Transformation-Based Error-Driven Parsing

Eric Brill*

Spoken Language Systems Group
Laboratory for Computer Science
M.I.T.
email: brill@goldilocks.lcs.mit.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. 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 the number of errors. 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 that can be trained automatically to bracket text into syntactic constituents. In [MM90] mutual information statistics are extracted from a corpus of text and this information is then used to parse new text. [Sam86] 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 [BM92a], 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, which can be used to train stochastic context-free grammars. The inside-outside algorithm is an extension of the finite-state based Hidden Markov Model (by [Bak79]), 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 inside-outside algorithm to automatically learn a grammar [LY90, SJM90, PS92, BW92, CC92, SRO93].

Below, we describe a new technique for grammar induction. 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, a high degree of accuracy is achieved, and only a very small training corpus is necessary. The trained transformational parser is completely symbolic and can bracket text in linear time with respect to sentence

*This work was done while the author was at the University of Pennsylvania. 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.

length. In addition, since some tokens in a sentence are not even considered in parsing, the method could prove to be considerably more robust than a CFG-based approach when faced with noise or unfamiliar input. After describing the algorithm, we present results and compare these results to other recent results in automatic phrase structure induction.

2 TRANSFORMATION-BASED ERROR-DRIVEN LEARNING

The phrase structure learning algorithm is an application of a general learning technique called transformation-based error-driven learning. This learning paradigm, illustrated in figure 1, has proven to be successful in a number of different natural language applications. In [Bri93] (see also [Bri92, BM92b]), transformation-based learning is applied to part of speech tagging. It is shown that the transformation-based approach outperforms stochastic taggers ([MM91]) when trained on small corpora, and obtains performance comparable to stochastic taggers on larger corpora. This is significant in light of the fact that the transformation-based tagger is completely symbolic. In [BR93], this technique is applied to prepositional phrase attachment. The transformation-based approach is shown to significantly outperform the t-score technique for prepositional phrase attachment described in [HR91].

In its initial state, the transformation-based learner is capable of annotating text but is not very good at doing so. The initial state annotator is typically very easy to create. In part of speech tagging, the initial state annotator assigns every word its most likely tag in isolation, with unknown words being assigned a default tag. In prepositional phrase attachment, the initial state annotator always attaches prepositional phrases low. The naively annotated text is compared to the *true* annotation as indicated by a small manually annotated corpus, and transformations are learned that can be applied to the output of the initial state annotator to make it better resem-

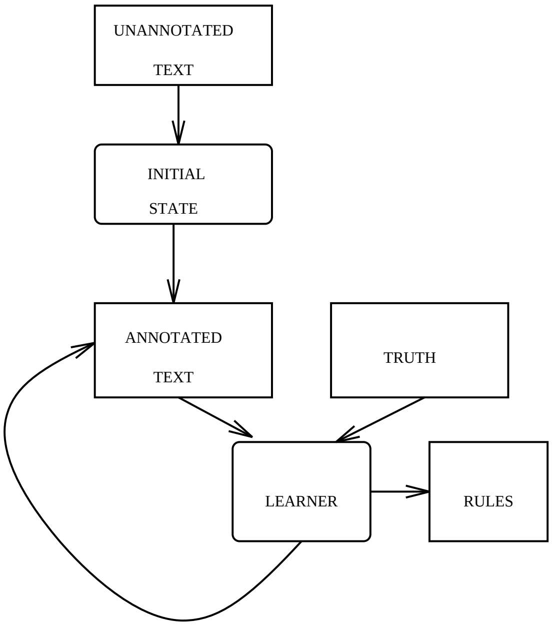


Figure 1: Transformation-Based Error-Driven Learning.

ble the *truth*. The learner learns a set of ordered transformations from a prespecified set of allowable transformations. A greedy search strategy is used to learn transformations: at each stage of learning, the best scoring transformation is learned for whatever scoring function is being used. Four elements must be defined to completely specify a transformation-based learner:

1. The initial-state annotator.
2. The list of allowable transformations.
3. The scoring function.
4. The search strategy.

3 LEARNING PHRASE STRUCTURE

The phrase structure 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[MSM93]. 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 (either manually tagged text, or the output of an automatic part of speech tagger) and returning a binary-branching structure with nonterminals unlabelled.[2]

3.1 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 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.

3.2 Structural Transformations

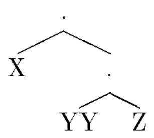
The next stage involves learning a set of transformations that can be applied to the output of the naive parser to make these sentences better conform to the proper structure specified in the training corpus. The list of possible transformation types is prespecified. Transformations involve making a 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 left 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[3], where X appears in a subtree of the form:

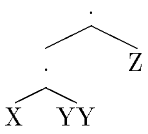


carrying out these operations will transform this subtree into:[4]

[2] This is the same output given by systems described in [MM90, Bri92, PS92, SRO93].

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

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



Given the sentence:[5]

The dog barked .

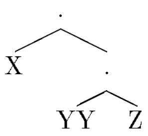
this would initially be bracketed by the naive parser as:

((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.

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) .)

3.3 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 that compares these parses to the correct structural descriptions for the sentences provided in the training corpus. The transformation resulting 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.[6]
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.

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

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

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 [PS92], which is one of the measures that arose out of a parser evaluation workshop [ea91]. 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.

Figure 2 shows the first ten transformations found from one run of training on the Wall Street Journal corpus, which was initially bracketed using the right-linear initial-state parser.

#	Add/ Delete	Left/ Right Paren	Environment
1	D	L	Left of NN
2	D	L	Left of NNS
3	A	R	Left of ,
4	D	L	Btwn NNP and NNP
5	D	L	Right of DT
6	A	R	Left of ,
7	D	R	Left of NNS
8	D	R	Btwn NN and NN
9	D	L	Btwn JJ and JJ
10	D	L	Right of $

Figure 2: The first 10 learned transformations.

The first two transformations, as well as transformation number 4, 5, 7, 8 and 9 all extract noun phrases from the right linear initial structure. After bracketing in the initial state, every word will be the leftmost terminal of a phrase containing the entire remainder of the sentence to its right. The first two transformations effectively remove singular and plural common nouns from such a structure and bracket them with the preceding constituent instead. The sentence "The cat meowed ." would initially be bracketed as:

((The/DT (cat/NN meowed/VBD)) ./.)

Applying the first transformation to this bracketing (or the second transformation to the same bracketing with *cats* replacing *cat*) would result in:

(((The cat) meowed) .)

If there is a left parenthesis between two proper nouns, then the second proper noun is initially bracketed with constituents that follow it rather than with the preceding proper noun. The fourth transformation fixes this. The sentence *General Motors is very profitable .* would initially be bracketed as:

((General/NNP (Motors/NNP (is (very profitable)))) .)

Applying the fourth transformation would convert this structure to:

(((General Motors) (is (very profitable))) .)

The following example demonstrates the interaction between transformations. The sentence *The fastest cars won .* would initially be bracketed as:

((The/DT (fastest/JJ (cars/NNS won/VBD))) .)

The first transformation to apply to this sentence would be number 2, resulting in:

((The ((fastest cars) won)) .)

The next applicable transformation is number 5, whose application results in:

(((The (fastest cars)) won) .)

After this transformation is applied, no other transformations can be applied to the sentence, and the correct structure is produced.

Transformation number 10 results from the fact that a number usually follows a dollar sign, and these two lexical items should be bracketed together. Transformations 3 and 6 result from the fact that a comma is a good indicator of the preceding phrase being terminated. Since each transformation is carried out only once per environment, multiple listings of a transformation are required if the transformation is to be applied multiple times to a single environment. The sentence *We called them , but they left .* would initially be bracketed as:

((We/PP (called/VBD (them/PP (,/, (but (they left)))))) .)

The first applicable transformation is number 3, whose application results in:

((We ((called them) (, (but (they left))))) .)

The next applicable transformation is number 6, whose application results in the correct structure:

(((We (called them)) (, (but (they left)))) .)

4 RESULTS

In the first experiment we ran, training and testing were done on the Texas Instruments Air Travel Information System (ATIS) corpus[HGD90].[7] In table 1, we compare results we obtained to results cited in [PS92] 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 4095 context-free rules and probabilities used in the inside-outside algorithm 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 Corp Sentences	Accuracy
Inside-Outside	700	90.4%
Transformation Learner	150	91.1%

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 3, 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.

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

[7] 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.

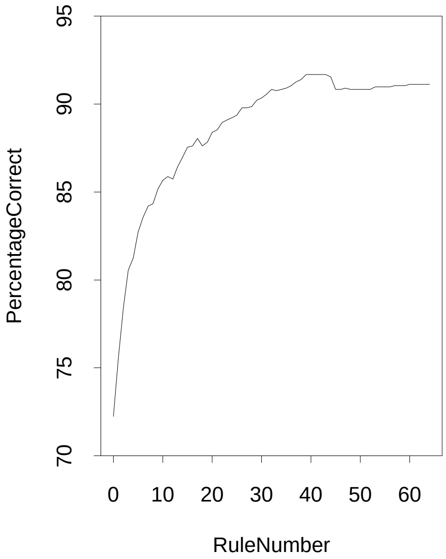


Figure 3: Accuracy as a function of transformation number for the ATIS Corpus.

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 87.1% 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.[8] Accuracy is again measured as the percentage of constituents in the test set which do not cross any Penn Treebank constituents.[9]

In the corpus we used for the experiments of sentence length 2-15, the mean sentence length was 10.80. In the corpus used for the experi-

Sent. Length	# Training Corpus Sents	# of Trans- formations	% 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

ment 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 that have no crossing constituents, and the percentage that have only a very small number of crossing constituents.[10]

Sent Length	# Training Corpus Sents	% of 0-error Sents	% of $\leq$2-error Sents
2-15	500	53.7	84.6
2-15	1000	62.4	87.8
2-25	250	29.2	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 for sentences of length 2 to 20.

In [SRO93], an experiment was run using the inside-outside algorithm to train a grammar from the partially bracketed Wall Street Journal corpus. As in the experiment with the ATIS corpus, all possible binary context-free rules were initially allowed, and random prob-

[8] 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%.

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

[10] 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.

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

Table 4: WSJ Sentences of Length 2 to 20.

abilities were assigned to each rule. A comparison of this approach to the transformation-based approach is shown in tables 5 and 6. The inside-outside experiment was carried out on sentences of length 1-15, and the transformation-based experiment was carried out on sentences of length 2-15. The inside-outside experiment had a grammar of 4095 probabilistic context free rules, which could be trimmed down to 450 rules without changing performance. 221 symbolic transformations were learned in the transformation-based experiment. In table 5, the transformation-based learner is shown to outperform the inside-outside algorithm when parsing accuracy is measured in terms of crossing brackets. In table 6, accuracy is measured as the percentage of sentences with no crossing bracket violations. We believe these results are significant, considering that the transformation-based approach is only a *weakly* statistical learner (only integer addition and comparison is done in learning) and is a completely symbolic parser that can parse in linear time.

Method	# Training Corpus Sents	% Accuracy
Inside-Outside	1095	90.2
Transformation Learner	1000	91.6

Table 5: Comparison of Two Learning Algorithms on the Wall Street Journal: Crossing Bracket Accuracy

A graph showing parsing performance for a WSJ run trained on a 500-sentence training corpus (training and testing on sentences of length 2-15) is shown in figure 4. 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 84.7%.

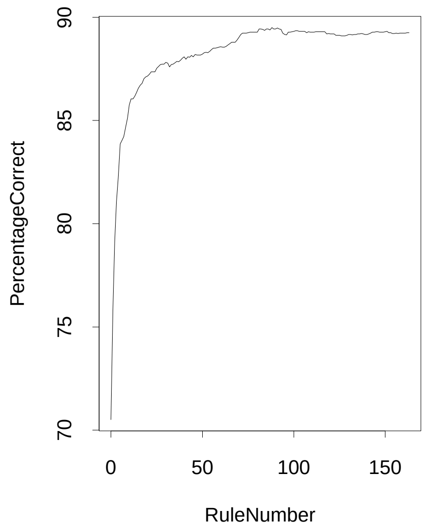


Figure 4: Accuracy as a function of transformation number for the WSJ Corpus.

Finally, in figure 5 we show the sentence length distribution in the Wall Street Journal 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 [PS92, SRO93].) 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

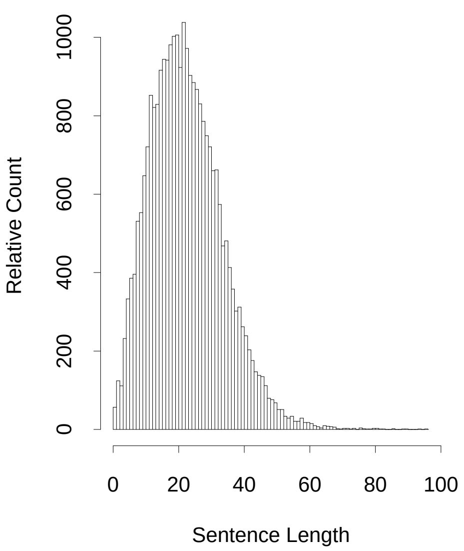


Figure 5: The Distribution of Sentence Lengths in the WSJ Corpus.

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 training corpus. Accuracy only degraded by a small amount when training on the corpus with adulterated part of speech tags.

5 SAMPLE OUTPUT

Below are ten randomly chosen parses from the Wall Street Journal. In each case, the output of the bracketing program is listed first, and the Penn Treebank bracketing is listed second. Crossing brackets are marked with a star.

((But (if *((a raider) *(takes *((over (when ((the stock) (is weak)))) (, ((the shareholder) (never (gets (his recovery))))))*)*)*)) .)
((But (if ((a raider) (takes over (when ((the stock) (is weak)))))) , ((the shareholder) (never gets (his recovery)))) .)

(((The company) (expects (to (resume *((full operations) (by today))*)))) .)
(((The company) (expects (to (resume (full operations)) (by today)))) .)

(((“ It) *((’s *((very likely) *((((the next) (five years)) (will (be (strong (for funds))))) (, ”))*)*) (he says))*) .)
(((“ It (’s (very likely ((the next five years) will (be strong (for funds))))) , ”) (he says)) .)

(((The (latest report)) (compares (with (((a modest) (9.9 (% increase))) (in *((July (machine orders)) (from ((a year) earlier)))*))))) .)
(((The latest report) (compares (with (a modest 9.9 % increase (in (July machine orders)) (from ((a year) earlier)))))) .)

(((The goal) (was (to (boost ((the circulation) (above ((the (500,000 level)) ((considered significant) (by advertisers))))))))) .)
(((The goal) (was (to (boost (the circulation) (above ((the 500,000 level) (considered significant (by advertisers)))))))) .)

(((Mr. Jones) (ran *((((for (the Senate)) (as (a Democrat))) (in 1986)) (, (but (lost (to ((incumbent Sen.) (Don Nickles)))))))*)) .)
(((Mr. Jones) ((ran (for (the Senate)) (as (a Democrat)) (in 1986)) , but (lost (to (incumbent Sen. Don Nickles))))) .)

((Then (((the ((auto paint) shop)) fire) (sent ((an (evil-looking cloud)) *(of *((black smoke) (into (the air)))*)*)))) .)
((Then ((the auto paint shop fire) (sent (an evil-looking cloud (of (black smoke))) (

into (the air))))) .)

((He (*(used *(to (be ((a boiler-room) salesman)))*)* (, (peddling (investments (*(in oil)* *(*((and (gas wells)) and)* (rare coins))*)))))) .)

((He (used (to (be (a boiler-room salesman) , (peddling (investments (in ((oil and gas wells) and (rare coins))))))))) .)

(((The board) (is (scheduled (to (meet Tuesday))))) .)

(((The board) is (scheduled (to (meet Tuesday)))) .)

((Ignore (the (present condition))) .)

((Ignore (the present condition)) .)

In the first example, there are three bracketing errors, all arising from the failure to end the clause following *if* at the comma. The second sentence has one error, which is a prepositional phrase attachment error. The third sentence has three bracketing errors, arising from crossing matching quotes. Perhaps a number of meta-rules, either learned or manually coded, such as information about matching parentheses and quotes, would significantly improve performance. The fourth sentence has one error, which is again a prepositional phrase attachment error. The sixth sentence has one error, from attaching the clause following (and including) the comma to the preposition *for* instead of the verb *ran*. The seventh sentence has two errors, both due to prepositional phrase attachment. The eighth sentence has five errors, one of which is due to prepositional phrase attachment and two arising from a difficult coordinate structure. In addition to meta-rules, postprocessors addressing particular parsing problems such as prepositional phrase attachment and coordination could lead to significant system performance improvements. Progress has already been made on a transformation-based prepositional phrase attachment program (see [BR93]).

6 ASSIGING NONTERMINAL LABELS

Once a tree is bracketed, the next step is to label the nonterminal nodes. Transformation-based error-driven learning is once again used for learning how to label nonterminals. Currently, a node is labelled based solely on the labels of its daughters. Therefore, an unlabelled tree can be labelled in a bottom-up fashion. Instead of addressing the problem of labelling the unlabelled tree output of the previous section, we have addressed a slightly different problem. The problem is to assign a tag to a node of a properly bracketed tree given the proper labels for the daughter nodes. This problem can be more easily evaluated and solving it is a significant step toward solving the problem of labelling the output of the transformation-based bracketer.

The Penn Treebank bracketed Wall Street Journal corpus was used for this experiment.[11] Two training sets were used (training set A had 1878 sentences and training set B had 1998), as well as a test set of 1971 sentences. In the first experiment, the initial state annotator assigned the label *noun phrase* to all nodes. Then, transformations were learned to improve accuracy. The transformation templates are:

1. Change the node label to X if Y is a daughter.[12]
2. Change the node label to X if Y and Z are adjacent daughters.

Transformations were learned using training set A. A total of 115 transformations were learned. Initially assigning the label *noun phrase* to all nonterminal nodes in the test set resulted in an accuracy of 44.9%. Applying all learned transformations to the test set resulted in an accuracy of 94.3%. Figure 7 shows the first twenty learned transformations. Transformations 15 and 18, as well as a number of similar transformations in the entire list capture the general rule $X \rightarrow$ X and X for co-

[11] Thanks to Rich Pito for providing corpus processing tools for running this experiment.

[12] Y can be a nonterminal or preterminal (and need not be the only daughter).

ordination. It appears that the transformation *Change a label to S if VP is a daughter* is particularly effective, appearing as transformation 2, 9 and 14. After the second transformation is applied, the transformations that follow could undo the second transformation as a side-effect. So, this transformation applies a number of times to remedy this.

Next, a less naive start state was used. A nonterminal node is assigned the most likely tag for its daughters, as indicated in a second training set (training set B). Unseen daughter sequences are tagged with a default tag (noun phrase). Transformations were learned after applying the start state annotator to training set A. On the test set, initial state accuracy was 92.6%. Applying the transformations resulted in an accuracy of 95.9%. A total of 107 transformations were learned.

We are very encouraged by the accuracy obtained using such a simple learning algorithm that only makes use of very local environments without recourse to any lexical information. Hopefully, adding richer environments such as *the word X is a daughter*, or *the nonterminal to the left is Y* will lead to an even more accurate nonterminal labeller. By first bracketing text and then labelling nonterminals, we can produce labelled parse trees in linear time with respect to sentence length. The bracketer runs in $O(|n| * |T|)$, where $|n|$ is the length of the sentence and $|T|$ is the number of bracketing transformations. The nonterminal labeller also runs in $O(|n| * |T|)$, as all transformations are tried at every nonterminal node. Therefore, parsing run time is: $O(|n| * |T|) + O(|n| * |T|) = O(|n| * |T|)$.

7 CONCLUSIONS

In this paper, we have described a new approach for learning a grammar to automatically parse text. The method can be used to obtain high parsing accuracy with a very small training set. Instead of learning a traditional grammar, an ordered set of structural transformations is learned that 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 that 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 hope 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.

The transformation-based learner can easily be extended simply by adding transformation templates. In the future, 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 for free text.

References

[Bak79] J. Baker. Trainable grammars for speech recognition. In *Speech communication papers presented at the 97th Meeting of the Acoustical Society of America*, 1979.

[BM92a] E. Brill and M. Marcus. Automatically acquiring phrase structure using distributional analysis. In *Darpa*

Workshop on Speech and Natural Language, Harriman, N.Y., 1992.

[BM92b] E. Brill and M. Marcus. Tagging an unfamiliar text with minimal human supervision. In *Proceedings of the Fall Symposium on Probabilistic Approaches to Natural Language – AAAI Technical Report.* American Association for Artificial Intelligence, 1992.

[BR93] E. Brill and P. Resnik. A transformation based approach to prepositional phrase attachment. Technical report, Department of Computer and Information Science, University of Pennsylvania, 1993. Forthcoming.

[Bri92] E. Brill. A simple rule-based part of speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing, ACL*, Trento, Italy, 1992.

[Bri93] E. Brill. *A Corpus-Based Approach to Language Learning.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1993.

[BW92] T. Briscoe and N. Waegner. Robust stochastic parsing using the inside-outside algorithm. In *Workshop notes from the AAAI Statistically-Based NLP Techniques Workshop*, 1992.

[CC92] G. Carroll and E. Charniak. Learning probabilistic dependency grammars from labelled text – aaai technical report. In *Proceedings of the Fall Symposium on Probabilistic Approaches to Natural Language.* American Association for Artificial Intelligence, 1992.

[ea91] E. Black et al. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of Fourth DARPA Speech and Natural Language Workshop*, pages 306–311, 1991.

[HGD90] C. Hemphill, J. Godfrey, and G. Doddington. The ATIS spoken language systems pilot corpus. In *Proceedings of the DARPA Speech and Natural Language Workshop*, 1990.

[HR91] D. Hindle and M. Rooth. Structural ambiguity and lexical relations. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, Berkeley, Ca., 1991.

[LY90] K. Lari and S. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4, 1990.

[MM90] D. Magerman and M. Marcus. Parsing a natural language using mutual information statistics. In *Proceedings, Eighth National Conference on Artificial Intelligence (AAAI 90)*, 1990.

[MM91] R. Weischedel M. Meteer, R. Schwartz. Empirical studies in part of speech labelling. In *Proceedings of the fourth DARPA Workshop on Speech and Natural Language*, 1991.

[MSM93] M. Marcus, B. Santorini, and M. Marcinkiewicz. Building a large annotated corpus of English: the Penn Treebank. To appear in Computational Linguistics, 1993.

[PS92] F. Pereira and Y. Schabes. Inside-outside reestimation from partially bracketed corpora. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, Newark, De., 1992.

[Sam86] G. Sampson. A stochastic approach to parsing. In *Proceedings of COLING 1986*, Bonn, 1986.

[SJM90] R. Sharman, F. Jelinek, and R. Mercer. Generating a grammar for statistical training. In *Proceedings of the 1990 Darpa Speech and Natural Language Workshop*, 1990.

[SRO93] Y. Schabes, M. Roth, and R. Osborne. Parsing the Wall Street Journal with the inside-outside algorithm. In *Proceedings of the 1993 European ACL*, Uterich, The Netherlands, 1993.

Method	# Training Corpus Sents	Sentence Accuracy
Inside-Outside	1095	57.1
Transformation Learner	1000	62.4

Table 6: Comparison of Two Learning Algorithms on the Wall Street Journal: Sentence Accuracy

Transformation Number	Tag As	If Daughter Includes
1	PP	IN
2	S	VP
3	VP	VBD
4	VP	VB
5	VP	VBN
6	VP	VBG
7	VP	VBZ
8	S	, S
9	S	VP
10	SBar	-NONE- S
11	PP	TO NP
12	SBar	IN S
13	VP	VBP
14	S	VP
15	S	CC S
16	WHNP	WDT
17	SBar	WHNP
18	VP	CC VP
19	WHNP	WP
20	ADJP	JJR

Table 7: Transformations For Labelling Nonterminals.