

# AN EVOLUTIONARY APPROACH TO NATURAL LANGUAGE GRAMMAR INDUCTION

MARGARET AYCINENA  
MYKEL J. KOCHENDERFER  
DAVID CARL MULFORD

ABSTRACT. This paper describes an approach for evolving natural language grammars using a genetic algorithm, based on part-of-speech tagged natural language examples. Grammars are represented by the genetic algorithm as a finite string of non-terminals and pre-terminals corresponding to the rules of a context-free grammar in Chomsky Normal Form. Fitness is based on the number of sentences correctly parsed by the grammar from a selection of language examples, inversely related to the number of random sentences parsed by the grammar, and discounted by the length of the string representing the grammar. Our experimentation reveals that this evolutionary approach produces grammars with high precision and recall, although they are dissimilar to grammars designed by humans.

## 1. INTRODUCTION

Grammar induction, also known as grammatical inference [1], describes a process in which a system produces a grammar given a set of corpora. Given the complexity of natural language and the increasingly high availability of large bodies of text, automated development of grammars is and will continue to be an important area of natural language processing and machine learning.

Many previous grammar induction systems have worked on the assumption that grammar induction is a subset of inductive logic programming (ILP), and as such, have primarily depended on ILP techniques. Much less work has been done on using evolutionary techniques to evolve natural language grammars.

One of the few papers applying evolutionary techniques to grammar induction, Keller and Lutz [4], used a genetic algorithm to evolve stochastic context-free grammars for finite languages. In this work, only positive examples of language data were presented. The fitness function was based on the probability of a particular grammar given the corpus, but the minimum description length principle was also incorporated so that simpler grammars would be preferred. Notably, the genetic algorithm was applied not to evolve a grammar itself, but to evolve probability parameter settings for a pre-chosen covering Chomsky Normal Form grammar. And perhaps most importantly (for this work, at least), the languages of the corpora were formal; e.g. the language of all strings consisting of equal numbers of a's and b's, and palindromes over  $\{a, b\}$ .

This paper presents some work that goes significantly beyond what was done in [4]. First, the system actually evolves non-stochastic grammars, rather than

---

*Date:* June 2, 2003.

*Key words and phrases.* grammar induction, genetic algorithm.

simply the probability parameters for a pre-chosen grammar. Second, the system uses both positive and negative examples of language data. Third, the fitness function was based on the ability of a given grammar to parse the data. Finally, the corpora were samples of part-of-speech tagged natural English language.

## 2. METHODS

In this section, we describe the evolutionary approach used to induce a grammar for a part-of-speech tagged natural language corpus. Although the approach is based on the idea of a genetic algorithm as originally presented in [3] and later in [2], we made a few significant adaptations that make the process more suitable for grammar induction.

**2.1. Genetic Algorithm.** Chromosomes represent context-free grammars with variable-length strings of non-terminals and pre-terminals. For example, the string

SABABCBCDAE

would represent the following CFG:

$$\begin{aligned} S &\rightarrow A B \\ A &\rightarrow B C \\ B &\rightarrow C D \\ C &\rightarrow A E \end{aligned}$$

We restrict the set of possible strings such that the left side of each production (i.e. every third character in the string) is a non-terminal. All of the other characters may be either pre-terminals or non-terminals.

The population is organized on the surface of a torus, represented as a two-dimensional grid with opposite sides connected (similar to [4]). The initial population is generated randomly from a uniform distribution of chromosomes of some specified length.

The genetic algorithm then executes the following select-breed-replace cycle:

- (1) Select an individual randomly from the grid
- (2) Breed that individual with its most highly fit neighbor to produce two children
- (3) Replace the weakest parent by the fittest child

In this paper, we refer to a single iteration through this cycle as a generation—even though only one individual in the population is replaced.

Given two parents, we create two children by first applying cross-over and then probabilistically applying mutation. Cross-over is accomplished by selecting a random production in each parent. Then a random point in these productions is selected and cross-over is performed, swapping the remainder of the strings after the cross-over points. Following cross-over, we iterate over each non-terminal and pre-terminal in the strings and apply mutation according to a specified probability. A mutation is simply the swapping of a non-terminal or pre-terminal with another non-terminal or pre-terminal.

**2.2. Parsing.** At the core of the fitness function is the ability to parse a sentence. Since our grammar is in Chomsky Normal Form, we can use the Cocke-Kasami-Younger (CKY) algorithm. This algorithm is efficient ( $O(n^3)$  performance) for grammars of this type. The algorithm can be easily extended to handle probabilistic context-free grammars.

Because speed is of the utmost importance, we have implemented the parser in C++. In order to make the parser as fast as possible, we have used the following implementation strategies:

- (1) When filling a square in the well-formed sentence table, we need to know which symbols can produce the two child symbols. A 2D array is used to store grammar rules. Entry  $(a, b)$  contains a vector of all parents  $c$  that produce  $a$  and  $b$  (all rules  $c \rightarrow ab$ ).
- (2) 2D arrays are simulated using a 1D array. If you have a pointer to an entry, you can increment it by 1 to advance to the next column, or increment it by `num_columns` to reach the next row. This eliminates the need for multiplication operations to index into the WFST when iterating over split points. Also, the array's size is forced to be a power of 2 so that the initial indexing operation can be done with a bit-shift rather than a multiplication.
- (3) STL vectors are used rather than STL sets for each WFST entry. We need to iterate over a table entry much more than we need to insert, so the the faster iteration provided by the vector trumps the gains from  $\log(n)$  insertion time from a set.

**2.3. Corpus Selection and Preparation.** Evolving grammars for natural language requires an appropriate selection of prepared corpora for training, testing, and cross-validation. We used a combination of self-selected (taken from online sources) and pre-selected (from the Brown linguistic data) corpora.

Specifically, the corpora were as follows:

- A selection of children's books, taken from <http://www.magickeys.com/books>.
- *The Wizard of Oz* (L. Frank Baum), taken from <http://www.ucalgary.ca/dkbrown/storclas.html>.
- *Alice in Wonderland* (Lewis Carroll), taken from <http://www.ucalgary.ca/dkbrown/storclas.html>.
- *Tom Sawyer* (Mark Twain), taken from <http://www.infomotions.com/alex/authors.html>.
- Five Brown untagged corpora, `brown1_a` through `brown1_e`, taken from [/afs/ir.stanford.edu/data/linguistic-data/Brown/ICAME-Brown1/](http://afs/ir.stanford.edu/data/linguistic-data/Brown/ICAME-Brown1/).

A wide spectrum of corpora were selected in order to compare the effectiveness of our evolutionary technique on increasingly sophisticated texts, ranging from young children through young adult to full adult.

Once the corpora were gathered, they were prepared for part-of-speech tagging: all special markings and tags used in the Brown corpora were removed, and the corpora were massaged to have exactly one sentence per line with no excess white space or new lines.

The formatted corpora were then part-of-speech tagged using the well-known Brill tagger, found at [/afs/ir.stanford.edu/class/cs224n/src/brill-tagger/Bin\\_and\\_Data](http://afs/ir.stanford.edu/class/cs224n/src/brill-tagger/Bin_and_Data).

The tagged sentences were then post-processed to remove the English words, leaving only the tags themselves, so that each line was a string of preterminals representing an actual English sentence.

After some preliminary experiments, it was determined that the tag set used by the Brill tagger was too expansive for efficient evolution of grammars. Thus, the tag set was reduced to include only the following set:

- N:** nouns, pronouns (NN, NNP, NNPS, NNS, PRP, WP)
- V:** verbs, helping verbs (MD, VB, VBD, VBG, VBN, VBP, VBZ)
- J:** adjectives, numeral, possessives (CD, JJ, JJR, JJS, PRP\$, WP\$)
- R:** adverbs (RB, RBR, RBS, WRB)
- P:** prepositions, particles (IN, RP, TO)
- T:** conjunctions, determiners (CC, DT, EX, PDT, WDT)
- O:** other (foreign words, symbols, and interjections) (FW, SYM, UH)

The final corpora were files containing strings consisting of the above seven preterminals, one sentence per line.

**2.4. Fitness Evaluation.** The objective is to induce a grammar that represents the target language. The likelihood that a grammar represents a target language given a corpus of sentences belonging to that language is related to the number of sentences parsed by the grammar. However, a grammar that parses all sentences is not useful. We must therefore penalize grammars that parse sentences that are outside of the grammar.

Simply evaluating grammars based on the number of corpus sentences that are accepted is not enough. It is fairly easy to generate a grammar that will accept every English sentence. For example, a grammar of the form:

$$S \rightarrow S X, \text{ for each terminal } X$$

$$S \rightarrow X Y, \text{ for each pair of terminals } X, Y$$

can parse any string of length at least 2. The first set of rules allows you to append any terminal to an existing string, and the last set of rules accepts any string of length 2. Initial runs of the algorithm indicated that a grammar of this form would be found:

$$S \rightarrow T N|T J|N V|N J|P N|P V|P J|R J|V J|R S|N S|P S|T S|J S|V S|S V|S J|S N$$

This grammar does not accept everything, but it does accept a lot. It contains rules to prepend as well as append symbols, which allows the bigram to appear anywhere in the sentence, not just at the beginning.

In order to correct this, we have decided to penalize the grammar for parsing sentences that are not valid English sentences. Generating sentences outside of the grammar is rather difficult without knowing the grammar. Our approach in this paper is to simply generate random sentences. Since most useful grammars impose a strong restriction on the set of possible sentences, the vast majority of randomly generated sentences fall outside of the language. In our implementation, we generated random strings of part-of-speech tags of length 5–15 from a uniform distribution.

Since extremely long and complex grammars are generally not as useful and are less likely to be correct by the principle of *Occam's razor*, we decided to exponentially discount the fitness of an evolved grammar based on the length of its chromosome.

Our implementation used the following formula to calculate the fitness of an evolved individual  $\alpha$ .

$$F(\alpha) = \gamma^{\max(0, |\alpha| - |P|)} C(\alpha) - \delta I(\alpha)$$

where  $P$  is the set of preterminals,  $C(\alpha)$  is the number of parsed sentences from the corpus,  $I(\alpha)$  is the number of sentences parsed from the randomly generated corpus,  $\delta$  is the penalty associated with parsing each sentence in the randomly generated corpus, and  $\gamma$  is the discount factor used for discouraging long grammars.

### 3. EXPERIMENTATION AND DISCUSSION

**3.1. Parameters.** Based on a few preliminary experiments the following parameters were settled upon. We made no attempt to further tune these parameters.

- The discount factor  $\gamma$  was set to 0.98.
- The size of the population grid was set to  $10 \times 10$  to store the individuals.
- The mutation rate was set to 0.01.
- The number of non-terminals was set to 8.
- The initial random population consisted of grammars with 10 rules.

**3.2. Feasibility Experiment.** To demonstrate the potential feasibility of our approach to grammar induction, we experimented with synthetically generated corpora from simple grammars. When ran our algorithm on sentences generated from the palindrome grammar:

$$\begin{aligned} S &\rightarrow A SA|B SB|A A|B B \\ SA &\rightarrow S A \\ SB &\rightarrow S B \end{aligned}$$

the algorithm found:

$$\begin{aligned} 0 &\rightarrow A 3|B 7|A A|B B \\ 3 &\rightarrow 0 A \\ 7 &\rightarrow 0 B \end{aligned}$$

and when run on:

$$\begin{aligned} S &\rightarrow A SA|B SB|C SC|A A|B B|C C \\ SA &\rightarrow S A \\ SB &\rightarrow S B \\ SC &\rightarrow S C \end{aligned}$$

the algorithm found:

$$\begin{aligned} 0 &\rightarrow A 1|B 7|C 2|A A|B B|C C \\ 1 &\rightarrow 0 A \\ 7 &\rightarrow 0 B \\ 2 &\rightarrow 0 C \end{aligned}$$

In both cases, the exact grammars were found within a reasonable amount of time—6,000 generations for the two character palindromes and 20,500 generations for the three character palindromes.

**3.3. Training Results.** In order to train the system and still have data on which to test our solutions, we divided each corpus into two parts. Every third sentence was reserved for the test corpus, and the remaining two-thirds became the training corpus.

We ran the evolution program on each training corpus for 200,000 generations, with the parameters given above. A complete run took anywhere between 36 and 60 hours, and as such, approximately half of the runs did not finish. Fig. 3.3 shows the number of generations each run was able to complete, the last grammar that evolved, and some statistics relating to that grammar: the percentage of positive examples parsed, the percentage of negative examples parsed, and the fitness. The grammar is given in string form.

Corpus	Number of generations completed	grammar	% positive examples parsed	% negative examples parsed	fitness
aliceinwonderland	200000	0V00R000J0VN0RJ0PN0PJ00 N0V0VJ0P00NN0N00TN0J00 T00TJ	92.50%	8.40%	657.364
brown1_a	48500	0N442N0TN0J420V0P03RN0N 040N0NV0R27T00T400T2V00 0P40V00V0R00J0V400N0JN	94.10%	6.10%	1667.85
brown1_b	200000	0JJ0NJ0R00N00T000N0J00V0 0VN0P00TN0TJ0PN0RJ0PJ0V J	94.70%	6.70%	1227.17
brown1_c	15500	0447R47TT1VJ22P40J5P72JN 40T71T4V42O140V1044PT03P 70600N0R77JN6JT4P11T6575 6N24JP70N07J4P500R4NN2V V52T4PN34N61504J0NV77J0 N44056N24JN0R622N5574RP 0NT4T004P0NP3V01154RO	80.50%	4.70%	302.996
brown1_d	45000	3V000J5TV00V0R230N0NV0T N1T33NN00T3TV00N0T33N00 J300P03020R20P00R30T0P30 NP	88.20%	5.60%	583.596
brown1_e	122000	0PJ0P00V00N00TN0T00TJ0VJ 0JJ0RJ0J00R00PN00N0NJ	93.90%	5.90%	1762.67
children	200000	0PJ0V00VN0JJ0NJ00N0N00TJ 0VJ0J00R00RJ0P00TN0T0	91.80%	5.70%	677.211
tomsawyer	200000	0PJ0VN0NN00V0T00TN0TJ00 N00J0NJ0P00PN0VJ0V00R00 J00N00RJ	92.70%	8.60%	2292.89
wizardofoz	200000	000OPJ0V00R00P00RJ0VN 0TN00000J0T00PN0NV0VJ 0TJ0N00N0JN	89.50%	9.20%	920.852

FIGURE 1. Grammar evolution results

These results bring up several important points. First, our approach clearly does a decent job of finding a grammar which parses a majority of the accepted sentences, and a minority of the incorrect sentences. Second, one of the most important roles of iteration appears to be that it shortens the grammar. The grammar for **brown1\_c** is extremely long, but it was taken from a much younger generation (15500) than the other runs.

**3.4. Testing Results.** The results of the training sessions above can only be fairly evaluated by testing them on some unseen but related corpora. For this reason, as mentioned above, we held out every third line of each original corpus as a test

corpus. We tested a solution grammar on a corpus by measuring its precision and recall, and combining these values into the well-known  $F$  measure.

As described on p. 268-269 of Manning and Schütze [5], *precision* is the measure of the proportion of selected items that the system got right:

$$P = \frac{\#truepositives}{\#truepositives + \#falsepositives}$$

*Recall* is defined as the proportion of the target items that the system selected:

$$R = \frac{\#truepositives}{\#truepositives + \#falsenegatives}$$

The  $F$  measure combines these terms using a harmonic mean:

$$F = \frac{1}{(\alpha(1/P) + (1 - \alpha)(1/R))}$$

where  $P$  is precision,  $R$  is recall, and  $\alpha$  is a factor that determines the relative weighting of  $P$  and  $R$ .

The graph in fig. 3.4 gives the precision, recall, and  $F$  measure for the grammar evolved from each corpus, when tested on the test set of the same corpus.

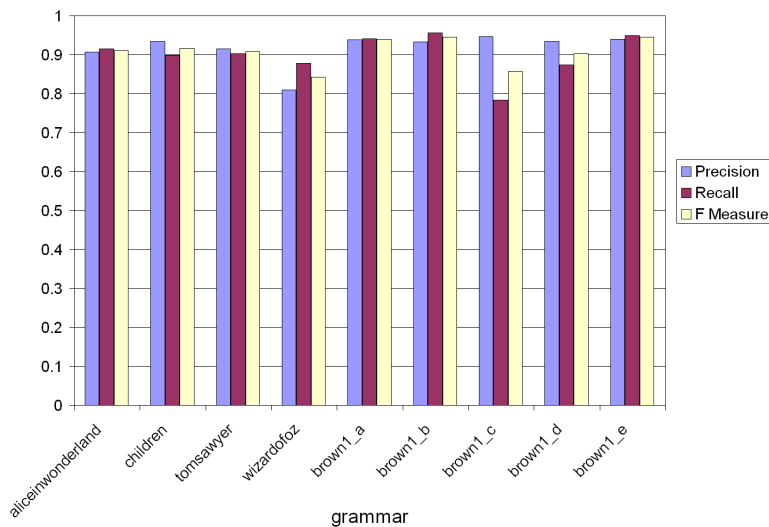


FIGURE 2. Precision, Recall, and  $F$  measure on the test corpora

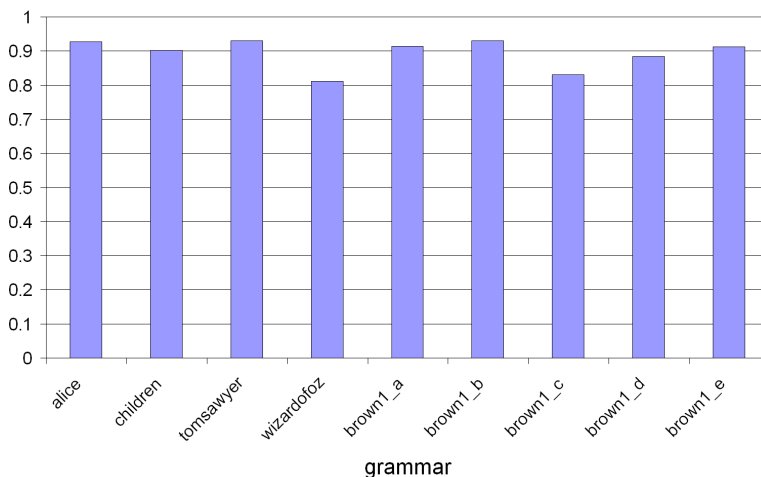
**3.5. Cross Validation.** Testing the evolved grammar on a held-out test set of the original corpus is an important evaluation tool. However, another, perhaps more important, question is how well the grammar evolved by training from a particular corpus can parse an entirely unrelated corpus. We perform cross-validation by testing the precision and recall of each grammar on each of the other corpora, and calculating the  $F$  measure.

In fig. 3.5, the rows correspond to the grammar evolved from the labelled corpus, the columns signify on which corpus the grammar was cross-validated, and the value itself is the  $F$  measure for that particular combination.

	alice	children	tomsawyer	wizardofoz	brown1_a	brown1_b	brown1_c	brown1_d	brown1_e
alice	0.911591	0.906504	0.913092	0.937697	0.944986	0.938218	0.931677	0.924354	0.942857
children	0.875653	0.916062	0.876955	0.918206	0.902412	0.918485	0.898113	0.885602	0.929317
tomsawyer	0.912488	0.919517	0.908583	0.940289	0.945024	0.941562	0.931556	0.927189	0.941222
wizardofoz	0.810916	0.751055	0.743917	0.842564	0.852771	0.832609	0.830553	0.819432	0.823074
brown1_a	0.919242	0.892405	0.863258	0.928385	0.939155	0.933708	0.923923	0.920152	0.908126
brown1_b	0.89858	0.921348	0.89914	0.935733	0.951108	0.944568	0.938739	0.929577	0.947574
brown1_c	0.807649	0.758454	0.723014	0.841649	0.897232	0.866748	0.857711	0.867347	0.856021
brown1_d	0.864979	0.835754	0.811912	0.902225	0.928623	0.917051	0.916357	0.902913	0.878954
brown1_e	0.851613	0.914226	0.855942	0.90275	0.943153	0.936194	0.938701	0.929389	0.944576

FIGURE 3. Cross-validation  $F$  measure

To get a better picture of the results, we average the  $F$  measure for a given grammar over all the corpora on which it was validated. This gives the graph in fig. 3.5, in which the  $x$ -axis is the grammar, and the  $y$ -axis is the mean  $F$  measure.

FIGURE 4. Precision, Recall, and  $F$  measure on the test corpora

From the testing and cross validation results, it appears that the approach has been quite effective.

#### 4. CONCLUSIONS AND FURTHER WORK

Here is a sample grammar produced when using negative examples:

$$S \rightarrow P \text{ J|P N|T N|T J|V J|J J|R J|N J|P S|V S|N S|J S|R S|T S|S N}$$

This grammar differs from the “accept-all” grammar in that it only allows you to append one symbol, a noun. For most English sentences, you can append a noun and still have a string of POS tags that could be a valid sentence. For example, if the sentence ends with a noun, the new noun becomes part of a compound noun. If the sentence ends with an adjective, the appended noun becomes the object of that adjective. If the sentence ends in a verb, the noun becomes the object of that verb. The genetic algorithm has caught onto that fact in the final rule. It has found a subset of all bigrams that are the most common in English. The grammar accepts any string of words that ends with one of those bigrams, optionally followed by any number of nouns. This, unfortunately, gives a grammar that is very capable of

detecting whether a sentence is valid in English, but it has not learned much English structure. The grammar has the property that it accepts too many sentences that are not valid English, but still rejects most non-English sentences.

We have tried generating synthetic corpora using a small English grammar. One grammar we tried was:

$$\begin{aligned} S &\rightarrow NP VP|N VP|NP V|N V \\ NP &\rightarrow N N|T N \\ VP &\rightarrow V NP|V N \end{aligned}$$

For this corpus, the algorithm output the following grammar:

$$\begin{aligned} S &\rightarrow 6 V|S 6|N V|S N \\ 6 &\rightarrow T N|N N \end{aligned}$$

This is actually a somewhat reasonable grammar. The unnamed non-terminal 6 could be interpreted as a noun phrase. However, for more complicated synthetic grammars, the algorithm again output grammars only using the start symbol.

We have explored several approaches to coax the algorithm into producing better grammars. If you forbid the start symbol to expand to itself, the algorithm just finds a grammar using  $S$  and one other non-terminal. Increasing the penalty for parsing negative examples does not really help: the original grammar already did not parse many negative examples.

A possible way to fix this would be to include the probability of parses. Since the above grammar only uses one non-terminal, on average each rule has a low probability. Thus, the trees assigned to the corpus would have low probabilities as well. A grammar that uses the proper phrasal units would be able to assign higher probabilities to each rule, and the algorithm might discover such a grammar.

However, it is still possible that English grammar is too complex to be learned from a corpus of words. When parents teach their children language, they provide subtle clues as to the structure of English. These clues could take the form of the rhythm of speech, as speakers might put space between their pronunciation of phrasal units (i.e. separating the subject, verb, and object from each other).

#### REFERENCES

1. E. M. Gold, *Language identification in the limit*, Information and Control **10** (1978), 447–474.
2. D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*, Addison-Wesley, Boston, 1989.
3. J. H. Holland, *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor, 1975.
4. B. Keller and R. Lutz, *Evolving stochastic context-free grammars from examples using a minimum description length principle*, Workshop on Automata Induction Grammatical Inference and Language Acquisition, ICML-97 (1997).
5. C. Manning and H. Schütze, *Foundations of statistical natural language processing*, The MIT Press, Cambridge, 1999.

STANFORD UNIVERSITY, STANFORD, CA 94305

*E-mail address:* aycinena@cs.stanford.edu, mykel@cs.stanford.edu, dmulford@cs.stanford.edu