An Interactive Software Tool for Parsing English Sentences

Claire M. Nelson
Oberlin College
Oberlin, OH
cmnelson@oberlin.edu

Rebecca E. Punch
Oberlin College
Oberlin, OH
rpunch@oberlin.edu

John L. Donaldson*
Oberlin College
Oberlin, OH
jdonalds@oberlin.edu

ABSTRACT

As natural language processing becomes an increasingly relevant field, there is a need for treebanks catered to the specific needs of more individualized systems. The current tools to construct such a treebank lack the clarity and utility to effectively and efficiently complete such a task; therefore, an interactive parser tool that caters more fully to the needs of a human annotator is necessary. This paper describes the implementation of an interactive parser tool which includes the addition of several utilities to facilitate the annotating process, including parse constraints and a manual mode to be used when the grammar cannot produce the correct tree.

1. INTRODUCTION

Natural language processing, a branch of artificial intelligence that deals with the analysis and interpretation of human languages, has become increasingly relevant as people begin to rely more and more on computers for aid in communication and information compilation. From translation between languages to automatic spell checking, natural language processing has helped shape the way people interact with computers and each other. The field has huge potential for growth, as well, given the overwhelming amount of textual corpora available which can be analyzed with the aid of linguistic models and machine learning.

1.1 Parsing, PCFGs, and the CKY Algorithm

A Context-Free Grammar (CFG) is a "mathematical system for modeling constituent structure" in natural languages [4], consisting of rules for the syntax of the grammar, as well as a lexicon of syntax and associated words. More formally, each rule in a CFG begins with a single start symbol, such as a type of phrase, followed by the constituents of that symbol. The constituents may be either a terminal symbol associated with a word in the lexicon (e.g. *Verb*), or a non-terminal symbol, associated with a symbol defined by its own set of constituents (e.g. *NounPhrase*). For example, consider the rules defining a *VerbPhrase* in this CFG (for simplicity, we will abbreviate a *NounPhrase* to *NP*, and a *PrepositionalPhrase* to *PP*):

$$\begin{aligned}
&\textit{VerbPhrase} \rightarrow \textit{Verb}\\
&\textit{VerbPhrase} \rightarrow \textit{Verb NP}\\
&\textit{VerbPhrase} \rightarrow \textit{Verb NP PP}\\
&\textit{VerbPhrase} \rightarrow \textit{Verb PP}\\
&\textit{VerbPhrase} \rightarrow \textit{Verb NP NP}\\
&\textit{VerbPhrase} \rightarrow \textit{VerbPhrase NP}
\end{aligned}$$

Still, this representation of grammar falls short in practical applications. If every possible combination of grammatical types is represented within a CFG, there is a high likelihood that a rarely-used rule will be given the same weight as a commonly-used rule. This becomes problematic especially in parsing natural language, where the best possible parse is more likely to be a commonly-used structure than a rare one. To address this problem, a Probabilistic Context-Free Grammar (PCFG) expands the structure of a CFG to include the probability that a given rule in the grammar will be used. For example, by adding probabilities to our previous example of a CFG, we can produce the following PCFG (probabilities from [4]):

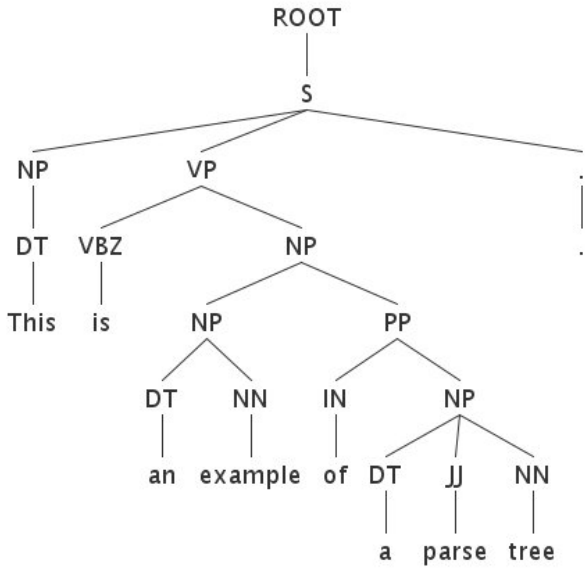
$$\begin{aligned}
&\textit{VerbPhrase} \rightarrow \textit{Verb [0.35]}\\
&\textit{VerbPhrase} \rightarrow \textit{Verb NP [0.20]}\\
&\textit{VerbPhrase} \rightarrow \textit{Verb NP PP [0.10]}\\
&\textit{VerbPhrase} \rightarrow \textit{Verb PP [0.15]}\\
&\textit{VerbPhrase} \rightarrow \textit{Verb NP NP [0.05]}\\
&\textit{VerbPhrase} \rightarrow \textit{VerbPhrase NP [0.15]}
\end{aligned}$$

*Faculty advisor, Dept. of Computer Science

Proceedings of the 2011 Midstates Conference on Undergraduate Research in Computer Science and Mathematics

Note that the combined probabilities of each rule for *VerbPhrase* add up to a total probability of 1; that is, the CFG ideally represents every possible combination of constituents with some probability according to its frequency. The statistical element of a PCFG allows for increased accuracy in parsing by lessening the likelihood of a parsing algorithm choosing a rule used infrequently in the grammar.

The Cocke-Kasami-Younger (CKY) algorithm is used to create a parse tree describing the syntax of a sentence. The algorithm uses a grammar in Chomsky Normal Form (CNF), a form in which each rewrite rule must expand either to a single terminal node or exactly two non-terminal nodes. The algorithm is a bottom-up parsing method which uses the technique of dynamic programming. In the probabilistic version of the CKY algorithm to be used for a PCFG, each possible parse tree is given a score based on the probabilities encoded in the PCFG, and the highest-scoring tree is returned.



A completed parse, using the Penn Treebank tag set, on the sentence "This is an example of a parse tree."

1.2 Training a PCFG: Treebanks

In order to use a PCFG with the CKY algorithm to parse some piece of text, the PCFG must first be trained with a corpus; that is, the probabilities assigned to each rule must be estimated for use by the parser. This is usually done by reading in a corpus of pre-constructed trees parsed by annotators with linguistic expertise; from there, probabilities can be estimated by counting the number of time each rule appears within the corpus. This corpus is generally referred to as a treebank, the most common example of which is the Penn Treebank [5], the current standard for natural language processing systems. Other treebanks available include the Prague English Dependency Treebank [2], LinGo Redwoods [6], and the Penn Chinese Treebank [9].

Although the Penn Treebank tends to be used as a standard, it is not without its limitations. In particular, the treebank is trained on a corpus of sentences mostly from *Wall Street Journal* articles. The type of language recognized most accurately by a parser trained on the Penn Treebank, therefore, is that which reflects the same journalistic style. The treebank also uses a fixed tagset for parts of speech, potentially leading to problems for treebank users wishing to annotate trees using a different set of annotations. Finally, the treebank is limited by its size; users do not have the option to expand upon it to better suit their needs.

There is therefore a need for additional parsed data to use in natural language applications, particularly in corpora of other genres, such as literature or poetry.

2. RESEARCH OBJECTIVES

We set out to develop an interactive software tool which can be used by linguists and students to identify and save correct parse trees for sentences, with the ultimate goal of building a custom treebank. Parsing by hand is a time-consuming process, and automatic parsing often is unable to find the best parse without user input. An interactive parser allows for creation of correct parse trees while minimizing effort in correcting parses on the part of the user [7]. The parser proposes a parse, and then the user can make a single change after which the parser re-parses under the given constraints. This process is repeated until a satisfactory parse is found.

A number of tools have been built to perform this task; however, with each tool a new set of limitations arises, complicating the process of building a treebank.

The TreeBanker application [1], designed for supervised training of a system, asks the user clarifying questions about the given parsed training data with the intention of gaining extra information about the trees. While the application is de-

signed for users without linguistic expertise, we found that its interface was unintuitive; the visual representation of the parse tree was difficult to decipher. In particular, the representation showed the phrase structures, but not the underlying part of speech tags for each individual word, making it difficult to determine the exact precision of the parse.

Although the LinGO Redwoods treebank application [6] provides an intuitive graphical interface, it is limited to creating trees using a head-driven phrase structure grammar (HPSG), limiting the number of applications that could use it, as many systems are structured with the assumption of a PCFG structure.

The IPP-Ann tool [7] provides the user with an excellent graphical representation of parse trees, including a manual editing feature. However, trees in IPP-Ann can only be modified in a top-down, left-right manner; if the user attempts to edit the tree in the wrong order, the parser overrides previous modifications. Each edit to the tree, likewise, is made by reparsing the tree in the grammar used by the tool; therefore, if a valid tree is, for any reason, not accepted by the grammar, the tool cannot properly construct that tree.

Given the limitations of the available systems for treebank production, our task was to implement a better application that would provide a linguist or native speaker the proper tools for editing trees, whether through constraints on the parse, retagging, or manual modifications to the tree and its structure. With this tool, a human annotator could create and store a set of parse trees into a treebank, which could then be used for training a PCFG.

We worked with the source code for Stanford University's CoreNLP suite [8], a set of tools for NLP including part of speech tagging, named entity recognition, parsing, and recognition of dependencies. Most of our work focused on the Stanford Parser itself, an implementation of a lexicalized PCFG parser in Java. The Stanford Parser tool, and an interactive GUI interface to it, became the framework for our own interactive parser.

3. METHODOLOGY

In developing our parsing application, we started with the Stanford Parser and a GUI interface to it, both of which are found in the CoreNLP package [8]. The GUI program, written by Huy Nguyen, accepts sentences from the user, calls on the Stanford Parser to parse them, and then displays the resulting parse trees. As with any automatic parser, the Stanford Parser sometimes makes mistakes, generating an incorrect parse tree. We extended the GUI program to allow the user to find the correct parse tree by manipulating the parse tree in several ways. Basic features of the program include:

- Users start by loading in a grammar model (such as a PCFG) and an input file containing sentences that they wish to parse. Users may optionally enter the name of an output file in which to save parse trees in Penn Treebank form. If no input file is loaded in, the user may manually enter sentences to be parsed.
- Users may choose to enter tokenized or untokenized text; if the given text is untokenized, our parser automatically tokenizes it.
- Sentences within the input are automatically selected upon clicking or by scrolling through the sentences one at a time. Phrases within a sentence can also be highlighted and used to add constraints to the parse, described in section 3.1.
- Parsing a sentence displays an graphical, interactive parse tree from which the user may modify the tree through part-of-speech tags, phrasal constraints, or manual editing.

3.1 Constraints

We modified the Stanford Parser to parse under a set of constraints. When constraints are added, the program calls the Stanford Parser to parse the tree, subject to the given constraints. When a tree satisfying the constraints has been found by the parser, the new tree is displayed.

Part-of-speech tags can be edited and enforced by the user. For example, consider the sentence "I made her duck." How the sentence is parsed depends on whether the word "duck" is interpreted as a verb or a noun. Using our program, the user could specify a constraint on the word "duck" so that it must be tagged as a verb, or the user could choose to require that it be tagged as a noun. Either constraint would force the parser to produce a parse tree consistent with the user's choice of tag.

Negative tag constraints are also supported. The

user may specify that the tag for "duck" may *not* be "verb." The parse can then choose any tag other than "verb." In this way, the user can rule out certain trees, but still utilize the power of the parser.

The parsing algorithm recognizes several forms of phrasal constraints, allowing the user to specify how a certain phrase within the sentence should (or should not) be labeled within the parse tree. In particular, the user may indicate that:

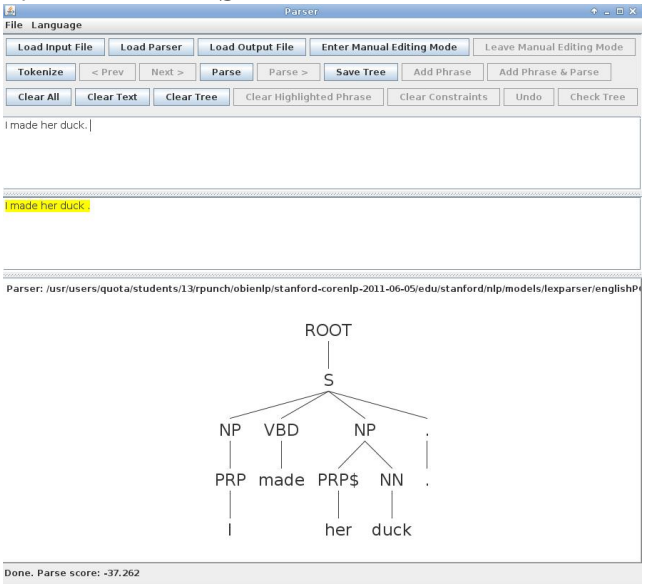
- A sequence of words should be treated by the parser as a phrasal unit. (e.g. a verb phrase or a prepositional phrase)
- A sequence of words should *not* be treated as a phrasal unit.
- A sequence of words should be treated as a particular type of phrase.
- A sequence of words should *not* be treated as a particular type of phrase.

For example, consider the sentence "The women talked about the dogs on the beach." The user may decide that "the dogs on the beach" is a single phrase, and this can be enforced by using constraints on the parse (effectively ensuring that the topic of these women's conversation is "the dogs on the beach" rather than that the conversation took place on the beach). Constraints may be added through the text interface or via the interactive tree.

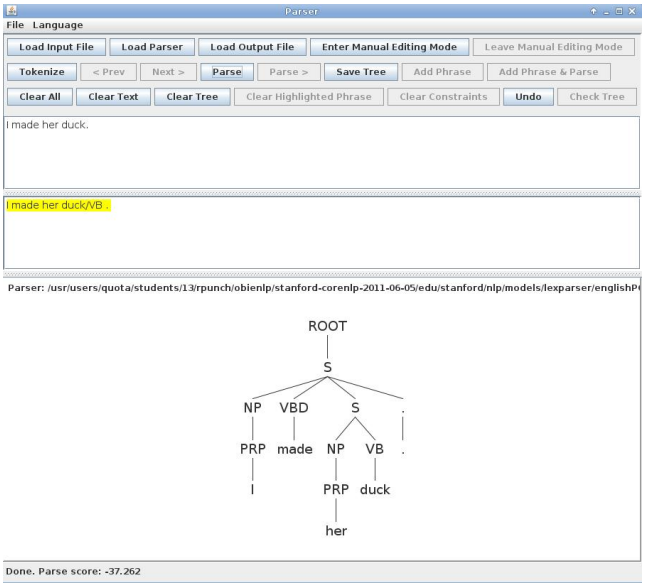
3.2 Manual Mode

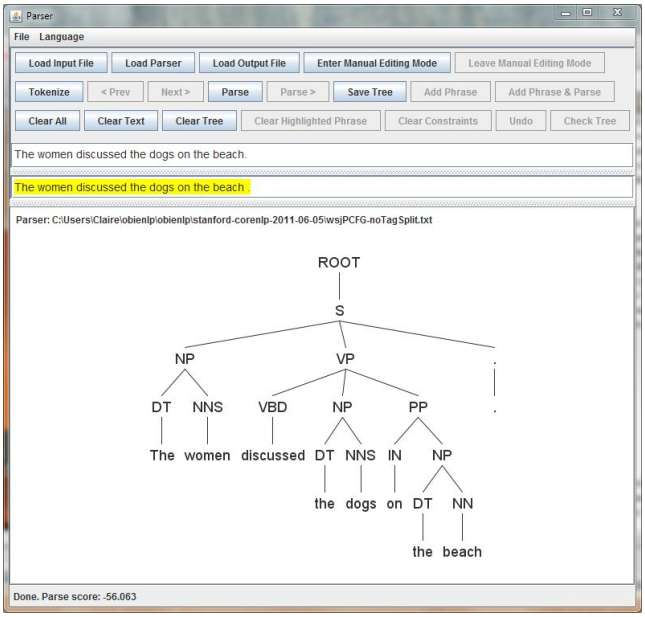
If the parser is unable to create a correct tree using the loaded grammar, the user then has the option to enter manual editing mode to edit the tree without calling the parser. This is useful for instances in which the grammar may not accept a tree that the user considers to be the correct tree. Trees produced manually may be checked for validity in the loaded grammar. A list of valid labels and/or tags for a specific node is available for use in editing the tree.

4. EXAMPLES

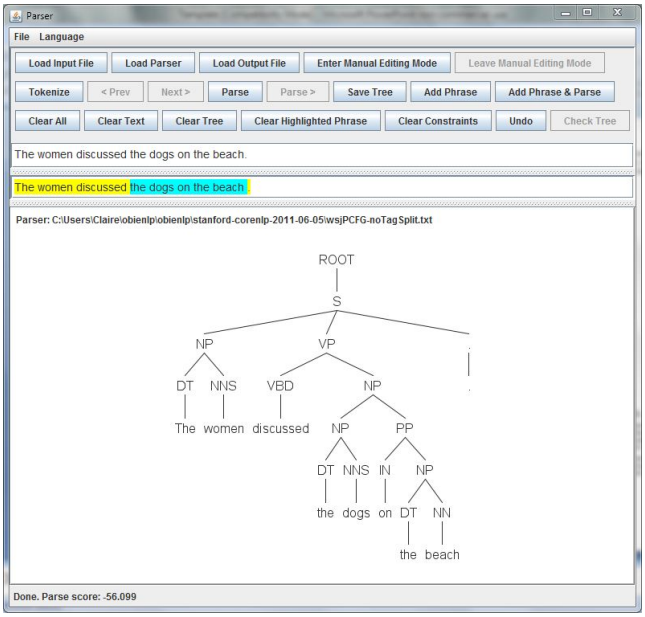


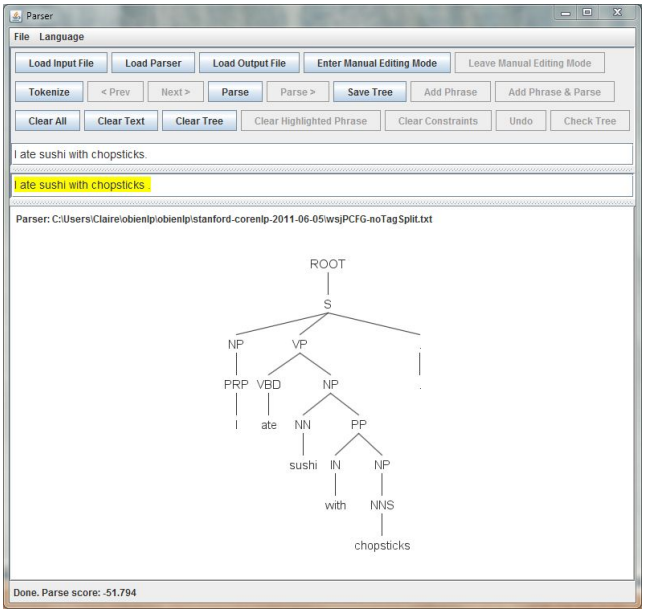
Above, an unedited parse tree for the sentence "I made her duck." With this parse tree, the sentence indicates that the speaker ("I") cooked waterfowl belonging to her. Below, a parse of the same sentence modified by selecting the tag for "duck" on the parse tree and changing the tag from NN to VB, specifying the constraint that the word "duck" should be a verb (VB). With this modification to the parse tree, the meaning of the sentence changes to "I caused her to quickly crouch."



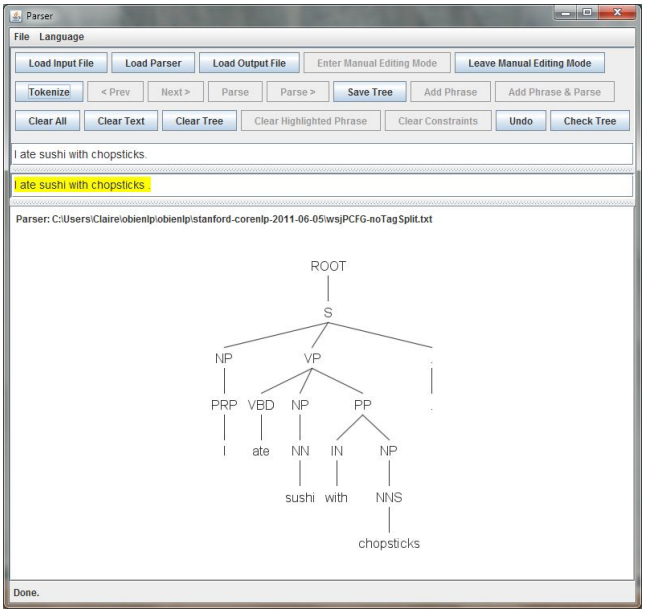


Above, an unedited parse tree for the sentence "The women discussed the dogs on the beach." With this parse tree, the sentence indicates that the women were on the beach while discussing the dogs, which could have been located elsewhere. Below, a parse of the same sentence modified with phrasal constraints by highlighting the phrase "the dogs on the beach" in the text section and clicking on the "add phrase" button. This specifies that "the dogs on the beach" should be treated as a phrasal constraint within the parse. With this constraint, the meaning of the sentence shifts to to the women discussing the dogs, which are located on the beach.





Above, an unedited parse tree for the sentence "I ate sushi with chopsticks." With this parse tree, the meaning of the sentence is nonsensical: the speaker ("I") ate sushi with chopsticks, where chopsticks are included in the meal. This sentence can be corrected in manual mode without parsing, specifically by dragging the prepositional phrase (PP) marker associated with the phrase "with chopsticks" from below the noun phrase (NP) "sushi with chopsticks" to VP, the parent verb phrase. This shifts the meaning to the speaker ("I") eating sushi with the help of chopsticks.



5. REFERENCES

[1] Carter, David. 1997. *The TreeBanker: a Tool for Supervised Training of a Parsed Corpora.* Proceedings of the ACL Workshop on Computational Environments for Grammar Development and Linguistic Engineering, Madrid.

[2] Cinková, Silvie, et. al. *Tectogrammatical Annotation of the Wall Street Journal.* Prague Bulletin of Mathematical Linguistics, 2009, 92.

[3] Jurafsky, Dan, and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2009.

[4] Klein, Dan and Christopher D. Manning. 2003. *Accurate Unlexicalized Parsing.* Proceedings of the 41st Meeting of the Association for Computational Linguistics, pp. 423-430.

[5] Marcus, Mitchell P., Beatrice Santorini, and Mary Ann Marcinkiewicz. *Building a large annotated corpus of English: the Penn Treebank.* Computational Linguistics, vol. 19, 1993.

[6] Oepen, Stephen, et al. 2002. *LinGO Redwoods: a Rich and Dynamic Treebank for HPSG.* Proceedings of the Workshop on Parseval and Beyond and the 3rd International Conference on Language Resources and Evaluation (LREC '02). Las Palmas, Spain.

[7] Sánchez-Sáez, Ricardo and Joan-Andreu Sánchez and José-Miguel Benedí. *Interactive Predictive Parsing.* 2009. Proceedings of the 11th International Conference on Parsing Technologies, pp. 222-225.

[8] Stanford Natural Language Processing Group. *Stanford CoreNLP.* 2011. http://nlp.stanford.edu/software/corenlp.shtml.

[9] Xue, Nianwen, et. al. *The Penn Chinese TreeBank: Phrase Structure Annotation of a Large Corpus.* Natural Language Engineering, 11(2)207-238, 2005.