Lang Resources & Evaluation (2008) 42:1–19
DOI 10.1007/s10579-007-9043-7

LTAG-spinal and the Treebank

A new resource for incremental, dependency and semantic parsing

Libin Shen · Lucas Champollion · Aravind K. Joshi

Published online: 20 October 2007

Abstract We introduce LTAG-spinal, a novel variant of traditional Lexicalized Tree Adjoining Grammar (LTAG) with desirable linguistic, computational and statistical properties. Unlike in traditional LTAG, subcategorization frames and the argument–adjunct distinction are left underspecified in LTAG-spinal. LTAG-spinal with adjunction constraints is weakly equivalent to LTAG. The LTAG-spinal formalism is used to extract an LTAG-spinal Treebank from the Penn Treebank with Propbank annotation. Based on Propbank annotation, predicate coordination and LTAG adjunction structures are successfully extracted. The LTAG-spinal Treebank makes explicit semantic relations that are implicit or absent from the original PTB. LTAG-spinal provides a very desirable resource for statistical LTAG parsing, incremental parsing, dependency parsing, and semantic parsing. This treebank has been successfully used to train an incremental LTAG-spinal parser and a bidirectional LTAG dependency parser.

Keywords Tree Adjoining Grammar · LTAG-spinal · Treebank · Dependency parsing

L. Shen (✉)
BBN Technologies, 10 Moulton Street, Cambridge, MA 02138, USA
e-mail: lshen@bbn.com

L. Champollion
Department of Linguistics, University of Pennsylvania, 619 Williams Hall, Philadelphia, PA 19104, USA
e-mail: champoll@ling.upenn.edu

A. K. Joshi
Department of Computer and Information Science, University of Pennsylvania, 3330 Walnut Street, Philadelphia, PA 19104, USA
e-mail: joshi@seas.upenn.edu

Abbreviation

LTAG	Lexicalized Tree Adjoining Grammar

1 Introduction

Lexicalized Tree Adjoining Grammar (LTAG) (Joshi and Schabes 1997) has attractive properties from the point of view of Natural Language Processing (NLP). LTAG has appropriate generative capacity (LTAG languages belong to the class of mildly context-sensitive languages) and a strong linguistic foundation.

In this article, we introduce LTAG-spinal, a variant of LTAG with very desirable linguistic, computational, and statistical properties. LTAG-spinal with adjunction constraints is weakly equivalent to traditional LTAG.

We first provide a brief introduction of LTAG in Sect. 1.1. In Sect. 1.2, we describe the motivation for the LTAG-spinal formalism. In Sect. 2, we introduce the definition of LTAG-spinal. Then we describe the process of extracting an LTAG-spinal Treebank from the Penn Treebank (PTB) (Marcus et al. 1994), together with Propbank annotation (Palmer et al. 2005) in Sect. 3. We illustrate the extracted LTAG-spinal Treebank and its treatment of certain syntactic phenomena of linguistic interest in Sect. 4. We also present the statistical properties of the LTAG-spinal Treebank in Sect. 5, especially the compatibility with the Propbank. We discuss our conclusions and future work in Sect. 6.

1.1 Lexicalized Tree Adjoining Grammar

Tree Adjoining Grammar (TAG) was first introduced in (Joshi et al. 1975). A recent review of TAG is given in (Abeillé and Rambow 2001), which provides a detailed description of TAG with respect to linguistic, formal, and computational properties (see also Frank 2002). In this section, we briefly describe the TAG formalism and its relation to linguistics.

In traditional lexicalized TAG, each word is associated with a set of *elementary trees*, or e-trees for short. Each e-tree represents a possible tree structure for the word.

There are two kinds of e-trees, *initial trees* and *auxiliary trees*. A derivation always starts with an initial tree. Auxiliary trees must have a *foot node*, a leaf node whose label is identical to the label of the root. E-trees can be combined through two operations, *substitution* and *adjunction*. Substitution is used to attach an initial tree α into a *substitution slot* of a host tree α'. Substitution slots are specially marked leaf nodes whose label must be identical with the root of α. Adjunction is used to attach an auxiliary tree α to a node n of a host tree α'. n must carry the same label as the root and foot nodes of α. Adjunction is carried out by replacing the node n with the entire tree α. The foot node of α is then replaced by the subtree under n.

The tree resulting from the combination of e-trees is called a *derived tree*. We can record the history of a derivation by building a *derivation tree*, in which every e-tree used in the derivation is represented by a single node and every operation by a single arc, whose parent is the host tree of the operation.

1.2 Motivation for LTAG-spinal

For the purpose of statistical processing, we need a large scale LTAG style treebank. As far as automatic treebank extraction and statistical processing is concerned, a variant of traditional LTAG, namely LTAG-spinal, turns out to be more attractive. We now illustrate these two aspects in turn.

1.2.1 LTAG Treebank Extraction

LTAG encodes the subcategorization frames of predicates explicitly by modeling each predicate as an e-tree that contains substitution slots for (obligatory) arguments but not for (optional) adjuncts. Predicates with more than one subcategorization frame are represented with multiple e-trees.

In previous work of LTAG treebank extraction (Xia 2001; Chen et al. 2006), heuristic rules were used to distinguish arguments from adjuncts. However, e-trees extracted in this way are different from the e-trees of a handcrafted LTAG grammar, e.g., the XTAG English grammar (XTAG-Group 2001). It turns out to be a non-trivial task to map the automatically generated templates to those in the XTAG grammar. One extracted e-tree can be mapped to several XTAG e-trees which differ in their feature structures. It is difficult to obtain this information from the original resources.

Therefore, we desire a framework in which the representations for arguments and adjuncts are similar. In this way, we can encode the ambiguity with a single structure, and leave the disambiguation for further processing.

Our solution is a *sister adjunction* like operation. Sister adjunction was previously proposed to represent adjuncts in Chiang (2000) for Tree Insertion Grammars (TIG) (Schabes and Waters 1995), as well as in D-Tree substitution grammars (Rambow et al. 2001). We call our operation *attachment* (see below for a definition). We use attachment both for arguments and for non-predicate adjuncts,[1] thereby encoding argument–adjunct ambiguity.

The *extended domain of locality* (EDL) (Joshi and Schabes 1997) of LTAG is still retained in the sense that syntactically dependent arguments are directly attached to the predicate. By domain of locality, we mean a domain over which various kinds of syntactic dependencies can be specified. In traditional LTAG, EDL is expressed in terms of hard constraints via the structure of e-trees representing *extended* projections of lexical items. In our presentation, EDL is expressed in terms of soft constraints, in particular in terms of the distributions of argument and adjunct attachment operations.

As a result, our e-trees are in the so-called spinal form since arguments do not appear in the e-tree of the predicate.

[1] By *non-predicate* adjuncts, we mean those auxiliary trees whose foot node does not subcategorize for the anchor; these are essentially modifier trees. LTAG also uses auxiliary trees to model phenomena other than non-predicate adjuncts. Examples are raising verbs and parentheticals. In going from LTAG to LTAG-spinal, we do not change the analysis of these phenomena. See Sect. 4 for further discussion.

1.2.2 Statistical processing

The complexity of using automatically extracted LTAG templates in parsing is greatly increased due to increased local ambiguity (i.e., the average number of e-trees per word). According to the *coarse to fine* approach (Charniak and Johnson 2005), it is attractive to use some structure to encode these templates, so as to make the search space more tractable at each step of parsing.

The LTAG-spinal formalism, which we formally introduce in the next section, substantially reduces the local ambiguity. For example, the e-tree of a transitive verb and the e-tree of a ditransitive verb have identical spines from the *S* node to the *V* node. In parsing, when we encounter a predicate of a given sentence, we do not need to guess its subcategorization frame immediately. Instead, we use the spinal form to represent a verb without its subcategorization frame. We defer identifying the correct subcategorization frames to a later stage in the processing chain, when enough contextual information becomes available, in a way similar to Charniak (1997) and Collins (1999).

To sum up, the key reasons that lead us to adopt the LTAG-spinal framework are these: Unlike traditional LTAG, LTAG-spinal does not encode the argument–adjunct distinction explicitly, which makes it easier to automatically convert the PTB to LTAG-spinal format. LTAG-spinal trees generalize over predicates with different subcategorization frames, which follows the *coarse to fine* spirit and alleviates the sparse data problem for parsing. In particular, the parser is not forced to make a decision on subcategorization without enough contextual information.

2 Formalism

In LTAG-spinal, just as in traditional LTAG, we have two kinds of e-trees, initial trees and auxiliary trees (see Fig. 1). What makes LTAG-spinal novel is that e-trees are in the spinal form. A spinal initial tree is composed of a *lexical spine* from the root to the anchor, and nothing else. A spinal auxiliary tree is composed of a lexical spine and a *recursive spine* from the root to the foot node. For example, in Fig. 1, the lexical spine for the auxiliary tree is $B_1, ..., B_i, ..., B_n$, the recursive spine is $B_1, ..., B_i, ..., B_1^*$.

There are two operations in LTAG-spinal, namely, *adjunction* and *attachment*. *Adjunction* in LTAG-spinal is the same as in traditional LTAG (see Sect. 1.1). To *attach* an initial tree α to a node n of another tree α', we add the root of α to n as a new child. Unlike in the substitution operation, α' need not have a *substitution slot* that subcategorizes for the root of α. Attachment applies to initial trees only, and adjunction applies to auxiliary trees only.

Attachment can be modeled as a special case of adjunction. We can add artificial root and foot nodes to an initial tree to build an auxiliary tree, and simulate the attachment of an initial tree by a (non-wrapping) adjunction of the artificial auxiliary tree, as in TIG. On the other hand, attachment is similar to substitution in that, unlike adjunction, it can not generate any non-projective dependencies.

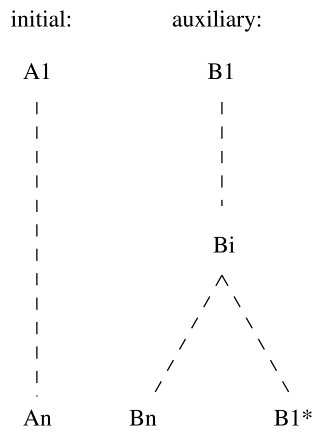

Fig. 1 Spinal e-trees

However, the flexibility of attachment can be constrained by null (NA), obligatory (OA) and selective (SA) attachment constraints analogous to adjunction constraints in traditional LTAG (Joshi and Schabes 1997). With these constraints, LTAG-spinal is weakly equivalent to traditional LTAG. A detailed proof is given in (Shen 2006).

As for the LTAG-spinal Treebank described in this article, we do not use the three hard constraints described above, which means that the predicate e-trees do not contain slots for their arguments. In our data oriented approach, the constraints are represented in a soft way via statistics. In other words, even ungrammatical sentences receive a (low probability) parse. However, this does not represent a theoretical commitment on our part. As the weak equivalence with LTAG shows, it is perfectly possible to write an LTAG-spinal grammar that assigns no structure to ungrammatical sentences.

An example of LTAG-spinal derivation trees is shown in Fig. 2. Each arc is associated with a label which represents the type of operation. We use *att* for attach and *adj* for adjoin.

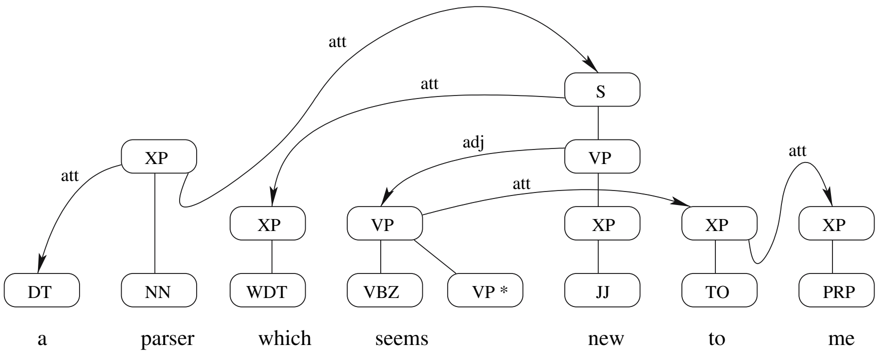

Fig. 2 An example of an LTAG-spinal derivation

In Fig. 2, *seems* adjoins to *new* as a *wrapping adjunction*, which means that the leaf nodes of the adjunct subtree appear on both sides of the anchor of the main e-tree in the resulting derived tree. Here, *seems* is to the left of *new* and *to me* is to the right of *new*. Wrapping adjunction allows us to describe non-projective dependencies. In this case, the dependency between *to* and *seems* is non-projective. It should be noted that attachment/sister adjunction does not allow wrapping structures like this one.

3 Extracting an LTAG-spinal Treebank

3.1 Previous work

For the purpose of statistical processing, many attempts have been made for automatic construction of LTAG treebanks. Joshi and Srinivas (1994) presented a supertag corpus extracted from the Penn Treebank with heuristic rules. However, due to certain limitations of the supertag extraction algorithm, the extracted supertags of the words in a sentence cannot always be successfully put together. Xia (2001) and Chen et al. (2006) described deterministic systems that extract LTAG-style grammars from the PTB. In their systems, a *head table* in Magerman's style (1995) and the PTB functional tags were used to resolve ambiguities in extraction. Chiang (2000) reported a similar method of extracting an LTAG treebank from the PTB, and used it in a statistical parser for Tree Insertion Grammar.

3.2 Our approach

We automatically extracted an LTAG-spinal Treebank from the PTB together with Propbank annotation. The following two properties make our extracted treebank different from previous work: incorporation of Propbank information and treatment of coordination. In this section, we discuss each of these properties in turn and then describe our extraction algorithm.

3.2.1 Propbank guided extraction

Propbank provides annotation of predicate–argument structures and semantic roles on the Penn Treebank, and was unavailable to most of the previous LTAG treebank extraction systems.[2]

There is an obvious connection between Propbank argument sets and e-trees in LTAG. Therefore, one of the goals of our work is to incorporate Propbank annotation into the extracted LTAG-spinal Treebank. In this way, the extracted e-trees for each lexical anchor (predicate) will become semantically relevant. At the

[2] Most recently, subsets of the PTB and Propbank have been reconciliated by hand (Babko-Malaya et al. 2006; Yi 2007). Our own extraction process was carried out automatically before that data became available and covers the entire PTB and Propbank. To a certain extent, it has been informed by that ongoing work.

same time, as explained below, Propbank provides syntactic information that helps us successfully extract various structures of interest.

In Chen and Rambow (2003), in a procedure called *head filtering*, a head table was used as a first step to recognize the head constituent for each phrase. Propbank annotation was then used to distinguish arguments from adjuncts, the second step of the extraction procedure. We employ Propbank annotation as early as the head filtering step. This turns out to be helpful for recognizing structures that are hard to discover with a head table. For example, Propbank annotation on discontinuous arguments helps us recognize auxiliary trees.

Example 1 (the market could)$_{\mathrm{arg1.1}}$ (continue)$_{\mathrm{Pred}}$ (to soften)$_{\mathrm{arg1.2}}$ in the months ahead.

Example 1 illustrates raising. The Propbank annotation tells us that *the market could ... to soften* is ARG1 of *continue*. Unlike Propbank, the PTB does not distinguish raising from control. Based on the Propbank information, we can avoid mistakenly taking *the market* by itself as an argument of *continue*, as we would want to do if this was a control structure. (The extracted tree is shown in Fig. 5.)

3.2.2 Treatment of predicate coordination

Predicate coordination structures such as VP coordination and Right Node Raising can be seen as predicates either sharing or dropping some of their arguments. Traditional LTAG's notion of locality requires each predicate to carry its complete subcategorization frame with it "hard-coded" as a part of its elementary tree. For this reason, an account of coordination cannot easily be given in traditional LTAG. Previous work has suggested contracting shared substitution slots (Sarkar and Joshi 1996). That approach extends the notion of a derivation tree to an acyclic derivation graph. Alternatively, it can be viewed as transforming the e-trees of some of the conjuncts (e.g., the right conjunct in VP coordination) into auxiliary e-trees lacking some arguments.

LTAG-spinal does not share traditional LTAG's notion of fixed constituency, so representing predicate coordination becomes much easier. Predicate e-trees do not contain slots for their arguments, so we do not have to transform them.

In the LTAG-spinal Treebank, predicate coordination is represented with a special structure. We *conjoin* two spinal e-trees, which are of the same category, as shown in Fig. 3. We conjoin *interesting* onto *new*, and obtain a *coordination* structure, which is represented as a box in the figure. Here *conjoining* is a special operation to build predicate coordination structures incrementally.[3]

This method is more flexible than the well-known treatment of coordination in Combinatory Categorial Grammar (CCG) (Steedman 2000) and the CCG treebank (Hockenmaier and Steedman 2002). In CCG, the conjuncts of the same category are

[3] We treat conjoining as if it were a distinct operation. Theoretically, though, conjoining can be seen as a special case of the attachment operation. This is somewhat similar to traditional LTAG, where substitution is a distinct operation but can be seen as a special case of adjunction. Indeed, historically the first definition of TAG does not refer to substitution at all (Joshi et al. 1975).

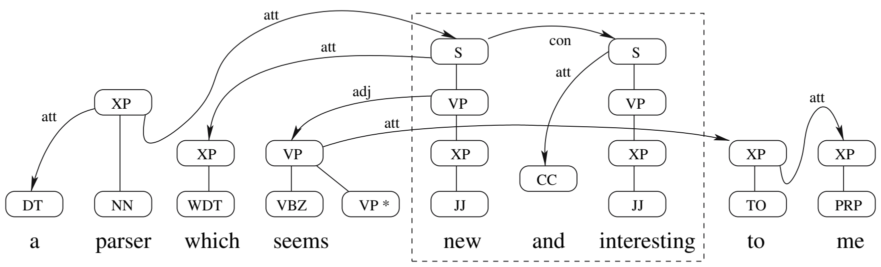

Fig. 3 An example of conjoining in LTAG-spinal

combined first, and then combined with the shared arguments. In our approach, we do not need to combine the conjuncts first as in CCG.

In Sturt and Lombardo (2005), it is shown that a combination order other than that of CCG's is more preferable from the viewpoint of psycholinguistics and incremental parsing. In their order, a complete sentence structure is first built using the first conjunct, and then the second conjunct is introduced into the derivation. Our formalism is flexible enough to accommodate either order. For example, in Fig. 3, we could either conjoin *interesting* to *new* first as in CCG, or attach *new* to *parser* first as in Sturt and Lombardo (2005). The order of operations for predicate coordination is flexible in LTAG-spinal.

In traditional LTAG, constituency is fixed once the e-trees are defined. Any continuous string generated in LTAG always has a semantic type, which can be read off from the derived tree built so far. It is not required that there be a single constituent dominating just that string. As for LTAG-spinal, e-trees are in the spinal form, so that we could easily employ underspecification of argument sharing. In this way, representation of predicate coordination becomes even simpler.

The example in Fig. 3 illustrates adjective coordination. In the treebank, S and VP coordination and right node raising are represented in a similar way. As for gapping, we have not pursued the question of how to represent it in LTAG-spinal, mainly because the traces of the gapped predicates are not annotated in PTB and so we could not extract this information.

3.2.3 Extraction algorithm

We now describe the algorithm that we have used to extract the LTAG-spinal Treebank from the PTB with Propbank annotation. We use a rule-based method for treebank extraction. We take a PTB tree as an LTAG *derived* tree. The algorithm is implemented in several rounds of tree traversal. Each round is implemented with a recursive function over trees. Therefore, whenever possible, we try to divide different operations into different rounds so as to simplify the implementation. The following is a list of the steps for extraction.

1. We first automatically generate the annotation for *be*, since the latest release of Propbank does not provide annotation for the verb *be*.

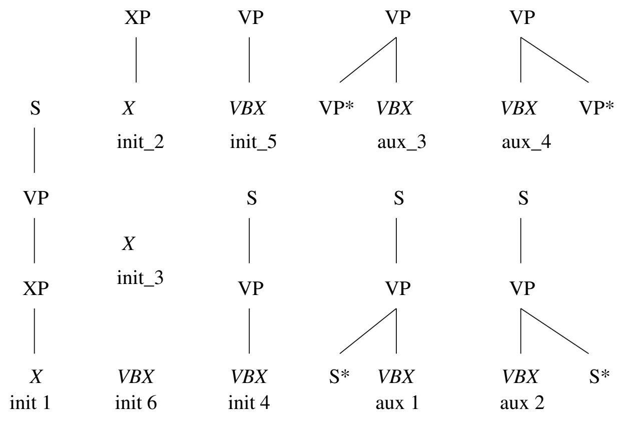

Fig. 4 Types of normalized spinal e-trees

2. Then we reconcile PTB and Propbank by tree transformations on PTB trees to make them compatible with Propbank annotations.[4]
3. We recognize LTAG predicate adjunction and predicate coordination in the PTB with respect to Propbank annotations.
   The recognition of predicate adjunction is based on discontinuous arguments as shown in the example in Sect. 3.2.1. Auxiliary trees are extracted for raising verbs, Exceptional Case Marking (ECM) Verbs and predicate parentheticals. In all other cases, auxiliary trees are mapped to initial trees (see step 5). The resulting structures are shown in Sect. 4.
   Predicate coordination is detected if there are several predicates whose arguments are under the same lowest dominating node, and there exist connectives between each pair of adjacent predicates in the sentence.
   For both predicate adjunction and predicate coordination, we transform the PTB tree by cutting it into segments and reconnecting them with the LTAG derivation operations, i.e., attachment, adjunction, and coordination. For each connected tree segment, we simply use head information to recursively recognize the LTAG derivation tree and e-trees that generate the segment.
4. Then we extract LTAG e-trees from the transformed PTB subtrees recursively, with respect to Propbank annotations for predicates and a head table for all other constituents.
5. At the end, we map all of the e-trees into a small set of normalized e-trees, as shown in Fig. 4. For example, an e-tree (S (S (VP VB))) with duplicated S nodes is normalized to (S (VP VB)). Phrasal projections (NP, PP, etc.) are all mapped to "XP" since this information is already encoded in the POS tags.

We map a non-predicate auxiliary tree to an initial tree by removing its foot node and root node. As a result, we have only six different kinds of initial trees

[4] Detailed operations for tree transformations were described in Shen (2006). Similar work was reported in Babko-Malaya et al. (2006) and Yi (2007).

(three for verbs and three for non-verbs) and four different kinds of full auxiliary trees. In Fig. 4, *VBX* represents a verbal POS tag, and *X* represents a non-verbal POS tag.

4 The LTAG-spinal Treebank

In this section, we focus on linguistic phenomena of special interest to us. Some are difficult to represent with CFG, but are easy with TAG thanks to the use of adjunction, such as raising verbs (i.e., continue, begin, etc.), Exceptional Case Marking (ECM) verbs (i.e., expect, believe, etc.), and parentheticals. Some are important in order to make the parsing output useful, such as the treatment of relative clauses as well as predicative nominals and adjectives.[5]

The figures used in this section were generated with the graphical interface of our treebank API (see Sect. 6). In the figures, solid lines are used within e-tree spines, and dotted arrows are used between e-trees. Auxiliary trees are recognizable by their footnodes, which are marked with an asterisk. Empty elements (*-1, *t*-48, etc.) are carried over from the PTB into the LTAG-spinal Treebank.[6] For convenience, the root node of every e-tree is annotated with the span that this tree and its descendants cover in the string.

4.1 Raising verbs and passive ECM verbs

In the LTAG-spinal Treebank, raising verbs and passive ECM verbs are associated with an auxiliary tree. For example, in Fig. 5, the e-tree for *continue* adjoins onto the S node of the e-tree for *soften*. Furthermore, *in* attaches to *continue*. Since *soften* is between *continue* and *in* in the flat sentence, this is a case of a non-projective dependency. We use the adjoining operation to distinguish raising verbs from control verbs.

4.2 Relative clauses

In the LTAG-spinal Treebank, a relative clause is represented by attaching the predicate of the clause to the head of the phrase that it modifies. For example, in Fig. 6, the predicate of the relative clause is *shown*. *which* attaches onto *shown*, and *shown* attaches onto *earnings*.

[5] For a general reference for the use of LTAGs for linguistic description, see Frank (2002).

[6] Coindexation information is not maintained in the trees because Propbank can be used to recover it. We have included these traces in the LTAG-spinal treebank to record the annotation decisions of the PTB. We do not attach any theoretical significance to these traces and provide them for informational purposes only. If this information is not needed, a *purely lexicalized* version of our treebank can be easily obtained by stripping off the e-trees anchored in traces.

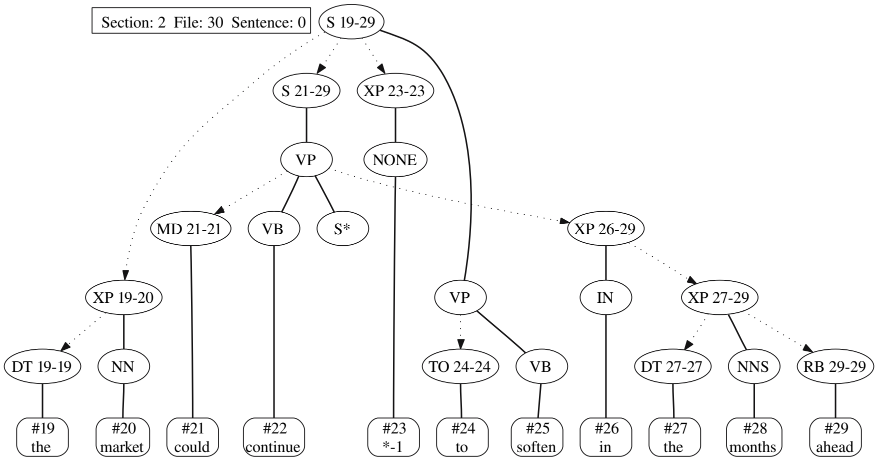

Fig. 5 The market could continue to soften in the months ahead

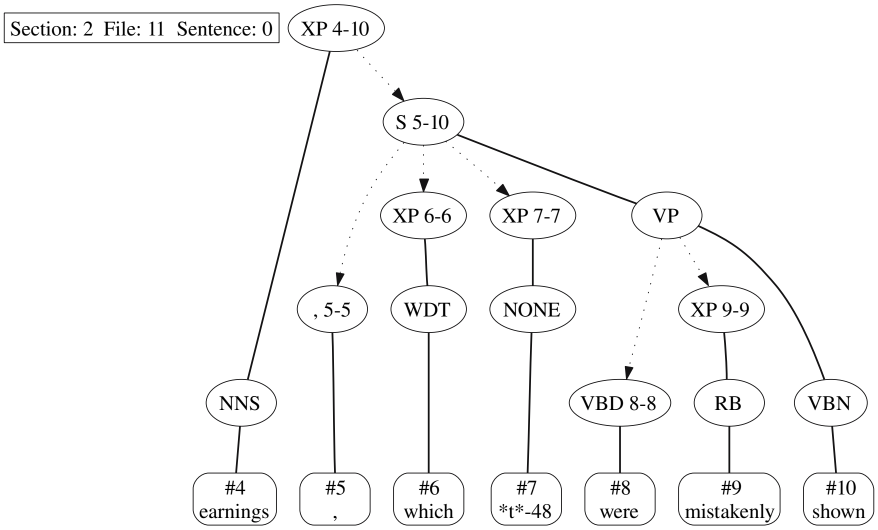

Fig. 6 ... earnings, which were mistakenly shown ...

4.3 Parentheticals

In the LTAG-spinal treebank, parentheticals containing a predicate, such as “Mr. Green testified”, are treated using adjunction. This predicate adjoins into the verb of the clause that contains the parenthetical. The argument structure of that clause is not disrupted by the presence of the parenthetical. For example, in Fig. 7, *testified* adjoins into *began* from left. Arguments and adjuncts of *began* are attached directly

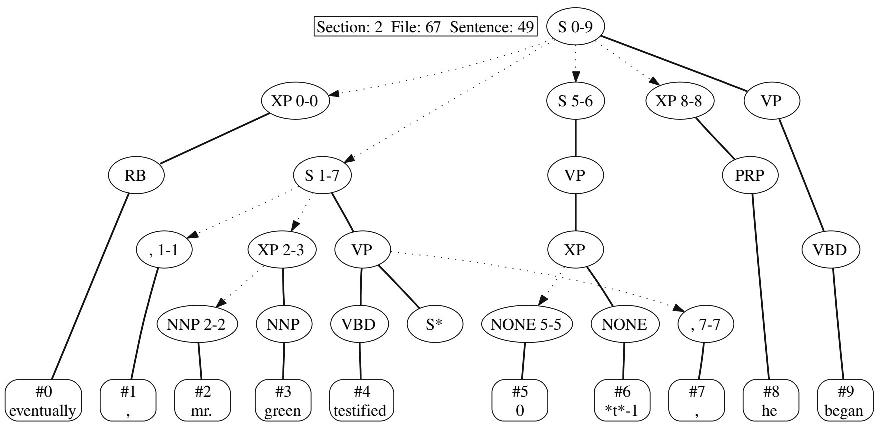

Fig. 7 Eventually, Mr Green testified, he began ...

to *began*, even when they are separated from it by the parenthetical, as is the case with *eventually*.

4.4 Predicative trees

In the current version of the LTAG-spinal Treebank, most of the predicate nominals and adjectives are not annotated as the head predicate. Instead, in order to avoid propagating potential errors, we treat the copula as the head of the sentence. For example, in Fig. 8, *writer* attaches to *is*.

We are aware that, in the XTAG English grammar, predicate nominals and adjectives are regarded as the head. Our differing treatment is due to the difficulty in finding the head of a noun phrase. In the PTB, NP representation is flat (Vadas and Curran 2007), so that it is non-trivial to recognize coordination at the NP level automatically. For example, the NP *those workers and managers* and the NP *the US sales and marketing arm* are both represented as flat NPs.

Furthermore, appositives and NP lists are represented in the same way. The problem of distinguishing NP coordination from coordination within an NP results in the difficulty of choosing the head of NPs.

4.5 Extraposition

Extraposition is a class of dependencies that cannot be represented with traditional LTAG.[7] It is also a problem for the LTAG-spinal formalism. For the sentence in Fig. 9, *more than three times the expected number* should modify *28*. However, in the LTAG-spinal Treebank, *number*, the head of the NP, attaches to the predicate *died* instead.

[7] Extraposition can be handled by multi-component LTAG (MC-LTAG) (Kroch and Joshi 1985; Frank 2002). Our LTAG-spinal Treebank at present does not support MC-LTAG.

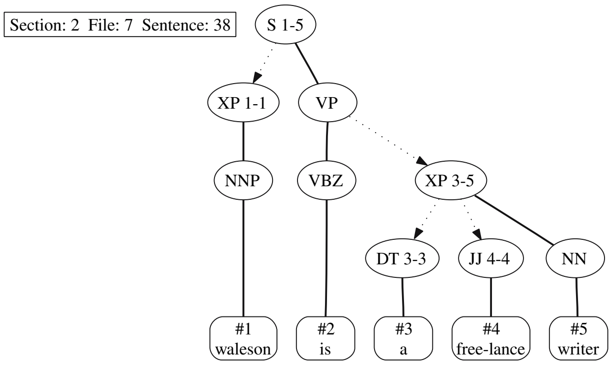

Fig. 8 Waleson is a free-lance writer ...

5 Properties of the LTAG-spinal Treebank

In this section, we describe the LTAG-spinal Treebank in numbers, and argue that LTAG-spinal as an annotation format represents an improvement on the PTB since it facilitates the recovery of semantic dependencies.

We ran the extraction algorithm on 49,208 sentences in the PTB. However, 454 sentences, or less than 1% of the total, were skipped. About 314 of these 454 sentences have gapping structures. Since the PTB does not annotate the trace of deleted predicates, additional manual annotation would be required to handle these sentences. For the rest of the 140 sentences, abnormal structures are generated due to tagging errors.

5.1 Statistics

In the LTAG-spinal Treebank extracted from the remaining 48,754 sentences in the PTB, there are 1,159,198 tokens, of which 2,365 are auxiliary trees and 8,467 are coordination structures. Five percent of all sentences contain at least one adjunction and 17% at least one coordination.

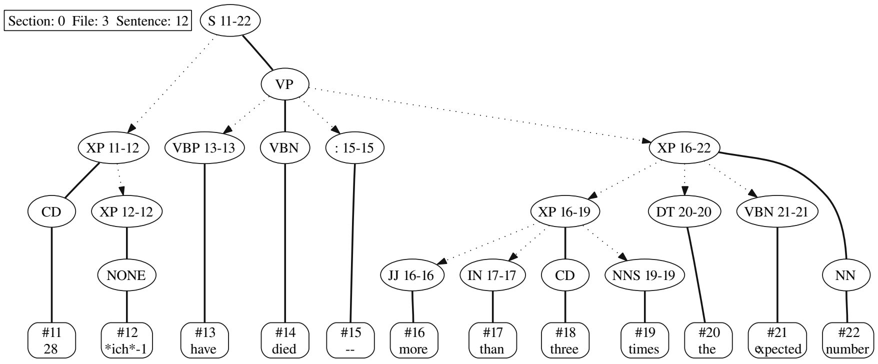

Fig. 9 ... 28 have died—more than three times the expected number ...

Table 1 Distribution of pred–arg pairs with respect to the distance between predicate and argument

Distance	Number	Percent
1	261,554	88.4
2	12,287	4.2
3	10,789	3.6
≥4	3,426	1.2
Ill-formed	1,661	0.6
Complex arg	6,135	2.1
Total	295,852	100.0

In the grammar extracted from 48,754 sentences in the PTB using steps 1–4 of the algorithm described in Sect. 3.2.3, there are 1,224 different types of spinal e-trees, and 507 of them appear only once in the LTAG-spinal Treebank. This result is compatible with the long tail of the distribution observed in Xia (2001) and Chen et al. (2006). Many of these e-trees are just noise. On the other hand, after executing step 5 (normalization), there remain only 135 different *normalized* spinal e-trees, and only seven of them appear only once in the treebank. We also avoid the sparse data problem by using normalized e-trees.

5.2 Compatibility with Propbank

This section shows that our treebank maintains a high level of compatibility with Propbank and that its derivation trees, for the most part, permit easy recovery of Propbank predicate–argument relationships.

Propbank arguments are represented as word spans, not subtrees. So the first question is whether they correspond to subtrees in the LTAG-spinal derivation tree. We say that an argument is *well-formed* in the LTAG-spinal Treebank if it can be generated by a subtree some of whose direct children trees may be cut away. For example, *and the stocks* is generated by a sub-derivation tree anchored on *stocks*, while *and* and *the* attach to the tree for *stocks*. Then we say that the argument *the stocks* is well-formed because we can get it by cutting the *and* tree, a direct child of the *stocks* tree.

As shown in Table 1, we have 295,852 pairs[8] of predicate–argument structures. Only 1,661 arguments, 0.6% of all of the arguments, are not well-formed. Most of these cases are extraposition structures.

For the remaining 294,191 arguments, we now ask how easy it is to recover the argument from a given subtree containing it. By using a few heuristic rules, for example, removing the subtrees for the punctuation marks at the beginning and at the end, we can easily recover 288,056, or 97.4% of all the arguments. For the remaining 6,135 arguments, more contextual information is required to recover the argument. For example, we have a phrase *NP PP SBAR* (*a book in the library that*

[8] For the sake of convenience, particles are represented as arguments.

has never been checked out), where both *PP* and *SBAR* attach to the *NP* as modifiers. Here *NP*, instead of *NP PP*, is an argument of the main verb of *SBAR* in the Propbank. In order to handle cases like these, learning methods should be used. However, we have a baseline of 97.4% for this task, which is obtained by just ignoring these difficult cases.

The next question is how to find the subtree of an argument if we are given a predicate. We evaluate the LTAG-spinal Treebank by studying the pattern of the path from the predicate to the argument for all the predicate–argument pairs in the treebank. Table 1 shows the distribution of the distances between the predicate and the argument in derivation trees. Distance = 1 means the predicate and the argument are directly connected.

The following is a list of the most frequent patterns of the path from the predicate to the argument. *P* represents a predicate, *A* represents an argument, *V* represents a modifying verb, and *Coord* represents predicate coordination. Arrows point from the parent to the child. The number of arrows is the distance between the predicate and argument, except for the case of a conjunct and its parent, which are considered directly connected although there is an artificial Coord node in between. Conjuncts are regarded as two steps apart from each other. We use A_x, P_x and P_y to represent other arguments or predicates appearing in the sentence.

1. **P → A**
   ex: (What)$_{arg1}$ (will)$_{argM}$ *happen* (to dividend growth)$_{arg2}$?
2. **P ← A** (relative clause, predicate adjunction)
   ex: (the amendment)$_{arg0}$ which *passed* today
   ex: (the price)$_{arg1.1}$ *appears* (to go up)$_{arg1.2}$
3. **P ← Px → A** (subject and object controls, Fig. 10a)
   ex: (It)$_{arg0}$ plans to *seek* approval. (**Px** = plans)
4. **P ← Coord → Px → A** (shared arguments)
   ex: (Chrysotile fibers)$_{arg1}$ are curly and are more easily *rejected* by the body. (**Px** = are *on the left*.)
5. **V ← A**
   ex: the Dutch *publishing* (group)$_{arg0}$
6. **P ← Ax ← Py → A** (Fig. 10b)
   ex: (Mike)$_{arg0}$ has a letter to *send*. (**Ax** = letter, **Py** = has)
7. **P ← Coord ← Px → A** (control + coordination)
   ex: (It)$_{arg0}$ expects to *obtain* regulatory approval and *complete* the transaction. (**Px** = expects)
8. **P ← Px ← Py → A** (chained controls, Fig. 10c)
   ex: (Officials)$_{arg0}$ began visiting about 26,000 cigarette stalls to *remove* illegal posters. (**Px** = visiting, **Py** = began)

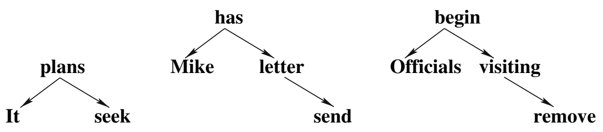

Fig. 10 Patterns: (a) **P** ← **Px** → **A** (b) **P** ← **Ax** ← **Py** → **A** (c) **P** ← **Px** ← **Py** → **A**

These eight patterns account for 95.5% of the total 295,852 pred–arg pairs in the treebank. Table 2 shows the frequency of these patterns. Patterns 1, 2 and 5 account for all the directly connected pred–arg pairs in Table 1.

We take this result to provide empirical justification for LTAG's notion of EDL. In addition, this result shows that the LTAG-spinal derivation tree provides support for automatically identifying predicate–argument relationships in a way that PTB annotation by itself does not.

5.3 Unlabeled argument identification

For the purpose of showing the compatibility of the LTAG-spinal Treebank with the Propbank, here we present a preliminary experiment on unlabeled argument identification, a task which is used to generate all the argument candidates for an argument classification system. We compare the performance of a rule-based approach for extracting unlabeled Propbank arguments from the LTAG-spinal Treebank with a SVM-based approach (Pradhan et al. 2005) for extracting the same information from the PTB. The point of this section is to evaluate how easily Propbank information can be recovered from LTAG-spinal annotation. The comparison with Pradhan et al. (2005) (see Table 3) is given for informational purposes only since we used Propbank information in the process of creating the LTAG-spinal Treebank (including the LTAG-spinal test data).

In Chen and Rambow (2003), pattern 1 is used to recognize arguments. However, it is not enough, since it only accounts for 82.4% of the total data. We have implemented a simple rule-based system for unlabeled argument identification by employing patterns 1–5 as follows. For each verbal predicate, we first collect all the sub-derivation trees in the local context based on path patterns 1, 2 and 5 in the previous section. If there is no argument candidate in subject position, we look for the subject by collecting sub-derivation trees according to patterns 3 and 4. Then we transform these sub-derivation trees into phrases with a few simple rules as described in the previous section. We achieved an *F*-score of 91.3% for unlabeled non-trace argument identification on Section 23 of this treebank,[9] and 91.6% on the whole treebank.

This illustrates that the LTAG-spinal Treebank makes explicit semantic relations that are implicit or absent from the original PTB. Training a parser on the LTAG-spinal Treebank appears to be a very interesting alternative approach

[9] Section 23 of our treebank contains 2401 of the 2416 sentences in PTB Section 23.

Table 2 Distribution of pred–arg pairs with respect to the path from the predicate to the argument

	Path pattern	Distance	Number	Percent
1	P → A	1	243,796	82.4
2	P ← A	1	14,658	5.0
3	P ← Px → A	2	10,990	3.7
4	P ← Coord → Px → A	3	5,613	1.9
5	V ← A	1	3,100	1.0
6	P ← Ax ← Py → A	3	3,028	1.0
7	P ← Coord ← Px → A	2	839	0.3
8	P ← Px ← Py → A	3	704	0.2
Other patterns		2	458	0.2
Other patterns		3	1,444	0.5
Other patterns		≥4	3,426	1.2
Ill-formed			1,661	0.6
Complex arg			6,135	2.1
Total			295,852	100.0

Table 3 Unlabeled non-trace argument identification on Section 23

Model	Training data	Recall (%)	Precision (%)	*F*-score (%)
Rules on LTAG	0	90.8	91.7	91.3
SVMs on PTB	1M	96.2	95.8	96.0

toward semantic role labeling, one in which syntax and semantics are tightly connected.

6 Conclusions and future work

In this article, we have introduced LTAG-spinal, a novel variant of traditional LTAG with desirable linguistic, computational and statistical properties. Unlike in traditional LTAG, subcategorization frames and the argument–adjunct distinction are left underspecified in LTAG-spinal. LTAG-spinal with adjunction constraints is weakly equivalent to traditional LTAG.

The LTAG-spinal formalism is used to extract an LTAG-spinal Treebank from the Penn Treebank with Propbank annotation. Based on Propbank annotation, predicate coordination, and LTAG adjunction are successfully extracted. The LTAG-spinal Treebank makes explicit semantic relations that are implicit or absent from the original PTB. It provides a very desirable resource for statistical LTAG parsing, incremental parsing, dependency parsing, and shallow semantic parsing.

In Shen and Joshi (2005), the LTAG-spinal Treebank was used to train and evaluate an incremental parser for LTAG-spinal. In Shen and Joshi (2007), an efficient LTAG dependency parser was trained and evaluated on this treebank, and it achieved an F-score of 90.5% on dependencies on Section 23 of this treebank. In the future, we will extend our work to semantic parsing based on this treebank.

The corpus is freely available for research purposes. The homepage of this resource is http://www.cis.upenn.edu/~xtag/spinal. The two parsers described above are also available for download at that page. We plan to release this resource through LDC in the future, at which time we will be able to include the mapping to the Propbank annotation.

We have created a comprehensive Java API that provides full access to the LTAG-spinal Treebank, the output of our parsers, and the special format of the Propbank annotation used in this work. It can be used for tasks such as postprocessing the parser output and producing graphical representations as in the illustrations. The API will be available under the link given above.

We hope this resource will promote research in statistical LTAG parsing, as the Penn Treebank did for CFG parsing. In the future, we also plan to build a standard LTAG treebank based on this LTAG-spinal Treebank.

Acknowledgments We would like to thank our anonymous reviewers for valuable comments. We are grateful to Ryan Gabbard, who has contributed to the code for the LTAG-spinal API. We also thank Julia Hockenmaier, Mark Johnson, Yudong Liu, Mitch Marcus, Sameer Pradhan, Anoop Sarkar, and the CLRG and XTAG groups at Penn for helpful discussions.

References

Abeillé, A., & Rambow, O. (Eds.) (2001). *Tree Adjoining Grammars: Formalisms, linguistic analysis and processing*. Center for the Study of Language and Information.

Babko-Malaya, O., Bies, A., Taylor, A., Yi, S., Palmer, M., Marcus, M., Kulick, S., & Shen, L. (2006). Issues in synchronizing the English Treebank and PropBank. In *Frontiers in Linguistically Annotated Corpora (ACL Workshop)*.

Charniak, E. (1997). Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*.

Charniak, E., & Johnson, M. (2005). Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Chen, J., Bangalore, S., & Vijay-Shanker, K. (2006). Automated extraction of Tree Adjoining Grammars from treebanks. *Natural Language Engineering, 12*(3), 251–299.

Chen, J., & Rambow, O. (2003). Use of deep linguistic features for the recognition and labeling of semantic arguments. In *Proceedings of the 2003 Conference of Empirical Methods in Natural Language Processing*.

Chiang, D. (2000). Statistical parsing with an automatically-extracted Tree Adjoining Grammar. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Collins, M. (1999). Head-driven statistical models for natural language parsing. PhD thesis, University of Pennsylvania.

Frank, R. (2002). *Phrase structure composition and syntactic dependencies*. The MIT Press.

Hockenmaier, J., & Steedman, M. (2002). Generative models for statistical parsing with combinatory categorial grammar. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Joshi, A. K., Levy, L. S., & Takahashi, M. (1975). Tree adjunct grammars. *Journal of Computer and System Sciences, 10*(1), 136–163.

Joshi, A. K., & Schabes, Y. (1997). Tree-Adjoining Grammars. In G. Rozenberg & A. Salomaa (Eds.), *Handbook of formal languages* (Vol. 3, pp. 69–124). Springer-Verlag.

Joshi, A. K., & Srinivas, B. (1994). Disambiguation of super parts of speech (or Supertags): Almost parsing. In *Proceedings of COLING '94: The 15th Int. Conf. on Computational Linguistics*.

Kroch, A., & Joshi, A. K. (1985). The linguistic relevance of Tree Adjoining Grammar. Report MS-CIS-85-16. CIS Department, University of Pennsylvania.

Magerman, D. (1995). Statistical decision-tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*.

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

Palmer, M., Gildea, D., & Kingsbury, P. (2005). The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics, 31*(1), 71–106.

Pradhan, S., Hacioglu, K., Krugler, V., Ward, W., Martin, J., & Jurafsky, D. (2005). Support vector learning for semantic argument classification. *Machine Learning, 60*(1–3), 11–39.

Rambow, O., Weir, D., & Vijay-Shanker, K. (2001). D-Tree substitution grammars. *Computational Linguistics, 27*(1), 89–121.

Sarkar, A., & Joshi, A. K. (1996). Coordination in Tree Adjoining Grammars: Formalization and implementation. In *Proceedings of COLING '96: The 16th Int. Conf. on Computational Linguistics*.

Schabes, Y., & Waters, R. C. (1995). A cubic-time, parsable formalism that lexicalizes context-free grammar without changing the trees produced. *Computational Linguistics, 21*(4), 479–513.

Shen, L. (2006). Statistical LTAG parsing. PhD Thesis, University of Pennsylvania.

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

Shen, L., & Joshi, A. K. (2007). Bidirectional LTAG dependency parsing. Technical Report 07-02, IRCS, University of Pennsylvania.

Steedman, M. (2000). *The syntactic process*. The MIT Press.

Sturt, P., & Lombardo, V. (2005). Processing coordinated structures: Incrementality and connectedness. *Cognitive Science, 29*(2), 291–305.

Vadas, D., & Curran, J. (2007). Adding noun phrase structure to the Penn Treebank. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Xia, F. (2001). Automatic grammar generation from two different perspectives. PhD thesis, University of Pennsylvania.

XTAG-Group (2001). A lexicalized tree adjoining grammar for English. Technical Report 01-03, IRCS, University of Pennsylvania.

Yi, S. (2007). Robust semantic role labeling using parsing variations and semantic classes. PhD thesis, University of Pennsylvania.