Press.

R. Frank. 2002. *Phrase Structure Composition and Syntactic Dependencies*. MIT Press.

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

A. Fujiyoshi and T. Kasai. 2000. Spinal-formed context-free tree grammars. *Theory Computing Systems*, 33(1).

J. Henderson. 2000. A neural network parser that handles sparse data. In *IWPT 2000*.

J. Henderson. 2003. Generative versus discriminative models for statistical left-corner parsing. In *IWPT 2003*.

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.

A. K. Joshi and B. Srinivas. 1994. Disambiguation of super parts of speech (or supertags): Almost parsing. In *COLING 1994*.

A. Kinyon and C. Prolo. 2002. A classification of grammar development strategies. In *COLING 2002 Workshop: Grammar Engineering and Evaluation*.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML 2001*.

A. Lavelli and G. Satta. 1991. Bidirectional parsing of lexicalized tree adjoining grammars. In *EACL 1991*.

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.

R. Mooney. 2004. Learning semantic parsers: An important but under-studied problem. In *AAAI 2004 Spring Symposium on Language Learning: An Interdisciplinary Perspective*.

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

C. Prolo. 2003. *LR Parsing for Tree Adjoining Grammars and its Application to Corpus-based Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

V. Punyakanok and D. Roth. 2001. The use of classifiers in sequential inference. In *NIPS 2001*.

A. Ratnaparkhi. 1997. A linear observed time statistical parser based on maximum entropy models. In *EMNLP 1997*.

S. Riezler, T. King, R. Kaplan, R. Crouch, J. Maxwell, and M. Johnson. 2002. Parsing the wall street journal using a lexical-functional grammar and discriminative estimation techniques. In *ACL 2002*.

B. Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2):249–276.

A. Sarkar and A. K. Joshi. 1996. Coordination in tree adjoining grammars. In *COLING 1996*.

Y. Schabes and R. C. Waters. 1995. A cubic-time, parsable formalism that lexicalizes context-free grammar without changing the trees produced. *Computational Linguistics*, 21(4).

L. Shen and A. K. Joshi. 2005. Building an LTAG treebank. Technical Report MS-CIS-05-15, CIS Dept., UPenn.

L. Shen. 2005. Statistical Natural Language Processing with Lexicalized Tree Adjoining Grammar. Ph.D. proposal, University of Pennsylvania.

S. Shieber and M. Johnson. 1993. Variations on incremental interpretation. *Journal of Psycholinguistic Research*, 22(2):287–318.

B. Srinivas. 1997. Performance evaluation of supertagging for partial parsing. In *IWPT 1997*.

P. Sturt and V. Lombardo. 2005. Processing coordinated structures: Incrementality and connectedness. *Cognitive Science, to appear*.

B. Taskar, C. Guestrin, and D. Koller. 2003. Max-margin markov networks. In *NIPS 2003*.

G. van Noord. 1994. Head corner parsing for TAG. *Computational Intelligence*, 10(4).

K. Vijay-Shanker and A. K. Joshi. 1985. Some computational properties of tree adjoining grammars. In *ACL 1985*.

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