Automatic Grammar Induction and Parsing Free Text: A Transformation-Based Approach

Eric Brill *

Department of Computer and Information Science
University of Pennsylvania
brill@unagi.cis.upenn.edu

ABSTRACT

In this paper we describe a new technique for parsing free text: a transformational grammar[1] is automatically learned that is capable of accurately parsing text into binary-branching syntactic trees with nonterminals unlabelled. The algorithm works by beginning in a very naive state of knowledge about phrase structure. By repeatedly comparing the results of bracketing in the current state to proper bracketing provided in the training corpus, the system learns a set of simple structural transformations that can be applied to reduce error. After describing the algorithm, we present results and compare these results to other recent results in automatic grammar induction.

1. INTRODUCTION

There has been a great deal of interest of late in the automatic induction of natural language grammar. Given the difficulty inherent in manually building a robust parser, along with the availability of large amounts of training material, automatic grammar induction seems like a path worth pursuing. A number of systems have been built which can be trained automatically to bracket text into syntactic constituents. In [10] mutual information statistics are extracted from a corpus of text and this information is then used to parse new text. [13] defines a function to score the quality of parse trees, and then uses simulated annealing to heuristically explore the entire space of possible parses for a given sentence. In [3], distributional analysis techniques are applied to a large corpus to learn a context-free grammar.

The most promising results to date have been based on the inside-outside algorithm (i-o algorithm), which can be used to train stochastic context-free grammars. The i-o algorithm is an extension of the finite-state based Hidden Markov Model (by [1]), which has been applied successfully in many areas, including speech recognition and part of speech tagging. A number of recent papers have explored the potential of using the i-o algorithm to automatically learn a grammar [9, 15, 12, 6, 7, 14].

Below, we describe a new technique for grammar induction.[2] The algorithm works by beginning in a very naive state of knowledge about phrase structure. By repeatedly comparing the results of parsing in the current state to the proper phrase structure for each sentence in the training corpus, the system learns a set of ordered transformations which can be applied to reduce parsing error. We believe this technique has advantages over other methods of phrase structure induction. Some of the advantages include: the system is very simple, it requires only a very small set of transformations, learning proceeds quickly and achieves a high degree of accuracy, and only a very small training corpus is necessary. In addition, since some tokens in a sentence are not even considered in parsing, the method could prove to be considerably more resistant to noise than a CFG-based approach. After describing the algorithm, we present results and compare these results to other recent results in automatic phrase structure induction.

2. THE ALGORITHM

The learning algorithm is trained on a small corpus of partially bracketed text which is also annotated with part of speech information. All of the experiments presented below were done using the Penn Treebank annotated corpus[11]. The learner begins in a naive initial state, knowing very little about the phrase structure of the target corpus. In particular, all that is initially known is that English tends to be right branching and that final punctuation is final punctuation. Transformations are then learned automatically which transform the output of the naive parser into output which better resembles the phrase structure found in the training corpus. Once a set of transformations has been learned, the system is capable of taking sentences tagged with parts of speech and returning a binary-branching structure with nonterminals unlabelled[3].

2.1. The Initial State Of The Parser

Initially, the parser operates by assigning a right-linear structure to all sentences. The only exception is that final punctuation is attached high. So, the sentence "*The dog and old cat ate* ." would be incorrectly bracketed as:

((The (dog (and (old (cat ate))))) .)

*The author would like to thank Mark Liberman, Meiting Lu, David Magerman, Mitch Marcus, Rich Pito, Giorgio Satta, Yves Schabes and Tom Veatch. This work was supported by DARPA and AFOSR jointly under grant No. AFOSR-90-0066, and by ARO grant No. DAAL 03-89-C0031 PRI.

[1] Not in the traditional sense of the term.

[2] A similar method has been applied effectively in part of speech tagging; see [5, 4].

[3] This is the same output given by systems described in [10, 3, 12, 14]

The parser in its initial state will obviously not bracket sentences with great accuracy. In some experiments below, we begin with an even more naive initial state of knowledge: sentences are parsed by assigning them a random binary-branching structure with final punctuation always attached high.

2.2. Structural Transformations

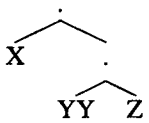
The next stage involves learning a set of transformations that can be applied to the output of the naive parser to make these sentences better conform to the proper structure specified in the training corpus. The list of possible transformation types is prespecified. Transformations involve making a simple change triggered by a simple environment. In the current implementation, there are twelve allowable transformation types:

- (1-8) $(Add|delete)$ a $(left|right)$ parenthesis to the $(left|right)$ of part of speech tag X.
- (9-12) $(Add|delete)$ a $(left|right)$ parenthesis between tags X and Y.

To carry out a transformation by adding or deleting a parenthesis, a number of additional simple changes must take place to preserve balanced parentheses and binary branching. To give an example, to delete a left paren in a particular environment, the following operations take place (assuming, of course, that there is a left paren to delete):

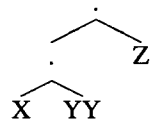
1. Delete the left paren.
2. Delete the right paren that matches the just deleted paren.
3. Add a left paren to the left of the constituent immediately to the left of the deleted left paren.
4. Add a right paren to the right of the constituent immediately to the right of the deleted paren.
5. If there is no constituent immediately to the right, or none immediately to the left, then the transformation fails to apply.

Structurally, the transformation can be seen as follows. If we wish to delete a left paren to the right of constituent X[4], where X appears in a subtree of the form:



[4]To the right of the rightmost terminal dominated by X if X is a nonterminal.

carrying out these operations will transform this subtree into[5]:



Given the sentence[6]:

The dog barked .

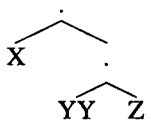
this would initially be bracketed by the naive parser as:

((The (dog barked)) .)

If the transformation *delete a left paren to the right of a determiner* is applied, the structure would be transformed to the correct bracketing:

(((The dog) barked) .)

To add a right parenthesis to the right of YY, YY must once again be in a subtree of the form:



If it is, the following steps are carried out to add the right paren:

1. Add the right paren.
2. Delete the left paren that now matches the newly added paren.
3. Find the right paren that used to match the just deleted paren and delete it.
4. Add a left paren to match the added right paren.

[5]The twelve transformations can be decomposed into two structural transformations, that shown here and its converse, along with six triggering environments.

[6]Input sentences are also labelled with parts of speech.

This results in the same structural change as deleting a left paren to the right of X in this particular structure.

Applying the transformation *add a right paren to the right of a noun* to the bracketing:

((The (dog barked)) .)

will once again result in the correct bracketing:

(((The dog) barked) .)

2.3. Learning Transformations

Learning proceeds as follows. Sentences in the training set are first parsed using the naive parser which assigns right linear structure to all sentences, attaching final punctuation high. Next, for each possible instantiation of the twelve transformation templates, that particular transformation is applied to the naively parsed sentences. The resulting structures are then scored using some measure of success which compares these parses to the correct structural descriptions for the sentences provided in the training corpus. The transformation which results in the best scoring structures then becomes the first transformation of the ordered set of transformations that are to be learned. That transformation is applied to the right-linear structures, and then learning proceeds on the corpus of improved sentence bracketings. The following procedure is carried out repeatedly on the training corpus until no more transformations can be found whose application reduces the error in parsing the training corpus:

1. The best transformation is found for the structures output by the parser in its current state.[7]
2. The transformation is applied to the output resulting from bracketing the corpus using the parser in its current state.
3. This transformation is added to the end of the ordered list of transformations.
4. Go to 1.

After a set of transformations has been learned, it can be used to effectively parse fresh text. To parse fresh text, the text is first naively parsed and then every transformation is applied, in order, to the naively parsed text.

One nice feature of this method is that different measures of bracketing success can be used: learning can proceed in such a way as to try to optimize any specified measure of success. The measure we have chosen for our experiments is the same measure described in [12], which is one of the measures that arose out of a parser evaluation workshop [2]. The measure is the percentage of constituents (strings of words between matching parentheses) from sentences output by our system which do not cross any constituents in the Penn Treebank structural description of the sentence. For example, if our system outputs:

(((The big) (dog ate)) .)

and the Penn Treebank bracketing for this sentence was:

(((The big dog) ate) .)

then the constituent *the big* would be judged correct whereas the constituent *dog ate* would not.

Below are the first seven transformations found from one run of training on the Wall Street Journal corpus, which was initially bracketed using the right-linear initial-state parser.

1. Delete a left paren to the left of a singular noun.
2. Delete a left paren to the left of a plural noun.
3. Delete a left paren between two proper nouns.
4. Delet a left paren to the right of a determiner.
5. Add a right paren to the left of a comma.
6. Add a right paren to the left of a period.
7. Delete a right paren to the left of a plural noun.

The first four transformations all extract noun phrases from the right linear initial structure. The sentence "The cat meowed ." would initially be bracketed as:[8]

((The (cat meowed)) .)

Applying the first transformation to this bracketing would result in:

(((The cat) meowed) .)

[7] The *state* of the parser is defined as naive initial-state knowledge plus all transformations that currently have been learned.

[8] These examples are not actual sentences in the corpus. We have chosen simple sentences for clarity.

Applying the fifth transformation to the bracketing:

((We (ran (, (and (they walked))))) .)

would result in

(((We ran) (, (and (they walked)))) .)

3. RESULTS

In the first experiment we ran, training and testing were done on the Texas Instruments Air Travel Information System (ATIS) corpus[8].[9] In table 1, we compare results we obtained to results cited in [12] using the inside-outside algorithm on the same corpus. Accuracy is measured in terms of the percentage of noncrossing constituents in the test corpus, as described above. Our system was tested by using the training set to learn a set of transformations, and then applying these transformations to the test set and scoring the resulting output. In this experiment, 64 transformations were learned (compared with 4096 context-free rules and probabilities used in the i-o experiment). It is significant that we obtained comparable performance using a training corpus only 21% as large as that used to train the inside-outside algorithm.

Method	# of Training Corpus Sentences	Accuracy
Inside-Outside	700	90.36%
Transformation-Learner	150	91.12%

Table 1: Comparing two learning methods on the ATIS corpus.

After applying all learned transformations to the test corpus, 60% of the sentences had no crossing constituents, 74% had fewer than two crossing constituents, and 85% had fewer than three. The mean sentence length of the test corpus was 11.3. In figure 1, we have graphed percentage correct as a function of the number of transformations that have been applied to the test corpus. As the transformation number increases, overtraining sometimes occurs. In the current implementation of the learner, a transformation is added to the list if it results in *any* positive net change in the training set. Toward the end of the learning procedure, transformations are found that only affect a very small percentage of training sentences. Since small counts are less reliable than large counts, we cannot reliably assume that these transformations will also improve performance in the test corpus. One way around this overtraining would be to set a threshold: specify a minimum level of improvement that must result for a transformation to be learned. Another possibility is to use additional training material to prune the set of learned transformations.

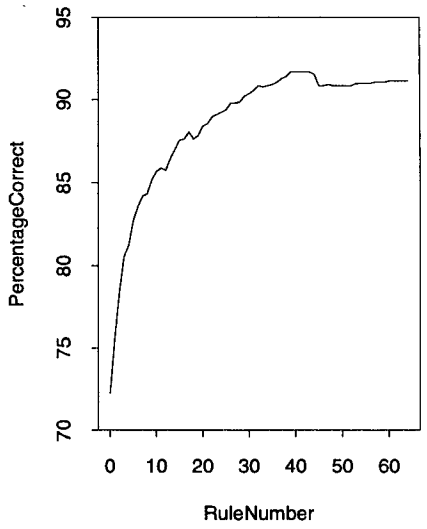


Figure 1: Results From the ATIS Corpus, Starting With Right-Linear Structure

We next ran an experiment to determine what performance could be achieved if we dropped the initial right-linear assumption. Using the same training and test sets as above, sentences were initially assigned a random binary-branching structure, with final punctuation always attached high. Since there was less regular structure in this case than in the right-linear case, many more transformations were found, 147 transformations in total. When these transformations were applied to the test set, a bracketing accuracy of 87.13% resulted.

The ATIS corpus is structurally fairly regular. To determine how well our algorithm performs on a more complex corpus, we ran experiments on the Wall Street Journal. Results from this experiment can be found in table 2.[10] Accuracy is again measured as the percentage of constituents in the test set which do not cross any Penn Treebank constituents.[11] As a point of comparison, in [14] an experiment was done using the i-o algorithm on a corpus of WSJ sentences of length 1-15. Training was carried out on 1,095 sentences, and an accuracy of 90.2% was obtained in bracketing a test set.

[9] In all experiments described in this paper, results are calculated on a test corpus which was not used in any way in either training the learning algorithm or in developing the system.

[10] For sentences of length 2-15, the initial right-linear parser achieves 69% accuracy. For sentences of length 2-20, 63% accuracy is achieved and for sentences of length 2-25, accuracy is 59%.

[11] In all of our experiments carried out on the Wall Street Journal, the test set was a randomly selected set of 500 sentences.

Sent. Length	# Training Corpus Sents	# of Transformations	% Accuracy
2-15	250	83	88.1
2-15	500	163	89.3
2-15	1000	221	91.6
2-20	250	145	86.2
2-25	250	160	83.8

Table 2: WSJ Sentences

In the corpus used for the experiments of sentence length 2-15, the mean sentence length was 10.80. In the corpus used for the experiment of sentence length 2-25, the mean length was 16.82. As would be expected, performance degrades somewhat as sentence length increases. In table 3, we show the percentage of sentences in the test corpus which have no crossing constituents, and the percentage that have only a very small number of crossing constituents[12].

Sent. Length	# Training Corpus Sents	% of 0-error sents	% of $\leq$1-error sents	% of $\leq$2-error sents
2-15	500	53.7	72.3	84.6
2-15	1000	62.4	77.2	87.8
2-25	250	29.2	44.9	59.9

Table 3: WSJ Sentences

In table 4, we show the standard deviation measured from three different randomly chosen training sets of each sample size and randomly chosen test sets of 500 sentences each, as well as the accuracy as a function of training corpus size.

Sent. Length	# Training Corpus Sents	% Correct	Std. Dev.
2-20	0	63.0	0.69
2-20	10	75.8	2.95
2-20	50	82.1	1.94
2-20	100	84.7	0.56
2-20	250	86.2	0.46
2-20	750	87.3	0.61

Table 4: More WSJ Results

We also ran an experiment on WSJ sentences of length 2-15 starting with random binary-branching structures with final punctuation attached high. In this experiment, 325 transformations were found using a 250-sentence training corpus, and the accuracy resulting from applying these transformations to a test set was 84.72%.

Finally, in figure 2 we show the sentence length distribution in the Wall Street Journal corpus.

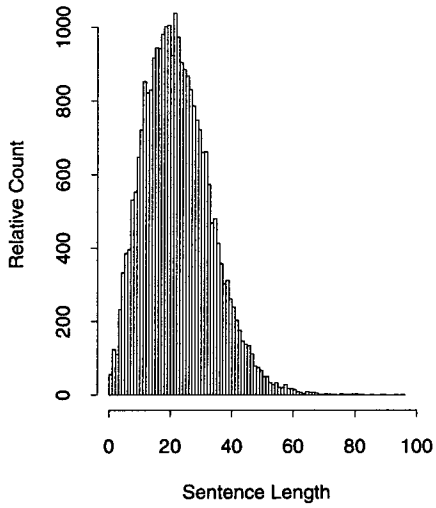


Figure 2: The Distribution of Sentence Lengths in the WSJ Corpus.

While the numbers presented above allow us to compare the transformation learner with systems trained and tested on comparable corpora, these results are all based upon the assumption that the test data is tagged fairly reliably (manually tagged text was used in all of these experiments, as well in the experiments of [12, 14].) When parsing free text, we cannot assume that the text will be tagged with the accuracy of a human annotator. Instead, an automatic tagger would have to be used to first tag the text before parsing. To address this issue, we ran one experiment where we randomly induced a 5% tagging error rate beyond the error rate of the human annotator. Errors were induced in such a way as to preserve the unigram part of speech tag probability distribution in the corpus. The experiment was run for sentences of length 2-15, with a training set of 1000 sentences and a test set of 500 sentences. The resulting bracketing accuracy was 90.1%, compared to 91.6% accuracy when using an unadulterated corpus. Accuracy only degraded by a small amount when using the corpus with adulterated part of speech tags, suggesting that high parsing accuracy rates could be achieved if tagging of the input was done automatically by a tagger.

[12]For sentences of length 2-15, the initial right linear parser parses 17% of sentences with no crossing errors, 35% with one or fewer errors and 50% with two or fewer. For sentences of length 2-25, 7% of sentences are parsed with no crossing errors, 16% with one or fewer, and 24% with two or fewer.

4. CONCLUSIONS

In this paper, we have described a new approach for learning a grammar to automatically parse free text. The method can be used to obtain good parsing accuracy with a very small training set. Instead of learning a traditional grammar, an ordered set of structural transformations is learned which can be applied to the output of a very naive parser to obtain binary-branching trees with unlabelled nonterminals. Experiments have shown that these parses conform with high accuracy to the structural descriptions specified in a manually annotated corpus. Unlike other recent attempts at automatic grammar induction which rely heavily on statistics both in training and in the resulting grammar, our learner is only very weakly statistical. For training, only integers are needed and the only mathematical operations carried out are integer addition and integer comparison. The resulting grammar is completely symbolic. Unlike learners based on the inside-outside algorithm which attempt to find a grammar to maximize the probability of the training corpus in hopes that this grammar will match the grammar that provides the most accurate structural descriptions, the transformation-based learner can readily use any desired success measure in learning.

We have already begun the next step in this project: automatically labelling the nonterminal nodes. The parser will first use the "transformational grammar" to output a parse tree without nonterminal labels, and then a separate algorithm will be applied to that tree to label the nonterminals. The nonterminal-node labelling algorithm makes use of ideas suggested in [3], where nonterminals are labelled as a function of the labels of their daughters. In addition, we plan to experiment with other types of transformations. Currently, each transformation in the learned list is only applied once in each appropriate environment. For a transformation to be applied more than once in one environment, it must appear in the transformation list more than once. One possible extension to the set of transformation types would be to allow for transformations of the form: add/delete a paren as many times as is possible in a particular environment. We also plan to experiment with other scoring functions and control strategies for finding transformations and to use this system as a postprocessor to other grammar induction systems, learning transformations to improve their performance. We hope these future paths will lead to a trainable and very accurate parser of free text.

References

1. Baker, J. (1979) Trainable grammars for speech recognition. In Jared J. Wolf and Dennis H. Klatt, eds. *Speech communication papers presented at the 97th Meeting of the Acoustical Society of America*, MIT.
2. Black, E., Abney, S., Flickenger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Marcus, M., Roukos, S., Santorini, B. and Strzalkowski, T. (1991) A Procedure for Quantitatively Comparing the Syntactic Coverage of English Grammars. Proceedings of the DARPA Workshop on Speech and Natural Language.
3. Brill, E. and Marcus, M. (1992) Automatically acquiring phrase structure using distributional analysis. Proceedings of the 5th DARPA Workshop on Speech and Natural Language. Harriman, N.Y.
4. Brill, E. and Marcus, M. (1992) Tagging an Unfamiliar Text With Minimal Human Supervision. American Association for Artificial Intelligence (AAAI) Fall Symposium on Probabilistic Approaches to Natural Language, Cambridge, Ma. AAAI Technical Report.
5. Brill, E. (1992) A Simple Rule-Based Part of Speech Tagger. Proceedings of the Third Conference on Applied Computational Linguistics (ACL). Trento, Italy.
6. Briscoe, T and Waegner, N. (1992) Robust Stochastic Parsing Using the Inside-Outside Algorithm. In Workshop notes from the AAAI Statistically-Based NLP Techniques Workshop.
7. Carroll, G. and Charniak, E. (1992) Learning Probabilistic Dependency Grammars from Labelled Text. In: Working Notes of the AAAI Fall Symposium on Probabilistic Approaches to Natural Language. Cambridge, Ma.
8. Hemphill, C., Godfrey, J. and Doddington, G. (1990). The ATIS spoken language systems pilot corpus. In 1990 DARPA Speech and Natural Language Workshop.
9. Lari, K. and Young, S. (1990) The estimation of stochastic context-free grammars using the inside-outside algorithm. Computer Speech and Language.
10. Magerman, D. and Marcus, M. (1990) Parsing a natural language using mutual information statistics, *Proceedings, Eighth National Conference on Artificial Intelligence (AAAI 90).*
11. Marcus, M., Santorini, B., and Marcinkiewicz, M. (1993) Building a large annotated corpus of English: the Penn Treebank. To appear in Computational Linguistics.
12. Pereira, F. and Schabes, Y. (1992) Inside-outside reestimation from partially bracketed corpora. Proceedings of the 20th Meeting of the Association for Computational Linguistics. Newark, De.
13. Sampson, G. (1986) A stochastic approach to parsing. In *Proceedings of COLING 1986*, Bonn.
14. Schabes, Y., Roth, M. and Osborne, R. (1993) Parsing the Wall Street Journal with the Inside-Outside algorithm. 1993 European ACL.
15. Sharman, R., Jelinek, F. and Mercer, R. (1990) Generating a grammar for statistical training. Proceedings of the 1990 Darpa Speech and Natural Language Workshop.