"SUE ATE AN APPLE AND FRED A PEAR" – A TUTORIAL ON PARSERS AND EVOLUTIONARY ALGORITHMS

Sanjay. Patil
Center of Advanced Vehicular System
Mississippi State University
patil@cavs.msstate.edu

ABSTRACT

Parsing the sentence as mentioned in the title of the paper requires the parser to overcome ambiguity. Natural language processing (NLP) is a task to understand the conversational speech by using the knowledge of language. Automatic speech recognition (ASR) systems are deployed for understanding conversational speech. The major step in improving the performance of ASR is to parse the words correctly. Probabilistic structure for the parsers has shown improvement in the performance, but still more needs to be done. This paper tries to mesh natural evolution concepts with the probabilistic parsers (PP) with an intention of seeking improvement in the performance. This paper explains the concept of genetic algorithms (GA) and how it can be used along with PP through some stripped down illustrations. Finally, the paper concludes by anticipating future research directions.

1. INTRODUCTION

Starting with a complete sentence as a root and then by chopping it into smaller segments or constituents, the semantic and syntactic purpose of the sentence can be understood. By speaking to a machine and by making it understand your intention has been the foremost goal of scientists working in the field of speech recognition. In the application of machine translation, NLP task is to determine the lexical tag for each word, parsing sentences and determining the *expression* of the sentence.

Probabilistic parsers have been used to improve the performance. Few are of the view that, PCFG parsers are based on the psycholinguistic research [13]. But ambiguity with differentiating the words for a particular intention is a difficult task. The ambiguity arises because of the large search space, lack of training dataset, and many other factors. Heuristic methods have been used. One such heuristic technique is evolutionary algorithms and genetic algorithms are one such evolutionary approach. Research [13] also hints that human parsing tries to turn on and off a particular rule, and often do not apply rules sequentially, indicating a possibility of adopting evolutionary approach in tandem with PCFG parsers.

The task dealt within this paper is to understand how GAs can be used along with the present PP to improve the parsing performance.

The paper is developed in following way. The next section explains the basics related to parsers and the ambiguity associated. Following which, GAs is explained in subsequent sections. In section 4, the application of Gas to PPs is detailed out. Finally, the paper concludes with the research directions that can drive the improvement in the performance of PPs by coupling GA.

2. PARSERS

Parsing is defined as a process of taking an input and producing some sort of structure for it [2]. Parsing is a very important step in recognition and generation of natural language. *Syntactic parsing* is typically defined as the task of recognizing a sentence and assigning a syntactic structure to it. Such a parser can be deployed as:

1. Grammar checking block in word-processing systems,
2. Semantic analysis block in machine translation, question answering, and information extraction,
3. Most important, to assist language models in speech recognition.

Most of the parsers are based on context free grammars (CFG). CFGs are often referred to as phrase structure grammars. A CFG has four parameters: 4-tuple – (N, Σ, P, S), with, N being the set of non-terminal symbols, Σ the set of terminal symbols, P being set of productions, a set of rules relating a non-terminal to another non-terminal or a terminal symbol, and lastly, S is the designated start symbol. Consider the following example shown in Figure 1.

Set of non-terminal symbols : NP, Det, Nom, Noun
Set of terminal symbols : A, flight
Rules:

S	→	NP
NP	→	Det Nom
Nom	→	Noun
Det	→	a
Noun	→	flight

Figure 1: An example of context-free grammar (CFG)

In this example, I have adopted the standard abbreviations: NP for *noun phrase*, Det for *determiner*, Nom for *nominal*. Subsequently, based on the CFG in the example, a parse tree can be developed as in Figure 2.

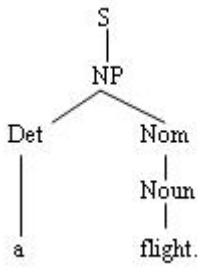

Figure 2: Parse tree fro the CFG described above

For the CFG described above, we have a single parse tree and a single combination. But, as the number of terminal and non-terminal symbols increases, the number of rules will as well increase and thus many more parse trees be generated. Naturally, the complexity to tackle with ungrammatical sentences *has* to as well increase.

Consider an example shown in the Figure 3 below. The CFG rules are not described here, but the example is stated to describe the complexity involved. The parse tree was developed with the help of Machinese natural language parser [15].

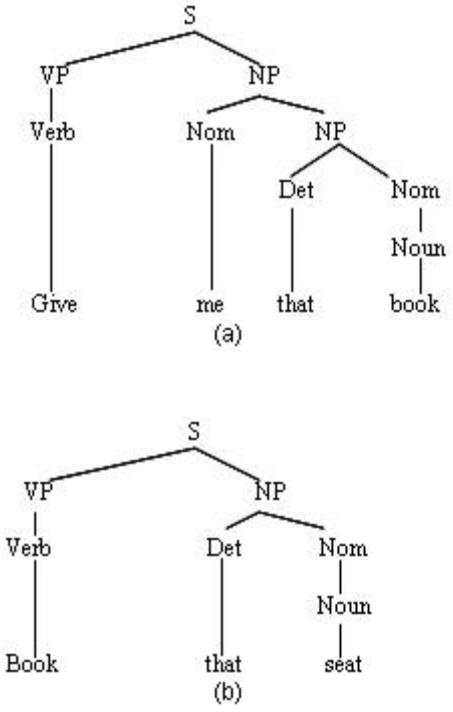

Figure 3: Parse tree which resolves ambiguity

The above example is to illustrate that ambiguity can arise in parsing the word *book*. But, the grammar defined within does take into account *book* as a verb as well as a noun. So, the parsing algorithm parses the two occurrences of *book* correctly.

But, it important to mention here that, CFGs on its own will not define the parsing algorithm [13]. CFGs define language, but not the parsing algorithm.

As illustrated in Figure 4, CFG will define the structure for language, but the parsing algorithm will decide how to utilize the language to find *structure* in the input sentence.

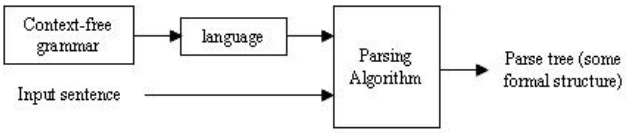

Figure 4: Relation between CFG and parsing algorithm

CFG parsers are mostly dynamic programming based algorithms, Earley algorithm (top-down parsers), Cocke-Younger-Kasami (CYK) and Graham-Harrison-Ruzzo (GHR) algorithm to name a few of them.

Parsers based on probabilistic structures will utilize it in one of the following three ways:

1. as language model, to determine what someone probably said,
2. as a search pruning approach,
3. as one-in-many decision maker [1].

For our case study, I am considering the third option, that of using PCFG parsers to resolve ambiguity.

What would define the parser performance? Parser performance is application specific, training and test corpora specific, disambiguation ability specific.

Not all sentences fed to a parser can generate a grammatically correct parse tree, and / or, it is likely that parser might generate more than one grammatically correct parse trees (based on the rules, productions, and the way in which the algorithm is implemented).

A parser has a responsibility to resolve as many as ambiguities as possible. A parser gets a sentence as an input, it has a set of symbols (term used loosely to include terminals and non-terminals), and a set of productions (which related terminals to terminals as well as terminals to non-terminals) and defines a criteria (called as derivation, and language). Based on all of the above information, parser will,

1. firstly break down the input sentence into smaller subsections within the production constraints,
2. secondly, this break-down process will continue till a terminal is reached (top-down approach) or a root symbol is reached (bottom-up),
3. after which generate a parse tree, linking the root symbol to the terminals.

All above steps seem to obvious, if both the input sentence and the set of productions are well within the bounds of grammar and language. But, still we can foresee a problem (or a couple of them):

Firstly, ambiguity may arise if parser finds more than one parse tree. Secondly, it many find none. Thirdly, the parse tree found may not be appropriate one. All these problems hint towards a modification in the parsing algorithms. Let us consider Figure 5 to exemplify the ambiguity issue a little more.

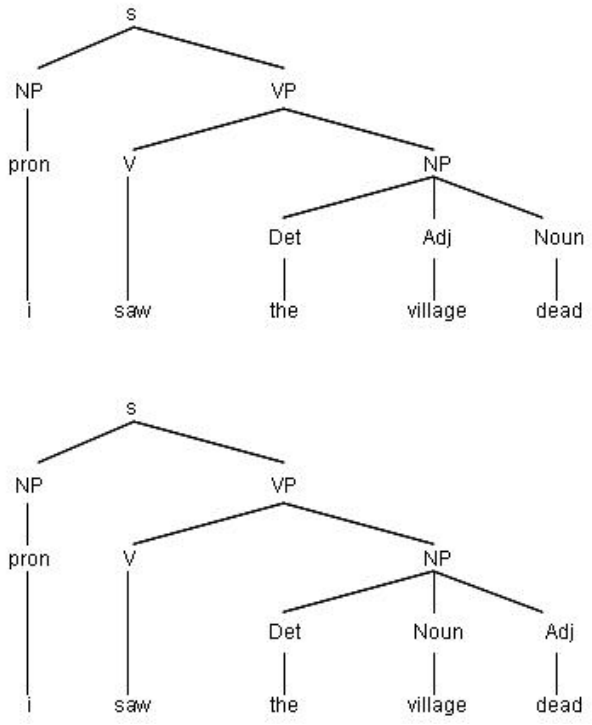

Figure 5: Ambiguity: one of the parse trees is inappropriate

Second of the parse trees shown in Figure 5 is more appropriate [12],[15]. Even if it *seems* obvious, S. Crain and J Fodor analyzed and stated that, grammar must assist parser in removing the sources of ambiguities [13]. Associating probabilities to different parse trees and then deciding on the most likely one to be the most probable tree can be one of the remedies, for which probabilistic CFG (PCFG) parser seems to be a viable option. The difficult to parse sentences need a dedicated rule or heuristics to resolve the ambiguities. Hence, some other combination with PCFG parsers could increase the chances of improving the performance.

Before I move on to explain how associating GA with PCFG parsers can help improve the performance, let us understand what genetic algorithm is and various terminologies associated with GA.

3. GENETIC ALGORITHM

Genetic Algorithms are modeled on natural evolution phenomena. The features existing in nature which are adopted in genetic algorithms and form the base are:

1. Basic building blocks of living beings are chromosomes. These chromosomes are operated on during the evolution process. Hence, an individual used in the computation is sometimes called as *chromosome*.
2. which chromosome to be selected to *father* the next generation is based on its ability to represent its best chance in representing survival.
3. there is a possibility that next generation may contain altogether different characteristics not seen in the parent.
4. the evolution process does not hold history i.e. the information of how the next evolution is evoked in not dependent directly on its preceding steps, but some sort of mechanism can be incorporated to that effect.

Each feature mentioned above acts as a plus for using genetic algorithms in parsing application, because genetic algorithms (GA) is just an approach, a skeleton which can be morphed to suit PCFG ideology.

GA finds its application in many areas of aircraft design, routing design for telecommunication networks, synthesis of neural network architecture [3][7][11][13].

GA can be broadly described as shown in Figure 6.

The algorithm starts with initializing the population. Thus, the number of individuals at the start must be known or else can be assumed based on some heuristics. Naturally, we will be seen ahead in the explanation, GA allows rather large degrees of freedom which can be controlled by the application. Because of this attribute, GA can easily mesh with the application and can become application specific. Linking GA to the application converts the algorithm from being application-alien to implementation specific. The next step is to evaluate the individuals within the population. This evaluation function is again developed based on the application or problem statement, after which the next generation is created based on *"survival of the fittest"* and *diversity criterion*. The mix of both these factors is again application specific. Each of the two factors mentioned above – survival of the fittest and diversity criterion – signifies the convergence rate and options to have different (multiple) solutions.

In the next step, couple of members are deleted or declared dead. The idea is to give sufficient enough survival chance to the next generation. Now GA steps ahead to evaluate the newer generation based on the *fitness* function. After which the algorithm decides to continue or stop the whole process. Thus, all the above steps baring the initialization one are iterated the number of times specified or target is achieved.

The Genetic Algorithm

1. Initialize a population of chromosomes (an individual).
2. Evaluate each chromosome in the population. Store the best evaluated individuals as *best-ever* individuals.
3. Create new chromosomes by mating current chromosomes; apply *mutation* and *crossover* as the parent chromosomes mate.
4. Delete members of the population to make room for the new chromosomes.
5. Evaluate the new chromosomes and insert them into the population. Update the *best-ever* individuals list.
6. If time is up, stop and return the best chromosome (*best-ever* list); if not, go to 3.

Figure 6: Procedure for simple genetic algorithm

Before we combine GA with PCFG parsers, let us understand a couple of more concepts related to GA.
GA works with two operators – mutation operator and crossover operator [9].

As shown in Figure 7, the mutation operator performs changes in the pattern of a chromosome (an individual) at random. The location at which the change will take place and the change that will take place will be decided by the problem statement. In the illustration shown in Figure 7 and Figure 8, the values with bold face positions are the one that will be changed. The values a location takes is either 1 or 0, for the example illustrated (thus we have an example set up for binary chromosome). Figure 7 shows two examples, first one represents changes at two locations while the second example shows change taking place at only one location. For mutation to take place, only one parent individual is required.

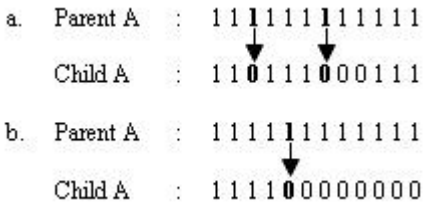

Figure 7: Examples on mutation operation

Crossover operator produces two individuals for the next generation by combining two parents. As shown in Figure 8, the parent individuals are segmented at n randomly selected points after which the segments are switched. For example a, two segments will be changed, and in example b, one segment undergoes change. So, the value associated with parent A will be interchanged with the values associated with the mating parent B at the segments to be switched.

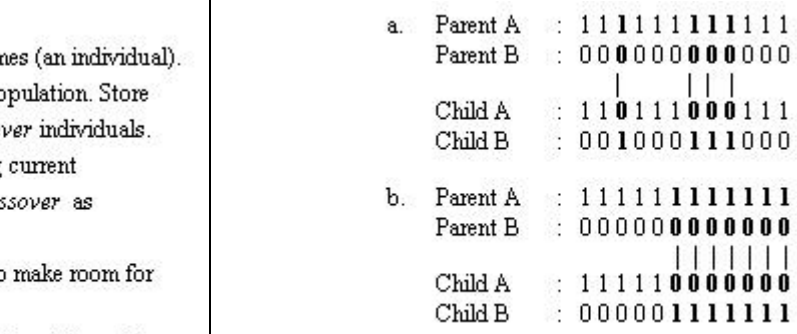

Figure 8: Examples on crossover operation

4. EVOLUTIONARY PARSING

To recap, parsing is the process of associating structure to a sentence according to a particular grammar. To overcome the ambiguities with parsing, probabilistic CFG parsers is one of the viable solutions.
Probabilistic parsing model is defined as a model which evaluates the probability of different parse trees T for a sentence S according to some grammar G by finding:

$$P(T \mid S,G) \quad where \quad \sum_{T} P(T \mid S,G) = 1 \qquad (1)$$

Based on the above model, the parser finds the most probable parse of a sentence $\hat{T}$:

$$\hat{T} = \arg\max_{T} P(T \mid S,G) \qquad (2)$$

Finding the probabilities has various options which are dealt elsewhere [1][2][10]. The major concern of this paper is, given these probabilities, how to incorporate GA to help improve the chances of getting a correct parse.

A couple of variations are explained in [5] [6] but the scheme used by this paper is also discussed in [7] [9].

Genetic algorithm as described in Figure 6 can be used with the PCFG parsers by initializing the potential parsers for the sentence based on grammar and language. The two genetic operators – mutation and crossover – are applied on the initial guesses, which will modify these potential parsers. But, it is possible that the next generation will either move away or close to the *best-ever* parser tree.

The *fitness* function is used to find the distance between the *feasible* and *probable* parse of the input sentence. This is done by defining the feasibility criterion. Feasibility criterion is defined as the number of *grammatical* correctly grammar rules applied to an individual parser. The probability of a parse will be the product of the probabilities of such grammar rules.

As discussed earlier, GA does not specify the fitness function neither does it provides the deadline on the number of iterations to be used. Hence, PCFG parsers and the grammar rules will define the constraints for *evolution* towards and *generation* of the *best-ever* parse tree.

Let us consider the GA for the example sentence – *the man sings a song*. The example is adopted from [7]. For the input sentence by referring to the PCFG rule-book, each individual word is categorized. Based on the start symbol rules, the individuals are defined. The details of two (from the many possible) individuals are listed below. The breakup from the terminals to the start symbol is listed for each individual. The example given below is a stripped down example to briefly illustrate the utility of GA for PCFG parsing. It is easily possibly that we would have more than two possible individuals (contenders) for the best possible parse.

Next, the words in the sentence are randomly segmented to have crossover and mutation operations. The main idea is to swap the places between the two (in our example) individuals. Mutation works on an individual and crossover on two individuals. The probability by which an individual undergoes mutation and / or crossover is defined at the start. Initially, as the search space is vast, it is ideal to have large number of competing individuals mutated or crossover-ed. After a couple of iterations, the rate of mutation and crossover can be reduced. These steps are illustrated in the subsequent figures (the mathematical rigor within the process is omitted, so that the reader can appreciate the process rather than tangle him (or her) in the math).

(1) NP: The man (2) VP: sings a song	(1) NP: The man (2) VP: sings a song
(1) NP → Det, (3) NP: Det: The (3)NP: man (3)NP → noun noun: man	(1) NP → Adj, (3) NP: Adj: The (3)NP: man (3)NP → noun noun: man
(2) VP → verb, (4) NP: Verb: sings (4)NP: a song (4)NP → (5) NP, (6) AP (5) NP → noun noun : a (6) AP → Adj Adj: song	(2) VP → verb, (4) PP: Verb: sings (4)PP: a song (4)PP → prep, (5) NP prep: a (5) NP → noun noun: song
Individual I	Individual II

Figure 9: Individuals (chromosome) for "the man sings a song" (adopted from [8])

During the crossover operation, the parses which might have inconsistent number of words in the sentences are avoided and also precaution is taken so that the crossover-ed segment should not again hold its previous syntactic category.

The crossover shown in Figure 10 is performed about the determiner *a*.

(1) NP: The man (2) VP: sings a song	(1) NP: The man sings (2) VP: a song
(1) NP → Det, (3) NP: Det: The (3)NP: man (3)NP → noun noun: man	(1) NP → Adj, (3) NP: Adj: The (3)NP: man (3)NP → noun noun: man
(2) VP → verb, (4) PP: Verb: sings (4)PP: a song (4)PP → prep, (5) AP prep: a (5) NP → noun noun: song	(2) VP → verb, (4) NP: Verb: sings (4)NP: a song (4)NP → (5) NP, (6) AP (5) NP→ noun noun: a (6) AP → adj adj: song
offspring I	Individual II

Figure 10: crossover operation keeping the syntactic category same

Mutation must be performed on the individual who is *unfit* to represent the best parse. As explained in Figure 11, mutation is done around the first VP, thus yielding the correct (*best-ever*) parse for the input sentence.

(1) NP: The man (2) VP: sings a song
(1) NP → Det, (3) NP: Det: The (3)NP: man (3)NP → noun noun: man
(2) VP → verb, (4) NP: Verb: sings (4)NP → Det, (5) NP Det: a (5) NP → noun noun: song

Figure 11: mutation operation on the individual I

In this manner the process continues till either the number of iterations specified are reached or the incremental improvement in the results is very less (this can be defined within the implementation).

Experiments conducted elsewhere have proved that using GA for parsing has performance comparable as seen from other implementations without GA [7]. GA can help in

reducing the search space by concentrating on a particular set of possible parse trees at a time. But, the biggest plus with GA is that the search space worked on does take into account all the possibly correct parses as none are discarded straight away during any of the steps.

5. CONCLUSIONS

In this paper the attempt was made to explain the possibility of using evolutionary algorithms, specifically genetic algorithms for parsing. Different experiments carried elsewhere [7][9] indicate a strong correlation in performance improvement using GAs. It has been also reported that the computational overheads are minimal.

I feel that extensive experiments needs to be done before the GA patch can heal ambiguity wounds caused in parsing by the PCFG parsers.

Also with vast computational resources available nowadays, parallel implementation of evolutionary parsing needs to attempt. But, definitely GAs promise to probe nw directions to investigate the statistical nature of the language.

6. REFERENCES

[1] C. Manning and H. Schutze, "Foundations of Statistical Language Processing," The MIT Press, 1999.

[2] D. Jurafsky and J. Martin, "Speech and Language Processing – An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition," Pearson, 2000.

[3] D. Whitley, "Genetic Algorithms and Evolutionary Computing," Van Nostrand's Scientific Enclyopedia, 2002.

[4] D. Whitley, "A Genetic Algorithm Tutorial," *Statistics and Computing*, vol 4, pp 65-85, 1994.

[5] D. Vrajitoru, "Evolutionary Sentence Building for Chatterbots," [online] http://citeseer.ist.psu.edu/596573.html.

[6] M. Aycinea, M. Koshenderfer, and D. Mulford, "An Evolutionary Approach to Natural Language Grammar Induction," Final Paper Stanford CS224N June 2003.

[7] A. Roberts, "Machine Learning in Natural Language Processing," October 2003 [online] http://www.comp.leeds.ac.uk/andyr/misc/latex/sessions/bibtex/bib_example.pdf

[8] L. Araujo, "Symbiosis of Evolutionary Techniques and Statistical Natural Language Processing," *IEEE Transactions on Evolutionary Computation*, vol 8, no 1, pp 14-27, February 2004.

[9] D. Kazakov, "Natural Language Processing Applications of Machine Language," PhD Thesis, Czech Technical University, Prague, May 1999.

[10] E. Charniak, "Statistical Techniques for Natural Language Parsing," *AI Magazine*, vol 18, no 4, pp 33-44, August 1997.

[11] L. Davis (Ed.), "Handbook of Genetic Algorithms," Van Nostrand Reinhold, 1991.

[12] M. King (Ed.), "Parsing Natural Language," Academic Press,1983.

[13] D. Dowty, L. Karttunen, and A. Zwicky, "Natural Language Parsing – Psychological, Computational, and Theoretical Perspectives," Cambridge University Press, 1985.

[14] Whitmore, J. and Fisher, S., "Speech during sustained Operations," *Speech Communication*, vol. 20, pp. 55–70, 1996.

[15] Machinese Language Tool [online] http://www.connexor.fi