

PARSING ENGLISH TEXTS IN PARE

NATHAN GILBERT, ERIC WELBORN,
AND SCOTT THEDE

ABSTRACT. We have designed and implemented a bottom-up text parsing strategy for English and integrated it into the automatic text summarizer PARE, replacing the old link-grammar parser previously used. Constituency trees from our parser provide all part-of-speech linkages as input to several other code modules in PARE. The parser was written to be highly modular, which facilitates its easy integration into the rest of the PARE system. The parser uses rules which are written in Chomsky Normal Form, a specialization of a general context-free grammar. These improvements should allow PARE to be more efficient and produce more accurate summaries.

1. PARE BACKGROUND

Originally coined the Pruner And Redundancy Eliminator, PARE is an automatic English text summarizer developed using the Java programming language at DePauw University by Kreger, Tornheim, and Thede [1]. The underlying engine for PARE was completely redesigned by Johnson, Vlahov, and Thede [2] using a summarization technique modeled after Google's PageRank algorithm. PARE was further improved by integrating an anaphora resolution module (an *anaphora* is a linguistic object that refers to some other object in the discourse — it is typically a pronoun) created by Bates, Mtandwa, Wray, and Thede [3], which correctly resolves pronouns to their antecedents. PARE presents an essential Java GUI, providing the original text, anaphora resolution data, and the summary. Behind the scenes, the text is foundationally analyzed using its parts of speech,

Received by the editors September 8, 2005 and in revised form October 11, 2005.

which are identified using a part-of-speech tagger, and its syntactical and semantic relationships, which are determined by a text parsing algorithm. PARE uses this rich information, provided by the parser for each sentence in the text, to resolve anaphora; categorize lexemes, such as the differing forms of the English word *find* (*find*, *finds*, *found*, *finding*); and to calculate salience, the relative importance of each idea of the sentence within the whole of the text. These calculations provide a “word graph”, which contains a mapping of the most important ideas of the text and how they interact. For example, providing PARE with a short story will produce a summary of its main ideas.

Since PARE's inception, a link grammar parser written in the C programming language at Carnegie Mellon [4] had been used for the parsing portion of the program. This combination of programming languages within the PARE system created some inefficiency; the passing of data from the parser to the rest of PARE and back was implemented using file input and output rather than by passing memory addresses, which would have been significantly faster. Also, C and Java use different character encoding schemes, ASCII and Unicode, respectively, which requires a conversion when passing data between the two programs. Writing a parser and part-of-speech tagger in the Java programming language would more tightly integrate the various modules of PARE, creating a more efficient, cohesive, and user-understandable system.

2. PARSING BACKGROUND

Parsing is the process of converting text input into a data structure, thus defining its syntactical structure and semantic meaning based upon a given formal grammar. Typically, in computer science, parsing is used in computer language compilers. However, human languages, which are typically ambiguous, can be parsed with excellent accuracy using context-free parsers. This is usually done by marking the words of the sentence with their appropriate parts of speech using a part-of-speech tagger,

then determining the sentence’s grammatical structure based upon a given context-free grammar [5]. Parsing human language is important in natural language processing, because it allows a computer system to represent the language as meaningful data structures, capable of being categorized, compared, and analyzed further: to correct spelling; to simulate comprehension of language; or, as with PARE, to determine the main ideas of a body of text.

There are two primary types of parsing algorithms: top-down parsers, which begin with the largest component of the input and successively attempt to derive simpler components, per its grammar rules, which span the sentence; and bottom-up parsers, which begin with the smallest components of the grammar and successively attempt to derive more complex components until the component spans the entire sentence. We chose to use the bottom-up parsing approach.

A context-free grammar is the “brain” of a parser - the scope and accuracy of a parser’s production rules largely determine its accuracy. Our parser was designed to compare two adjacent components in a sentence in order to derive more complex components, which required its context-free grammar to be in Chomsky Normal Form (CNF) [6]. Although the parsing algorithm and grammar were developed independently, they exhibit striking similarities to the CYK parsing algorithm developed in 1965 by Cocke, Younger, and Kasami [7], with an order of magnitude of $O(n^3)$ and its use of CNF context-free grammars.

3. PARE’S NEW PARSING ALGORITHM

The new parser’s functionality can be broken into two distinct stages. The first stage is pre-parse string processing, followed by the second stage, which is the actual parsing of the strings. During the pre-parse string processing, the parser reads the entire original text into a single string. Various regular expressions are applied to this string to break it apart into individual sentence strings to be parsed.

The original link-grammar parser used by PARE had strict rules for the format of the input text; for example, each sentence had to be on a single line, and punctuation rules were ignored. To compensate, PARE performed all of this preprocessing before any strings were sent to the parser. This preprocessing caused redundancy within PARE, for, in

reality, PARE was forced to parse sentences just to have proper input to send to the parser.

Our new parser is much more dynamic, accepting a variety of input string formats without any kind of outside processing needed. By including all the string preprocessing in our parser module, the code flow in PARE is much more understandable and manageable.

In the string parsing stage, we decided to take a different approach from that of the original parser. Instead of focusing on the grammatical links between two adjacent words, the new parser creates the constituency tree of a sentence from a CNF grammar which we supply (which is easily modifiable), then extracts all grammatical links from this tree. The decision to use a grammar completely in CNF was due to its power and ease of programming. Any context-free grammar can be represented in CNF; therefore, our parser is as powerful as the previous parser, which used a parser-specific grammar, but is easier to work with. The parsing algorithm is shown in Listing 1.

In general terms, this algorithm takes a list of trees, in which each tree initially contains a single word from the sentence to be parsed and its corresponding part of speech. The parser uses a fully integrated, freely available part-of-speech tagger created at Stanford University [8]. Attempts are then made to forge new, more complex trees by combining subsets from the tree list. We first check to see if two trees are adjacent and that their concatenation provides a production that can be found in the set of grammar rules. If this is the case, the new tree formed by the concatenation is appended to the tree list if it is not already present, and the search continues until no new trees can be created or a complete parse tree is found. Therefore, this parser is non-deterministic in design, but stops after finding the first constituency tree, making it deterministic in its implementation.

After this algorithm was completely designed and implemented, we found in it a striking resemblance to the CYK algorithm for parsing context-free grammars. The similarities are due to the fact that both algorithms consider every possible consecutive subsequence of the sequence of words in a sentence, and both require their grammars to be in CNF. They differ in the manner in which these subsequences and

these sentence-spanning productions of the CFG are stored.

4. INTEGRATION INTO PARE

Integrating the new parser into PARE proved to be an arduous task. We had underestimated the extent to which the original parser's constituency tree syntax was hard-coded into PARE's anaphora resolution module. This produced numerous problems during that stage of the project and, unfortunately, prohibited any comparison testing due to time constraints between the previous version of PARE and the current version. This testing will be done in later work on the project.

Near the end of our project (late July, 2005), the Stanford NLP group released a probabilistic parser with link-grammar capabilities to the general public. The Stanford parser was considered for use with PARE earlier in the project, but was rejected due to its inability to create any link dependency output. However, their parser's latest release now includes such functionality, so we have decided to begin another version of PARE which will use their parser for testing and comparison purposes. This alternate version will be concluded in later work.

5. CONCLUSION

PARE's capabilities outweigh its imperfections. While it will benefit from object-oriented code reorganizations, PARE, with its original parser and part-of-speech tagger, provides sufficiently concise and relevant summaries. However, a close integration of a CNF parser written in Java and an accompanying part-of-speech tagger lends additional robustness to PARE. The new parser we have written functions adequately, correctly parsing over 70% of the sentences contained within a variety of texts, containing up to 3000 or more sentences on average, with a simple CNF rule set. While we created and implemented a bottom-up, deterministic CYK parsing algorithm in Java with an expected overall efficiency of $O(n^3)$, its efficiency is worsened by the current design of PARE. Therefore, a redesign of PARE will complete the more efficient, cohesive, and user-understandable system we sought to employ.

6. FUTURE WORK

PARE's future development possibilities are wide open. Once the anaphora resolution/constituency tree syntax problem is resolved, future users have the chance to choose between two completely different parsers, the Stanford probability-based parser and our CYK derivative parser. This could lead to work testing whether PARE's performance with a probabilistic parser is measurably different than its performance with a deterministic, CNF, grammar-based parser.

PARE should be redesigned to take advantage of new developments in the Java programming language. Autoboxing and generic programming functionality are new to the Java programming language, and allow the programmer to write more readable and efficient code. Autoboxing refers to Java's new ability to forgo wrapping primitive classes (int, float, etc) in Wrapper classes (Integer, Character, etc). No longer is it required to insert a primitive data type into a Wrapper class to be able to use it with Java's container classes. Generic programming greatly reduces the amount of casting calls one has to make to place and remove objects from Java's container classes. These two changes could greatly simplify certain areas of PARE's code base.

Other work should involve improving the communication between PARE's member classes in order to better take advantage of our parser's constituency trees and reduce the need for the large amount of computing resources required presently. All immutable array types should be replaced by data structures which can be dynamically allocated, for more efficient memory management. The anaphora resolution modules need to be better integrated into the core PARE system in order to achieve their fullest potential.

For our current parser, enlarging our CNF grammar could improve our per-text parsing success percentage, but could also lead to an increase in incorrect constituency tree production. More testing is needed to determine the optimum combination of production rules in order to increase the number of correct parses and reduce parsing of ungrammatical sentences.

7. ACKNOWLEDGEMENTS

We are very grateful to the faculty of DePauw University's Computer Science Department for inviting us to spend the summer conducting research. Special thanks to Dr. Scott Thede for his mentorship in this challenging and fun project and to all faculty for their hospitality and support. This work was supported by National Science Foundation Grant Number EIA-0242293.

REFERENCES

1. L. Kreger, R. Tornheim, and S. Thede. *PARE: A Pruning Device for Automated Multi-Document Text Summarization*. Presented at the Consortium for Computing Sciences in Colleges: Midwest conference, Marion, Indiana, 2002.
2. T. Johnson, S. Thede, A. Vlahov. *PARE: An Automatic Text Summarizer*. First Midstates Conference for Undergraduate Research in Computer Science and Mathematics, 2003.
3. M. Bates, S. Mtandwa, S. Thede, J. R. Wray. *Anaphora Resolution in PARE, An Automatic Text Summarizer*. Second Midstates Conference for Undergraduate Research in Computer Science and Mathematics, 2004.
4. Daniel Sleator and Davy Temperley. *Parsing English with a Link Grammar*. Carnegie Mellon University Computer Science technical report CMU-CS-91-196, 1991.
5. L. Thomas. *Beginning Syntax*, 1993.
6. G. E. Revesz. *Introduction to Formal Languages*, 1983.
7. T. Kasami. *An efficient recognition and syntax-analysis algorithm for context-free languages*. Scientific report AFCRL-65-758, 1965.
8. K. Toutanova, C. D. Manning. *Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger*. Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000), 2000.

```

1  Let tList be a list of trees  $1, \dots, t_n$ . Where  $n$  at this stage is the
   number of words in a sentence, and each tree consists only of a single
   word and its corresponding part of speech in the sentence.
2  Let  $R$  be a grammar that contains only non-terminal symbols  $R_1, \dots, R_2$ . (
   All terminals are already on the list  $t$ .)
3  Each node of a tree consists of the actual word(s), parts of speech, and
   the beginning and ending index in the sentence of this subset of the
   sentence. [At this point in the algorithm, each tree consists of a
   single word, its one part of speech, and beginning index = ending index
   .]
4  Let addednode be a Boolean value indicating whether or not a new tree has
   been added to tList.
5  set addednode := false
6  for each  $i$  to  $t_n$ :
7    set tmpStart := the starting index of  $i^{th}$  element in the list (or tList[ $i$ 
   ].startIndex.
8    for each  $j = 0$  to  $t_n$ :
9      if  $i = j$ :
10     continue [Combining a subset of a sentence to itself will never produce
   another useful subset.]
11     set tmpLast := tList[ $j$ ].endIndex
12     if t[ $i$ ].endIndex = tList[ $j$ ].startIndex-1:
13       for each production  $R_a \rightarrow R_b + R_c$ :
14         if  $R_b + R_c = (tList[i]+tList[j]).POS$ : [The concatenation [7] of the  $i^{th}$ 
   and  $j^{th}$  trees' parts of speech.]
15         Create a new tree  $T$  which has the concatenation of the  $i^{th}$  and
    $j^{th}$  trees' words with  $R_a$  as the new part-of-speech of the root node, and
   the  $i^{th}$  and  $j^{th}$  trees as children.
16         if  $T$  does not already belong to tList:
17           add  $T$  to tList.
18           set addednode = true
19     for each  $i = 0$  to  $t_n$ :
20       if t[ $i$ ].POS = 'S' and t[ $i$ ].word = originalText
21         set finalTree := t[ $i$ ]
22         return finalTree
23     if addednode = false: [Both loops completed without adding any new trees to
   t.]
24     return null — no tree to be found.
25     else:
26     start this again with the new entries in list tList.

```

LISTING 1

MOREHEAD STATE UNIVERSITY
E-mail address: ngilbert@acm.org

WINTHROP UNIVERSITY
E-mail address: welborne2@winthrop.edu

DEPAUW UNIVERSITY
E-mail address: sthede@depauw.edu