Explanation-based Learning and Finite State Transducers: Applications to Parsing Lexicalized Tree Adjoining Grammars

B. Srinivas
Department of Computer and Information Sciences
University of Pennsylvania
Philadelphia, PA 19104
srini@linc.cis.upenn.edu

Abstract. Explanation-based Learning techniques have been applied in NLP for speeding up parsing in limited domains. In [7], we showed that combining the LTAG representation with the EBL technique provides a novel method for parsing based on a FST mechanism and demonstrated a speedup in parsing times on the ATIS corpus. In this paper, we extend that approach to account for cases of "long distance extractions" and show that the same FST mechanism can be used for these sentences as well.

1 Introduction

Explanation-based Learning (EBL) techniques were originally introduced in the AI literature by [2, 3, 8]. The main idea behind EBL is that it is possible to form generalizations from single positive training examples if the system can explain why the example is an instance of the concept under study. The generalizer possesses the knowledge of the concept under study and the rules of the domain for constructing the required explanation. The objective of EBL is to keep track of suitably generalized solutions (explanations) to problems solved in the past and to replay those solutions to solve new but somewhat similar problems in the future.

EBL Terminology	→	Parsing Terminology
Concept	→	Grammaticality
Rules of the Domain	→	Grammar
Instance	→	Sentence
Explanation	→	Parse
Explanation Structure	→	Derivation Structure
the grammar		

Table 1. Correspondence between EBL and parsing terminology.

The correspondence between the EBL terminology and parsing terminology is shown in Table 1. In parsing, the concept under study is *grammaticality* of an input. The fact and rules to be used in explaining the grammaticality of input is given by the *grammar*. A training *sentence* serves as an instance of the concept. A *parse* of a sentence represents an *explanation* of why the sentence is grammatical according to the grammar. A *derivation tree* represents the explanation structure. Parsing new sentences amounts to finding analogous explanations from the explanations for the training sentences.

Rayner [5] was the first to investigate the usefulness of the EBL technique in the context of natural language parsing systems. Samuelsson and Rayner [6] specialize a CFG-based grammar for the ATIS domain by storing chunks of the parse trees present in a treebank of parsed examples. Neumann [4] also attempts to specialize a grammar given a training corpus of parsed examples by generalizing the parse for each sentence and storing the generalized phrasal derivations under a suitable index.

We showed in [7] that combining the LTAG representation with the EBL technique provides a novel method for parsing in limited domains based on a FST mechanism. In this paper, we extend that approach to account for cases of "long distance extractions" and show that the same mechanisms of FST can be used for these sentences as well.

The paper is laid out as follows. In Section 2 we present an overview of our approach to using EBL. The two types of generalization mechanisms that fall out of the LTAG representation are presented in Section 3 and Section 4. In Section 5 we present the FST representation of the set of generalized parses.

2 Our Approach

We are pursuing the EBL approach in the context of a wide-coverage Lexicalized Tree-Adjoining grammar development system called XTAG [1]. Details of the LTAG formalism and the XTAG system are presented in [9].

The training phase of the EBL process involves generalizing the derivation trees generated by XTAG for the training sentences and storing these generalized parses in the generalized parse database under an index computed from the lexical features of the words of the sentence. The application phase of EBL is shown in the flowchart in Figure 1. An index using the lexical features of the words in the input sentence is computed. Using this index, a set of generalized parses is retrieved from the generalized parse database created in the training phase. If the retrieval fails to yield any generalized

ECAI 96. *12th European Conference on Artificial Intelligence*
Edited by W. Wahlster
Published in 1996 by John Wiley & Sons, Ltd.

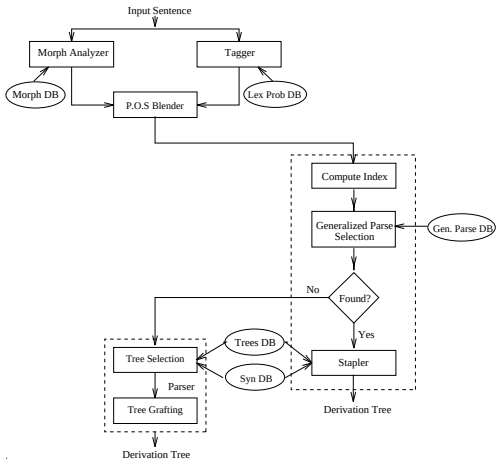

Figure 1. Flowchart of the XTAG system with the EBL component

parse then the input sentence is parsed using the full parser. However, if the retrieval succeeds, the output is a generalized parse that associates with each word the elementary tree that it anchors and the elementary tree into which it adjoins or substitutes into – an *almost parse*. The *almost parse* is input to the "stapler" which instantiates the generalized parse to the sentence. The details of the "stapler" are discussed in Section 7.

The three key aspects of LTAG (a) lexicalization, (b) extended domain of locality and (c) factoring of recursion from the domain of dependencies (1) lead to an *immediate* generalization of parses for the training set of sentences, (2) achieve generalization over recursive substructures of the parses, and (3) allow for a finite state transducer (FST) representation of the set of generalized parses.

3 Feature-generalization

In LTAGs, a derivation tree uniquely identifies a parse for a sentence. A derivation tree is a sequence of elementary trees associated with the lexical items of the sentence along with substitution and adjunction links among the elementary trees. Also, the values for features at each node of every elementary tree is instantiated during the parsing process. Given an LTAG parse, the generalization of the parse is truly *immediate* in that a generalized parse is obtained by (a) uninstantiating the particular lexical items that anchor the individual elementary trees in the parse and (b) uninstantiating the feature values contributed by the morphology of the anchor and the derivation process. This type of generalization is called *feature-generalization* and results in a feature-generalized parse. The process of feature generalization is shown in Figure 2 for the sentence *who did you say flies from Boston to Washington*. Figure 2(a) illustrates the derivation structure for the sentence. Each word in the derivation structure is associated with elementary trees with initial trees (αs) or auxiliary trees (βs). The nodes in the derivation tree are connected by substitution links (dotted lines) or adjunction links (). Applying the feature-generalization on Figure 2(a) results in Figure 2(b). The trees β_3 and β_4 are no longer distinct so we denote them by β_3. So also the trees α_4 and α_5 are no longer distinct, so we denote them by α_4 in Figure 2(b).

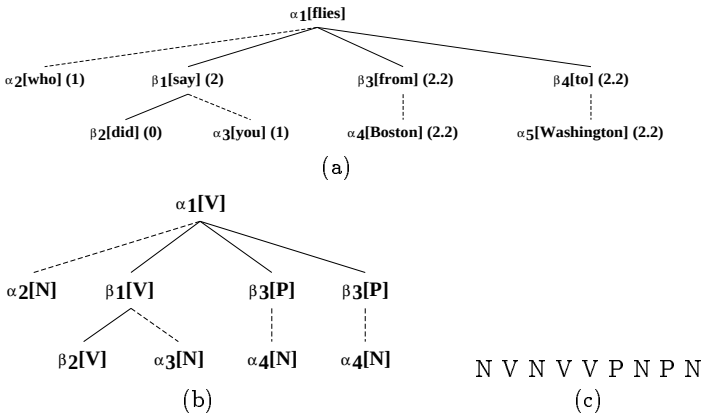

Figure 2. (a) Derivation structure (b) Feature Generalized Derivation structure (c) Index for the generalized parse for the sentence *who did you say flies from Boston to Washington*

3.1 Storing and retrieving feature-generalized parses

The feature-generalized parse of a sentence is stored in a generalized parse database under an index computed from the training sentence. In this case, the part-of-speech (POS) sequence of the training sentence is treated as the index to the feature-generalized parse. The index for the example sentence *who did you say flies from Boston to Washington* is shown in Figure 2(c). In the application phase, the POS sequence of the input sentence is used to retrieve a generalized parse(s) which is then instantiated to the features of the sentence.

We have chosen here to index the generalized parse under the POS sequence of the training sentence. However, it must be noted that depending on the degree of abstraction from the lexical items, the generalized parse may be stored under different indices containing varying amounts of lexical specific information. The degree of abstraction strongly influences the amount of training material required for adequate coverage of the test sentences and the number of parses assigned to the test sentences.

This method of retrieving a generalized parse allows for parsing of sentences of the same lengths and the same POS sequence as those in the training corpus. However, in our approach there is another generalization that falls out of the LTAG representation which allows for flexible matching of the index to allow the system to parse sentences that may differ in length from sentences similar to the ones in the training corpus.

4 Recursive-generalization

Auxiliary trees in LTAG represent recursive structures. So if there is an auxiliary tree that is used in an LTAG parse, then that tree with the trees for its arguments can be repeated any number of times, or possibly omitted altogether, to get parses of sentences that differ from the sentences of the training corpus only in the number of auxiliary trees. This type of generalization can be called *recursive-generalization*. Figure 3 illustrates the process of recursive generalization. The two Kleene star regular expressions around the preposition phrase in Figure 3(b) can be merged into one and the resulting recursive generalized parse is shown in Figure 3(c).

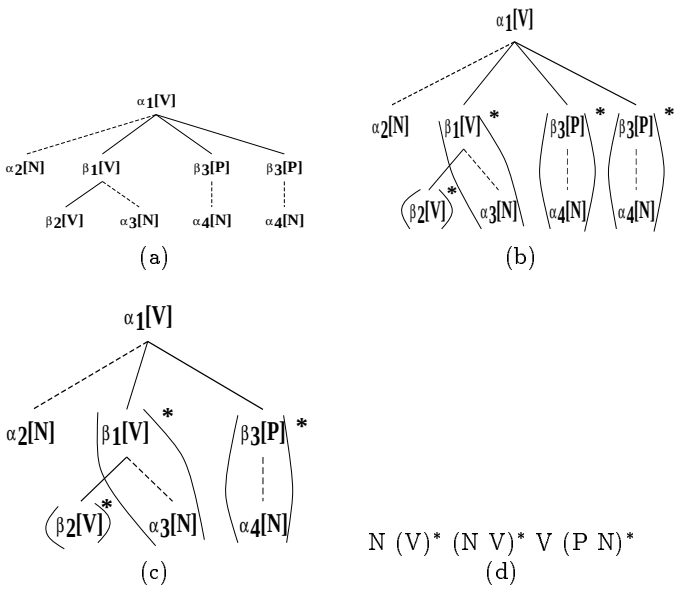

Figure 3. (a) Feature-generalized derivation tree (b) Recursive-generalized derivation tree (c) Recursive-generalized derivation tree with the two Kleene-stars collapsed into one (d) Index for the generalized parse for the sentence *who did you say flies from Boston to Washington.*

4.1 Storing and retrieving recursive-generalized parses

Due to recursive-generalization, the POS sequence covered by the auxiliary tree and its arguments can be repeated zero or more times. As a result, the index of a generalized parse of a sentence with auxiliary trees is no longer a string but a regular expression pattern on the POS sequence and retrieval of a generalized parse involves regular expression pattern matching on the indices. The index for the example sentence is shown in Figure 3(d), since the auxiliary verb, the verb with clausal complement, and the prepositions anchor auxiliary trees.

The most efficient method of performing regular expression pattern matching is to construct a *finite state machine* for each of the stored patterns and then traverse the machine using the given test pattern. If the machine reaches the final state, then the test pattern matches one of the stored patterns.

Given that the index of a test sentence matches one of the indices from the training phase, the generalized parse retrieved will be a parse of the test sentence, modulo the auxiliary trees. For example, if the test sentence, tagged appropriately, is

(1) who/N did/V you/N say/V flies/V from/P Boston/N to/P Washington/N on/P Monday/N.

the index of the test sentence matches the index of the training sentence, however, the generalized parse retrieved needs to be augmented to accommodate the additional modifier *on/P Monday/N*. To do so, we need to provide a mechanism that assigns the additional auxiliary trees and their arguments the following:

1. The elementary trees that they anchor and
2. The substitution and adjunction links to the trees they substitute or adjoin into.

In other words, each instantiation of the Kleene-star in the regular expression index has to be assigned the required structure so that it can be integrated into the derivation tree. A Finite-State Transducer (FST) representation allows a way of combining the generalized parse and the POS sequence (regular expression) that it is indexed by into a uniform representation.

5 Finite-State Transducer Representation

A finite-state transducer is a finite-state machine with each state transition arc labeled with two symbols – the input and the output symbols. For each successful state transition, the output symbol associated with the transition arc is output. For our purpose, the input symbol represents a POS of a word and the output symbol represents information on how the word integrates with the rest of the derivation.

A derivation structure can be encoded by specifying the elementary tree associated with each word along with the information to indicate which elementary tree it would be adjoined or substituted into. The encoding should be such that the additional auxiliary trees and their arguments that result from the recursive generalization are also assigned correct substitution and adjunction links.

5.1 Types of auxiliary trees

Auxiliary trees in LTAG have been distinguished as modifier auxiliary trees and predicative auxiliary trees. While the modifier auxiliary trees modify the constituent they adjoin on to, the predicative auxiliary trees subcategorize for the constituent they adjoin on to. Examples of the modifier auxiliary trees and predicative auxiliary trees are shown in Figure 4.

The additional auxiliary trees that result from recursive generalization display different adjunction behaviour depending on the type of the auxiliary tree as shown in sentences (2) and (3). The successive repetitions of the modifier in (2) can modify the same head (*flies*) as did the modifiers in the training example. However, successive repetition of the predicative auxiliary trees take as complement the clause following it. Thus, in (3), *think* adjoins on to *said* and not to *flies*.

Item		Description
this_tree	:	the elementary tree that the word anchors
head_word	:	the word on which the current word is dependent on; "–" if the current word does not depend on any other word.
head_tree	:	the tree anchored by the head word;
	:	"T_{clause}" if the type of the tree anchored by the head word is a clausal tree
	:	"–" if the current word does not depend on any other word;
	:	"$*$" if the tree does not matter.
number	:	a signed number that indicates the direction and the ordinal position of the particular head elementary tree from the position of the current word *OR*
	:	an unsigned number that indicates the gorn-address (i.e., the node address) in the derivation tree to which the word attaches *OR*
	:	"–" if the current word does not depend on any other word.

Table 2. Description of the components in the tuple representation associated with each word.

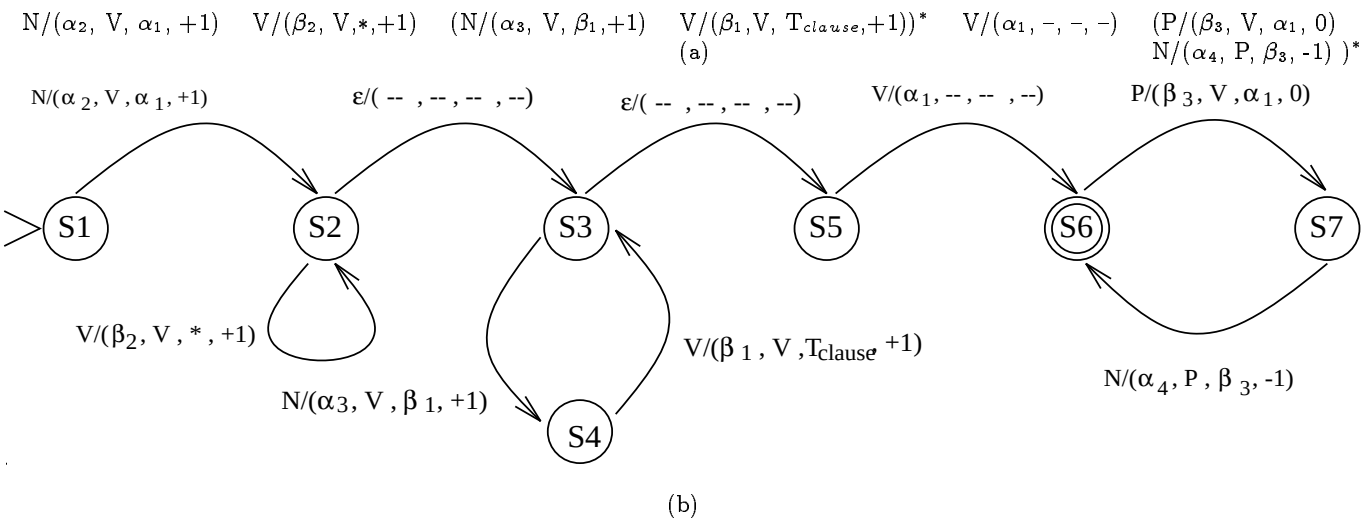

Figure 5. Finite State Transducer Representation for the sentences: *who did you say flies from Boston to Washington, who did you say flies from Boston to Washington on Monday, who did you think I said flies from Boston to Washington, ...*

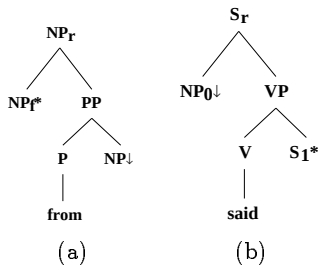

Figure 4. (a): Modifier Auxiliary Tree, (b): Predicative Auxiliary Tree

(2) who did you say flies from Boston to Washington on Monday.

(3) who did you think I said flies from Boston to Washington.

Based on the preceding observations we assume that the additional auxiliary trees along with their arguments would be assigned elementary trees along with substitution and adjunction links as follows:

- The additional auxiliary trees along with their arguments would be assigned the same generalized elementary trees as they were assigned in the training example.
- The arguments of the auxiliary trees will be assigned the same substitution and adjunction links as they were assigned the training example.
- If the auxiliary tree is a modifier auxiliary tree then it is assumed that it modifies the same head as it did in the training example.
- If the auxiliary tree is a predicative auxiliary tree then it is assumed to adjoin to the immediately following clause, in terms of string position.

5.2 Encoding the generalized derivation tree

We encode the derivation tree in a manner similar to the *dependency representation* by associating each word with a tuple (this_tree, head_word, head_tree, number) where the description for the components of the tuple is given in Table 2. A derivation tree expresses two types of relations between words: head-complement relations and head-modifier relations. For a head-complement dependency relation, the number in the tuple associated with a word is a signed number that indicates the ordinal position of its head in the input string. For a head-modifier dependency relation, the number in the tuple associated with a word is an unsigned number that represents the tree-address (Gorn address) of its head in the derivation tree.

Consider the example sentence (4) with the generalized derivation tree in Figure 3(c).

(4) Who did you say flies from Boston to Washington

Following this notation, the derivation tree in Figure 3(c) is represented as in Figure 5(a) which can be seen as a path in an FST as in Figure 5(b). The FSTs resulting from the generalized derivation for each sentence in a corpus are combined by a union and the resulting FST is minimized.

This FST representation is possible due to the lexicalized nature and the extended domain of locality of elementary trees because of which lexical dependencies are localized to with in a single elementary structure. Further, the factoring of recursion in LTAGs provides a method of generalizing over recursive structures independent of whether the structure expresses head-complement relation or head-modifier relation. Also, it is interesting to note that the head-complement relations are expressed on string positions, where as head-modifier relations are expressed on the structural positions.

6 Phrasal EBL

The method described in this paper is ideally suited for domains where the patterns are repetitive at the sentence level, However, this method can be extended for domains where the patterns are repetitive at the phrasal level. The idea would be to identify the repetitive phrasal subtrees from the derivation trees and create an FST as described in this paper for each of those subtrees, such as an FST for NPs, FST for PPs and so on. In the test phase these FSTs are applied as a sequence of transductions with each transduction resulting in the type of the phrase that is recognized along with the phrase internal substitution and adjunction links. The head of the phrase (word that does not depend on any other word in the phrase) identified by the current transduction is used in the next transduction for linking with other words.

7 Stapler

A stapler is an impoverished parser that is used in conjunction with the FST to generate all possible attachment sites for all (if any) modifiers besides instantiating the features of nodes of the trees by term unification. The stapler performs the following tasks.

1. Modifier Attachment: It computes the alternate sites of attachments for modifiers, if any, since the generalized derivation tree provides only one possible attachment site.
2. Address of Operation: The links in the derivation tree are labeled with node addresses to indicate the location of the substitution and adjunction operation.
3. Feature Instantiation: The values of the features on the nodes of the elementary trees are instantiated by a process of unification.

8 Experimental Results

We have tested the performance of our approach on the ATIS corpus. A total of 400 sentences, with an average length of 10 words per sentence, which had been completely parsed by the XTAG system was randomly divided into two sets, a training set of 300 sentences and a test set of 100 sentences.[1] For each of the training sentences, the correct parse was generalized and stored as a path in a FST. The FST was then minimized. The minimized FST was tested for retrieval of a generalized parse for each of the test sentences that were pretagged with the correct POS sequence. When a match is found, the output of the FST is a generalized parse that associates with each word the elementary tree that it anchors and the elementary tree into which it adjoins or substitutes into – an *almost parse*. The size of the resulting FST, the number of sentences which were assigned the correct parse, the number of parses that were assigned to each sentence and the average time spent per sentence are shown in Figure 3.

Size of train set	# of states	% recall	Avg. # of parses	Avg. response time
300	1162	80%	2	0.35 sec/sent

Table 3. Recall percentage, Average number of parses, Response times and Size of the FST for ATIS corpus

We obtained a speed-up of a factor of 60 in parsing times when using the FST in conjunction with the stapler in comparison to the XTAG parser on the test sentences.

REFERENCES

[1] Christy Doran, Dania Egedi, Beth Ann Hockey, B. Srinivas, and Martin Zaidel, 'XTAG System - A Wide Coverage Grammar for English', in *Proceedings of the* 17^{th} *International Conference on Computational Linguistics (COLING '94)*, Kyoto, Japan, (August 1994).

[2] Steve Minton, 'Quantitative Results concerning the utility of Explanation-Based Learning', in *Proceedings of* 7^{th} *AAAI Conference*, pp. 564–569, Saint Paul, Minnesota, (1988).

[3] Tom M. Mitchell, Richard M. Keller, and Smadar T. Kedar-Carbelli, 'Explanation-Based Generalization: A Unifying View', *Machine Learning 1*, **1**, 47–80, (1986).

[4] Günter Neumann, 'Application of Explanation-based Learning for Efficient Processing of Constraint-based Grammars', in 10^{th} *IEEE Conference on Artificial Intelligence for Applications*, San Antonio, Texas, (1994).

[1] We hope to test on more sentences by the time of the conference

[5] Manny Rayner, 'Applying Explanation-Based Generalization to Natural Language Processing', in *Proceedings of the International Conference on Fifth Generation Computer Systems*, Tokyo, (1988).

[6] Christer Samuelsson and Manny Rayner, 'Quantitative Evaluation of Explanation-Based Learning as an Optimization Tool for Large-Scale Natural Language System', in *Proceedings of the 12^{th} International Joint Conference on Artificial Intelligence*, Sydney,Australia, (1991).

[7] B. Srinivas and Aravind K. Joshi, 'Some Novel Applications of Explanation-based Learning to Parsing Lexicalized Tree-Adjoining Grammars', in *Proceedings of the 33^{rd} Conference of Association of Computational Linguistics*, (1995).

[8] Frank van Harmelen and Allan Bundy, 'Explanation-Based Generalization = Partial Evaluation', *Artificial Intelligence*, **36**, 401–412, (1988).

[9] The XTAG-Group, 'A Lexicalized Tree Adjoining Grammar for English', Technical Report IRCS 95-03, University of Pennsylvania, (1995).