An intelligent editor for natural language processing of unrestricted text

1999

Demetrios George Glinos
dgglinos@gmail.com

Find similar works at: https://stars.library.ucf.edu/rtd

University of Central Florida Libraries http://library.ucf.edu

Part of the Computer Sciences Commons

STARS Citation

https://stars.library.ucf.edu/rtd/2134

This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
AN INTELLIGENT EDITOR FOR NATURAL LANGUAGE PROCESSING OF UNRESTRICTED TEXT

by

DEMETRIOS GEORGE GLINOS
B.S. Trinity College, 1973
J.D. Georgetown University Law Center, 1976

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in the School of Computer Science in the College of Arts and Sciences at the University of Central Florida Orlando, Florida

Summer Term
1999
ABSTRACT

The understanding of natural language by computational methods has been a
continuing and elusive problem in artificial intelligence. In recent years there has been a
renewed interest in natural language processing research. Much of this work has been on
empirical or corpus-based methods which use a data-driven approach to train systems on
large amounts of real language data. Using corpus-based methods, the performance of
part-of-speech (POS) taggers, which assign to the individual words of a sentence their
appropriate part-of-speech category (e.g., noun, verb, proposition), now rivals human
performance levels, achieving accuracies exceeding 95%. Such taggers have proved
useful as preprocessors for such tasks as parsing, speech synthesis, and information
retrieval.

Parsing remains, however, a difficult problem, even with the benefit of POS
tagging. Moreover, as sentence length increases, there is a corresponding combinatorial
explosion of alternative possible parses. Consider the following sentence from a New
York Times online article:

After Salinas was arrested for murder in 1995 and lawyers for the bank had
begun monitoring his accounts, his personal banker in New York quietly
advised Salinas' wife to move the money elsewhere, apparently without the
consent of the legal department.
© 1999 Demetrios George Glinos
ABSTRACT

The understanding of natural language by computational methods has been a continuing and elusive problem in artificial intelligence. In recent years there has been a resurgence in natural language processing research. Much of this work has been on empirical or corpus-based methods which use a data-driven approach to train systems on large amounts of real language data. Using corpus-based methods, the performance of part-of-speech (POS) taggers, which assign to the individual words of a sentence their appropriate part of speech category (e.g., noun, verb, preposition), now rivals human performance levels, achieving accuracies exceeding 95%. Such taggers have proved useful as preprocessors for such tasks as parsing, speech synthesis, and information retrieval.

Parsing remains, however, a difficult problem, even with the benefit of POS tagging. Moreover, as sentence length increases, there is a corresponding combinatorial explosion of alternative possible parses. Consider the following sentence from a New York Times online article:

After Salinas was arrested for murder in 1995 and lawyers for the bank had begun monitoring his accounts, his personal banker in New York quietly advised Salinas' wife to move the money elsewhere, apparently without the consent of the legal department.
To facilitate parsing and other tasks, we would like to decompose this sentence into the following three shorter sentences which, taken together, convey the same meaning as the original:

1. Salinas was arrested for murder in 1995.
2. Lawyers for the bank had begun monitoring his accounts.
3. His personal banker in New York quietly advised Salinas' wife to move the money elsewhere, apparently without the consent of the legal department.

This study investigates the development of heuristics for decomposing such long sentences into sets of shorter sentences without affecting the meaning of the original sentences. Without parsing or semantic analysis, heuristic rules were developed based on: (1) the output of a POS tagger (Brill's tagger); (2) the punctuation contained in the input sentences; and (3) the words themselves. The heuristic algorithms were implemented in an intelligent editor program which first augmented the POS tags and assigned tags to punctuation, and then tested the rules against a corpus of 25 New York Times online articles containing approximately 1,200 sentences and over 32,000 words, with good results.

Recommendations are made for improving the algorithms and for continuing this line of research.
This work is dedicated to my wife Kathleen and our son George whose love and support, and particularly their good humor, sustained me throughout this experience.
ACKNOWLEDGMENTS

I wish to express my thanks and appreciation to my thesis advisor, Dr. Fernando Gomez, for his patience and firm guidance in the direction of this research, and to Dr. Sheau-Dong Lang and Dr. J. Michael Moshell for their encouragement in this endeavor and for serving on the committee and reviewing this thesis.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>xi</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>xii</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 The Natural Language Research Context</td>
<td>1</td>
</tr>
<tr>
<td>1.2 The Sentence Splitting Problem</td>
<td>3</td>
</tr>
<tr>
<td>2. DEVELOPMENT APPROACH</td>
<td>6</td>
</tr>
<tr>
<td>2.1 Design Approach</td>
<td>6</td>
</tr>
<tr>
<td>2.1.1 Scope</td>
<td>6</td>
</tr>
<tr>
<td>2.1.2 Software Architecture</td>
<td>7</td>
</tr>
<tr>
<td>2.1.3 Heuristic Rules</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Literature Search</td>
<td>12</td>
</tr>
<tr>
<td>3. THE BRILL TAGRER</td>
<td>15</td>
</tr>
<tr>
<td>3.1 Tagger Description</td>
<td>15</td>
</tr>
<tr>
<td>3.2 Applicability to the Intelligent Editor Task</td>
<td>17</td>
</tr>
<tr>
<td>4. HEURISTIC RULE SET</td>
<td>20</td>
</tr>
<tr>
<td>4.1 Post-Tagging Rules</td>
<td>20</td>
</tr>
<tr>
<td>4.1.1 Decimal Numbers and Time of Day</td>
<td>20</td>
</tr>
</tbody>
</table>

vii
4.1.2 Question and Exclamation Marks in Direct Quotes 21
4.1.3 Multiple Initial Identifiers 22
4.1.4 Idiomatic Expressions and Collocations 23
4.1.5 Non-Sentence Splitting Coordinating Conjunctions 23
4.1.6 Coordination Commas 24
4.1.7 Direct Quotes 25
4.1.8 Proper Noun Appositions 26
4.1.9 Noun Phrase Appositions 27
4.1.10 Commas Introducing Adverbials 28
4.1.11 Indirect Quotes 28
4.1.12 "Wh-" Relative Clauses 29
4.1.13 Prepositions and Subordinate Conjunctions 30
4.1.14 Unknown Commas 31
4.1.15 Parenthetical Expressions 31
4.1.16 Expressions Separated by Dashes 31

4.2 Sentence Splitting Rules 32
4.2.1 Comma-"Which" Splits 32
4.2.2 Comma-Preposition-"Which" Splits 34
4.2.3 Comma-"With"-Gerund Splits 36
4.2.4 Subordinate Clauses 38
4.2.5 Comma-"But" Splits 40
APPENDICES

A. "Salinas" Article Text, Tag, and Output Files 77
B. "Italy" Article Text, Tag, and Output Files 95
C. "Flag" Article Text, Tag, and Output Files 111
D. IE and Brill Tagger Makefiles for the PC 131
E. Intelligent Editor Source Code 136

LIST OF REFERENCES 219
LIST OF TABLES

Table 1: The Penn Treebank Tagset 18
Table 2: Intelligent Editor Augmented Tagset 53
Table 3: Intelligent Editor Tag Prefixes and Modified Brill Tags 54
Table 4: Intelligent Editor Sentence Split Tags 55
Table 5: Test Corpus of New York Times Online Articles 65
Table 6: Test Corpus Composition 66
Table 7: Sentence Length Reduction 68
Table 8: Sentence Split Precision 70
Table 9: Sentence Split Effectiveness 72
LIST OF FIGURES

Figure 1: Intelligent Editor Functional Components 58
Figure 2: Intelligent Editor Data Flow 60
1. INTRODUCTION

1.1 The Natural Language Research Context

The understanding of natural language by computational methods has been a continuing and elusive problem in artificial intelligence since its inception. This complex field includes such diverse tasks as speech recognition and spoken-language analysis, syntactic analysis, semantic analysis, discourse analysis and information extraction, and machine translation [3]. Speech recognition involves mapping a continuous speech signal into a sequence of recognizable words. Syntactic analysis begins with the words and includes assigning parts of speech to those words, determining the grammatical structure of a sentence by grouping the words into noun phrases, verb phrases, prepositional phrases and other components, and by associating such phrases one with the other (called "attachment"). Semantic analysis involves extracting meaning from a sentence in some form of knowledge representation such as a logical expression or a semantic network. Finally, machine translation involves translating text from one natural language, such as English, to another natural language, such as Japanese.

Each of these sub-fields has received renewed attention in recent years with the advent of more powerful workstations capable of processing large amounts of data.
relatively inexpensively, and with the availability of large annotated corpora, such as the Penn tree bank corpus, which contains syntactic parses for about 50,000 sentences from the Wall Street Journal [3], and WordNet, a lexical database containing 121,962 unique words organized into 99,642 "synsets" of words with similar meanings [18]. Such resources made it possible to investigate statistical methods for solving open issues in the field and to conduct research beyond the reach of restricted language domains.

Using corpus-based methods, a variety of part-of-speech (POS) taggers have been developed for assigning the appropriate part of speech category (e.g., noun, verb, preposition) to the individual words of a sentence. Employing a variety of techniques, including stochastic methods involving Hidden Markov Models (HMMs) and trainable rule-based methods, POS taggers now rival human performance levels, achieving accuracies exceeding 95% [3]. Performance at these levels permits using POS taggers as preprocessing steps to support downstream syntactic analysis, speech synthesis, and information retrieval.

Empirical, corpus-based methods hold out the promise of tackling some of the larger issues in current artificial intelligence research, including large-scale automated knowledge acquisition from text. They also offer the means for expanding the coverage, robustness, and extensibility of a natural language processing through using ever larger corpora incorporating more and more phenomena of interest for developing the processing components.
1.2 The Sentence Splitting Problem

Syntactic analysis, or parsing, continues to be difficult natural language processing (NLP) problem, even with the benefit of POS tagging. In discussing statistical parsing techniques, Charniak [5] estimates that for a sentence containing 23 words and punctuation, which is the average sentence length for the Wall Street Journal articles in the Penn tree bank corpus, the figure of one million parses for such a sentence would be "conservative."

Consider also the sentence: "Mary saw the man on the hill with the telescope." The question arises, who has the telescope? Is it Mary, the man, or the hill? This is the problem of "attachment" of prepositional phrases, a notoriously difficult sub-task. Brill [3] estimates that a sentence ending in N prepositional phrases such as "on the hill" or "with a telescope" has at least $2^N$ syntactic analyses.

Given the above, it is evident that as sentence length increases, there is a corresponding combinatorial explosion of alternative possible parses. And while there does not appear to be a straightforward means of splitting the sample sentence above into shorter sentences, it is also clear that there are many sentences that can be split without altering the meaning of the original sentences. As a simple example, consider: "Peter ate an apple, and Mary ate a carrot." If this sentence were split into the two sentences, "Peter
ate an apple," and "Mary ate a carrot," the two shorter sentences together clearly convey the meaning of the original sentence.

This, then, is the sentence splitting problem: how to identify and split off sentences that can be separated without affecting the original meaning. As a more realistic example, consider the following sentence containing 41 words and 4 punctuation marks, from a New York Times online article that is in the IE test corpus:

After Salinas was arrested for murder in 1995 and lawyers for the bank had begun monitoring his accounts, his personal banker in New York quietly advised Salinas' wife to move the money elsewhere, apparently without the consent of the legal department.

To facilitate parsing and other tasks, it is desirable to decompose this sentence into the following three shorter sentences which, together, convey the same meaning as the original:

1. Salinas was arrested for murder in 1995.
2. Lawyers for the bank had begun monitoring his accounts.
3. His personal banker in New York quietly advised Salinas' wife to move the money elsewhere, apparently without the consent of the legal department.

Similarly, given this 26-token (counting all words and punctuation separately) sentence from another New York Times online article in the IE test corpus:

The incident at Caserta, which occurred shortly after the new culture minister, Giovanna Melandri, was appointed, lighted a fire in Rome.
it is desirable to split it into:

1. The incident at Caserta lighted a fire in Rome.

2. [prior subject] occurred shortly after the new culture minister, Giovanna Melandri, was appointed.

which together convey the original meaning, where "[prior subject]" refers to the subject of the immediately preceding sentence. Interestingly, this sentence illustrates more of a "carve out" rather than a "split off", since the first sentence continues after the split off sentence ends. Nevertheless, for convenience, no distinction in terminology is made with respect this feature, and simply the term "split" is used in both cases.
2. DEVELOPMENT APPROACH

2.1 Design Approach

2.1.1 Scope

The design goal of this research was to implement an Intelligent Editor (IE) that is capable of solving the sentence splitting problem in the broadest possible context, that is, for unrestricted text. Unrestricted text is unedited "real world" text for which the subject matter is not restricted to any particular domain, which the NLP system could otherwise use to resolve ambiguities. It is not difficult to see how the word "fly" in an airline travel information domain most likely refers to the verb, while in a more general encyclopedic text domain, the word could also refer to the insect, a fisherman's lure, a tent flap, or other nouns or verbs.

As a close approximation to fully unrestricted text, the database of New York Times online articles was selected as the corpus for evaluation of IE performance. It possesses generality in the form of articles on numerous subjects in a number of areas of interest, from international news to domestic political and environmental subject areas. It is also characterized by a high average sentence length. Indeed, as more fully discussed in Section 6.2 of this thesis, the average sentence length of approximately 27 tokens is more than for the Wall Street Journal articles in the Penn tree bank corpus [5].
2.1.2 Software Architecture

The IE design approach also recognized that development of the IE must be an empirical task, as no formalism to describe the meaning of a sentence in general and the effect of making sentence splits had been discovered during the course of this research. As a result, a key design feature was one of modularity so that to the extent possible each sentence splitting heuristic rule was represented by a unique software module. This permitted an incremental development cycle in which new heuristic rules could be added to the existing rule set without changing the existing rules, and also in which greater detail could be added to an existing rule without affecting any of the others.

In this manner, the IE was developed as a testbed architecture, one which once basic input and output functionality were provided, furnished a framework for adding and testing heuristic rules incrementally.

2.1.3 Heuristic Rules

The IE design approach with respect to sentence splitting heuristic rules consists of four parts. First, concerning the data upon which the rules would operate, this research proceeded on the assumption that more information was better than less for rule development, rather than restricting consideration to one lexical knowledge source. On
this basis, the following readily available knowledge sources were selected for heuristic rule development: (1) POS tagger output, (2) the punctuation already embedded in the sentences, and (3) the words themselves. For the POS tagger, the Brill tagger [2] was chosen. It is discussed in detail in Chapter 3 of this thesis. Both the punctuation and the words are contained in the input data files and are therefore always available.

The second aspect of the heuristic rule design approach was to recognize that attempting to capture all possible types of sentence splits would require a far more thorough understanding of linguistics than could be learned during the course of this research. Accordingly, only the following situations (with sample inputs from the IE test corpus, and desired outputs) were selected for heuristic rule development:

1. **Sentence coordination:**
   
   **Input:** The weirs could then be removed, the compartments would disappear and natural water flow would resume.
   
   **Output:** The weirs would then be removed. The compartments would disappear. Natural water flow would resume.

2. **Verb phrase coordination:**
   
   **Input:** Maria Torres Garcia turned on a faucet in her home one recent morning and watched a little trickle run into the bucket she uses to collect water during the few hours a day when water still flows into homes here.
   
   **Output:** Maria Torres Garcia turned on a faucet in her home one recent morning.
[Subj] watched a little trickle run into the bucket she uses to collect water during the few hours a day when water still flows into homes here.

3. **Subordinate clause:**

   **Input:** Although Salinas had never been formally accused of wrongdoing, rumors of possible corruption were widespread in Mexican financial circles.

   **Output:** [Although] Salinas had never been formally accused of wrongdoing.

   Rumors of possible corruption were widespread in Mexican financial circles.

4. **Comma-"but" + clause:**

   **Input:** Neighborhood residents surrounded the plant recently in an attempt to close it down, but dispersed after police arrived.

   **Output:** Neighborhood residents surrounded the plant recently in an attempt to close it down.

   [Subj] dispersed after police arrived.

5. **Comma-"with" + gerund:**

   **Input:** The votes split largely along party lines, with the Republican majority electing the voluntary measure and the Democratic minority preferring the mandatory one.

   **Output:** The votes split largely along party lines.

   The Republican majority elected the voluntary measure and the Democratic minority preferred the mandatory one.
6. **Comma-"which" + clause:**

**Input:** The commander of the school, which was established here in 1948, said that as a soldier, he would obey orders to retreat from the palace.

**Output:** The commander of the school said that as a soldier, he would obey orders to retreat from the palace.

[prior] was established here in 1948.

7. **Comma-prep-"which" + clause:**

**Input:** Some scholars here argue that from ancient times until perhaps 150 years ago, virtually all Japanese learned to walk in a special style called the namba, in which the right arm and leg swing forward at the same time, and then the left arm and leg swing forward.

**Output:** Some scholars here argue that from ancient times until perhaps 150 years ago, virtually all Japanese learned to walk in a special style called the namba.

The right arm and leg swing forward at the same time, and then the left arm and leg swing forward in [ which] [prior].

8. **Clauses separated by dashes:**

**Input:** Many thousands of people have been uprooted in Kosovo -- where ethnic Albanians make up 90 percent of the 2 million people -- since Milosevic began his crackdown on separatists in February.

**Output:** Many thousands of people have been uprooted in Kosovo since Milosevic began his crackdown on separatists in February.

Ethnic Albanians make up 90 percent of the 2 million people.
9. **Parenthesized clauses:**

**Input:** The palace has 1,200 rooms (Versailles has 700) and is surrounded by a 250-acre park that includes a 256-foot cascade.

**Output:** The palace has 1,000 rooms and is surrounded by a 50-acre park that includes a 256-foot cascade. Versailles has 700.

where "gerund" refers to the gerund verb form (ending in "-ing"), "prep" refers to a preposition, "[Subj]" refers to the subject of the previous sentence, and "[prior]" can refer to either the subject or direct object of the previous sentence.

The third design choice concerning the heuristic algorithms involved the exclusion of certain situations from consideration for splitting. These situations were: (a) clauses containing the word "that" and (b) clauses within direct quotes. It was felt that both of these situations involve sentence structures that exceed the level of detail possible in an effort of this scope, and that the latter situation in particular requires the analysis of sentences beyond the one under consideration, which could be expected to increase program complexity significantly.

Finally, the fourth IE design choice for heuristic rules concerns the relative emphasis of components of split heuristics. Specifically, the choice was made to concentrate the primary effort of this research into the sub-task of identifying and effecting sentence splitting at the appropriate places in the input data stream. While some effort was made to handle verb form agreement and the propagation of modals, infinitives, and
complex verb forms, these aspects were considered of secondary importance at this stage of the research into this field.

While we do not possess any hard data concerning the what portion of all available splits is accounted for by the set of splits (with exclusions) that were selected for study, we felt that our set was sufficiently comprehensive to make a significant difference in sentence length for real-world applications, and so to serve as the basis for a proof of concept for the utility of sentence splitting for NLP.

2.2 Literature Search

A search of the published literature in the computational linguistics, natural language processing, and artificial intelligence fields did not disclose any references to prior work involving meaning invariant sentence splitting transformations, nor indeed any references to sentence splitting of any kind. Nonetheless, this research does have antecedents in the published literature.

Wilks [16] describes a multiple knowledge source approach, as done here, for word sense disambiguation, which is the NLP task of assigning to a given word in context the correct "sense" or meaning. For example, comparing "John saw the bird" and "John used the saw", it is not difficult for a human to assign the verb associated with vision to the first instance of "saw" and the cutting tool noun for the second. In this case, the distinction is made simple by the different parts of speech for the word (here, "saw"), but
more subtle distinctions must be made where, for instance, a polysemous word possesses numerous senses for the same part of speech, such as the many noun senses of the word "fly".

Wilks' system attempts disambiguation of only the "content" words of a given sentence, which the authors define as only nouns, verbs, adjectives, and adverbs, but not prepositions, conjunctions or other parts of speech. The system bases its sense determination on the output of a POS tagger (actually, the Brill tagger), dictionary definitions, thesaural hierarchies, and selectional restrictions. These multiple knowledge sources are combined using a "trained decision list" approach, in which the system automatically generates rules upon presentation of the outputs of each of the knowledge sources for a given word and the "true" sense value for the word.

That POS tagger output can be a powerful first step towards subsequent language understanding was investigated previously by Wilks [15], who reported achieving 92% correct sense tagging of content words by simply choosing the most frequently occurring sense, as determined by the Longman Dictionary of Contemporary English (LDOCE), after using the part of speech tag set by the Brill tagger to exclude senses which do not involve that part of speech. These results were obtained using a corpus of 1,700 words from five articles from the Wall Street Journal.

While no references to sentence length were disclosed by the literature search, Cozens [6] used word (not sentence) length as the sole data source in training a simplistic POS tagger. Using word length data alone, the tagger achieved a success rate of
approximately 33%, which the authors felt would help boost performance of a modular POS tagger by about 5% over current levels, helping them to achieve near-human performance levels.
3. THE BRILL Tagger

3.1 Tagger Description

The POS tagger described by Brill [2] is a corpus-based, trainable rule-based tagger. Unlike stochastic taggers, which encode the knowledge obtained from their training corpus in enormous bigram and/or trigram probability distribution data structures containing the probabilities of words having particular tags given the tag or tags of the previous or subsequent one or two words, a rule-based tagger extracts a relatively small number of rules, on the order of two or three hundred, from the many thousands of examples in the training corpus [2].

The Brill tagger is trained through an automated process which he calls "transformation-based error-driven learning." Although the training itself is automated, an essential component is a manually tagged training corpus containing the known "truth", so that this process is essentially one of supervised learning.

The training process proceeds as follows: First, an "initial-state annotator" is used to assign tags to the input text, which is untagged. The initial-state annotator can range in complexity from simply assigning tags randomly to implementing a sophisticated manually developed tagging scheme. Both extremes can be accommodated by the tagger.
Once tagged initially, the tagged text is compared against the "truth" contained in the manually tagged training corpus. Transformation templates are then used to "learn" appropriate tag transformations. The templates are rules schema such as "Change tag 'a' to 'b' when the preceding word is tagged 'z'." For each such template, the system automatically applies every possible transformation. In the example above, it would try every possible combination of tag choices for tags 'a', 'b', and 'z'. The system does this for every transformation template at its disposal. In this manner, the system applies every possible transformation permitted by its transformation rule set.

The transformation rule that results in the greatest improvement in the performance of the tagger for the given input training text is selected as a rule that has been "learned." The process then repeats, with additional rules being learned until no rule improves performance above a given threshold. When completed, the transformation rules compose an ordered set of transformations that, when applied in order, execute a hill-climbing strategy for optimizing tagging.

The tagging process for non-training text proceeds in two stages: First, untagged text is passed through a "start state tagger", which establishes an initial POS tag for each word. The start state tagger refers to a list of tags, developed during training, one for each word in the training corpus. This list contains the parts of speech that the listed word can take, in decreasing order of likelihood. This list, called the "lexicon", was derived from the Penn Treebank tagging of the Wall Street Journal, which contains approximately 3 million entries and the Brown Corpus [4]. The start state tagger assigns the most likely
tag for the given word. Unknown words are tagged initially as singular or mass nouns, or if they are capitalized, as proper nouns. The learned transformation rules are then applied to refine the initial tags for such words.

As the second stage of the tagging process, contextual rules are applied to refine the initial tags. Contextual rules, which were learned during training in the same manner as transformation rules, contain entries such as "VBN un fhaspref 2 JJ x", which translates to "if a word has a prefix of 'un' and it is currently tagged as 'VBN' (Verb, past participle), then change the tag to 'JJ' (Adjective)." This rule would change the tag for the word "annotated" from past participle to adjective, which is the correct result where the word is, for example, part of the noun phrase "annotated text."

3.2 Applicability to the Intelligent Editor Task

Brill reports impressive performance for his tagger, achieving an accuracy of 97.2% on a 150,000 word test set using only 267 simple nonstochastic rules, exceeding the performance of a stochastic tagger that encoded contextual information in 10,000 contextual probabilities [2]. This level of performance is more than adequate for the IE task.

The Brill tagger is also made available for research use such as this study. It is furnished as a set of C-language program files, headers, and text files. It can be easily hosted on both UNIX and PC systems. This accessibility makes it ideally suited for an
empirical study, such as this one, where the programs must be run many times, both for software development and for test and evaluation.

The Brill tagger reports tags in the form prescribed by the Penn Treebank Project [12], which uses the tags listed and described in Table 1.

Table 1

The Penn Treebank Tagset

| 1. CC  | Coordinating conjunction |
| 2. CD  | Cardinal number         |
| 3. DT  | Determiner              |
| 4. EX  | Existential *there*     |
| 5. FW  | Foreign word            |
| 6. IN  | Preposition or subordinating conjunction |
| 7. JJ  | Adjective               |
| 8. JJR | Adjective, comparative  |
| 9. JJS | Adjective, superlative  |
| 10. LS | List item marker        |
| 11. MD | Modal                   |
| 12. NN | Noun, singular or mass  |
| 13. NNS | Noun, plural           |
| 14. NNP | Proper noun, singular  |
| 15. NNPS | Proper noun, plural  |
| 16. PDT | Predeterminer         |
| 17. POS | Possessive ending      |
| 18. PRP | Personal pronoun       |
| 19. PRP$ | Possessive pronoun  |
| 20. RB | Adverb                 |
| 21. RBR | Adverb, comparative    |
| 22. RBS | Adverb, superlative    |
| 23. RP | Particle               |
| 24. SYM | Symbol                |
| 25. TO | to                     |
| 26. UH | Interjection           |
| 27. VB | Verb, base form        |
| 28. VBD | Verb, past tense       |
| 29. VBG | Verb, gerund or present participle |
| 30. VBN | Verb, past participle  |
| 31. VBP | Verb, non-3rd person singular present |
| 32. VBZ | Verb, 3rd person singular present |
| 33. WDT | Wh-determiner         |
| 34. WP  | Wh-pronoun             |
| 35. WP$ | Possessive wh-pronoun  |
| 36. WRB | Wh-adverb              |

While the Penn Treebank tagset is comprehensive, it does leave certain tagging issues unresolved. Most critically, the "IN" tag is used both for prepositions and for subordinating conjunctions. This makes it necessary for the Intelligent Editor to determine
on its own in a particular instance whether a word tagged with "IN" is actually a subordinating conjunction, in which case the clause it introduces can be split off from the sentence, or merely a preposition, which would not call for a sentence split.

Similarly, this tagset uses the tag "VBG" both for present participles (e.g., "Peter is swimming") and for gerunds, in which the present participle functions as a noun (e.g., "Swimming is fun"). This ambiguity makes a difference in any rule that involves identifying noun and verb phrases.

The Brill tagger does not tag any punctuation marks. Nor does it group numbers or times of day (such as "20.35" and "6:15"), leaving the punctuation within to serve as a source of ambiguity in the larger sentential context. Nor does it associate an abbreviation with its following period (for example, "Corp."), and similarly for multiple initial identifiers such as "U.S.". These situations can introduce error by confusing such punctuation with the punctuation that marks the beginning or end of a clause or sentence.

Although these shortcomings can be expected to affect performance if not compensated for adequately, they are relatively minor in comparison with the wealth of information that the Brill tagger makes available for the development of sentence splitting heuristic rules.
4. HEURISTIC RULE SET

A number of post-tagging, sentence splitting, and utility function heuristic rules are employed by the IE. These rules operate on the tags produced by the Brill tagger, and on tags added or modified by the IE itself. The major heuristic rules are discussed in separate sections below. The complete set of IE augmented tags is presented in Section 4.4.

4.1 Post-Tagging Rules

In order to resolve the ambiguities introduced by ambiguous tags, untagged punctuation marks, and tags that could serve to confuse the sentence splitting rules, the IE applies a number of post-tagging rules prior to the application of any sentence splitting rule. These post-tagging rules are described in the following subsections.

4.1.1 Decimal Numbers and Time of Day

Cardinal numbers that include decimal points and/or commas are tagged in pieces by the Brill tagger, so that the number "12,635.86", for example, is tagged as follows: "12/CD,/, 635/CD,/. 86/CD". Similarly, time of day is also tagged piecemeal, for
example, "9:37" is tagged as "9/CD :/. 37/CD". In each of these cases, the punctuation marks could serve to confuse heuristic rules that search for the beginnings or ends of clauses or sentences, which can do so using such punctuation marks.

Accordingly, as the input file is scanned, a three-token buffer is maintained as a sliding window across the input. If the window contains two cardinal numbers (tag "CD") separated by a comma, period, or colon, then the three tokens are merged into a single token, which is then marked as a cardinal number. This permits subsequent merges for any other parts of the same number that are separated by such punctuation. Thus, numbers such as 12,635.86 and 9:37:54 will be constructed incrementally.

This rule also merges a period followed by a CD into one CD token. However, it does not merge a CD followed by a period, as this could be the end of a sentence. For all merged tokens, the punctuation marks are embedded within the tokens, which has the effect of eliminating such punctuation from consideration for sentence splitting or sentence termination.

4.1.2 Question and Exclamation Marks in Direct Quotes

The Brill tagger marks all question marks and exclamation marks as alternative species of periods: "?./" and "!./.". This could inadvertently be construed as the end of a sentence when it should not. For example, consider: ' "Look out!" shouted Mary.' This is clearly one sentence containing an embedded direct quote.
A simple rule is followed in such instances. Where a question or exclamation mark is followed by a quotation mark and the quotation mark is followed by a word that either begins with a lower case letter (as above) or is a proper noun (e.g., "Look out!" Mary shouted.), then the question or exclamation mark is tagged with the IE tag "pq" to distinguish it from periods that mark the ends of sentences.

4.1.3 Multiple Initial Identifiers

As with decimal numbers, multiple initial identifiers such as "U.S.", "a.k.a.", "B.C.", and "p.m." can confound heuristic rules searching for clause terminators if the punctuation marks are not recognized as being part of such identifiers.

Using a similar approach, the tagged input file is scanned using a four-token window. The presence of each multi-initial identifier is checked explicitly, and if found, the tokens are merged into one, with the tag appropriate to the circumstance. Thus, "U.S." is tagged as a proper noun ("NNP"), but "a.m." is tagged as a cardinal number ("CD") and "e.g." is tagged as a preposition ("IN"). The three-initial identifier "a.k.a." is constructed in two passes and is tagged as a coordinating conjunction that does not cause a sentence split ("zCC"). And again, the confusing effect of the punctuation marks is avoided by embedding such marks within the merged tokens.
4.1.4 Idiomatic Expressions and Collocations

Certain word sequences comprise idiomatic expressions, such as "on the one hand" and "part and parcel". Other word sequences are simply words that are ordinarily found with each other (called "collocations"), such as "and a half" and "compared with". In order to prevent such expressions and collocations from being split up or dropped in whole or in part from output, they are marked for non-interference by appending the prefix "i" to their tags. The IE maintains a list of such expressions and collocations, and the input file is scanned for the presence of any of them. If a match is found, the tags for all of the words included in the expression or collocation are marked with the "i" prefix.

4.1.5 Non-Sentence Splitting Coordinating Conjunctions

Coordinating conjunctions such as "and", "or", and "but" often appear in the context of coordinations that involve merely enumerating a list of items and do not, therefore, call for splitting the sentence. For example, consider the sentence: "Peter was quick, climbing the mountain higher and faster than anyone had done before." Left uncorrected, the "and" in this sentence could fool a heuristic rule that looks for coordinations and finds that in this example, there are noun and verb phrases both before and after the "and", and so would split the sentence at that word.
To handle such situations, the coordinating conjunctions ("CC" tags) in the sentence are checked. If they are both preceded and followed by tokens with identical tags, then the CC is marked with the prefix "z" to indicate that it does not support sentence splitting.

4.1.6 Coordination Commas

Coordination refers to combining two or more short simple sentences or verb phrases into a single more complex sentence. Separation among the components is ordinarily maintained either by commas or by coordinating conjunctions such as "and", "or", or "nor", as in "Peter ate an apple, and Mary ate a carrot," and "The old plane sputtered to life, rumbled down the runway, and took off."

The IE uses the following rule to label coordination commas: For a given input sentence, scan the sentence until find an otherwise unmarked comma, that is, one whose tag is ",". If any such comma is found, the entire remaining sentence is scanned for the presence of any one of the six coordinating conjunctions (labeled "CC" by the Brill tagger) that are on a list maintained by the IE: "and", "or", "nor", "but", "yet", and "plus". If one of the listed CCs is found, then, beginning with the original unmarked comma, the sentence is scanned only until the next comma, CC, semicolon, colon, or end of sentence, ignoring any intervening apposition commas. If a verb form other than a present participle is found within such clause, then the original unmarked comma is tagged
with the IE tag code "ccrd". The process then repeats, beginning with the next unmarked comma, if any.

4.1.7 Direct Quotes

It is important for the IE to identify words within direct quotations properly, since words and punctuation within direct quotations are not to be considered by sentence-splitting heuristics, as determined by the IE heuristic design approach.

Complexity is introduced in the logic for this rule because of the sentence spanning nature of direct quotations. For a direct quote that includes multiple sentences, the first such sentence usually contains the open quotation mark, intermediate sentences usually contain no quotation marks, and the last sentence in the quotation usually contains only the close quotation mark.

The basic rule for direct quotes is to maintain a semaphore variable which indicates whether the current token is within or outside of a direct quote. Using the semaphore, the open and close quotation marks are tagged accordingly, and all intervening tokens are tagged with the IE prefix tag "q", which effectively removes all tokens so marked from consideration by any of the sentence-splitting rules.

The complexity described above is managed by augmenting the rule to provide for inserting an end-quote at the end of a sentence if the semaphore indicates that the sentence end is within the direct quote, and by inserting an open quote at the beginning of the next
sentence if the quotation continues and is not closed out by an end-quote following the end of the previous sentence.

4.1.8 Proper Noun Appositions

Proper noun appositions that are not set off by commas, such as "President Bill Clinton," usually involve a title (here, "President") preceding a proper name. The Brill tagger, however, marks each of them with the same singular or plural proper noun tag ("NNP" or "NNPS").

The heuristic rule to detect and mark such appositions proceeds as follows: Every sequence of consecutive proper noun tags is examined. The first token of the sequence is checked against a list of 38 titles maintained by the IE. The title set includes entries such as "Senate", "Admiral", "Doctor", and "Deacon". If the token is found on the list, successive tokens are examined for membership on the list. If at some point before the end of the sequence a proper noun token is found not to be on the list, then a proper noun apposition is constructed by inserting an open apposition comma at that point (IE tag "capo") and a close apposition comma following the last token in the sequence (IE tag "capc"). If all tokens in the sequence are on the title list, or if the first token is not on the list, then the sequence is not marked as a proper noun apposition.
4.1.9 Noun Phrase Appositions

The input text may contain appositions already set off by commas, such as:

"Charles Musson, an engineer for an oil company here, stood in front of a portrait ... ."

Here, the phrase "an engineer for an oil company here" is in apposition to the proper noun phrase "Charles Musson" and is set off by commas in the original input text.

The heuristic rule for detecting and marking such appositions is as follows: The phrase introduced by a comma preceded by a noun (or by a noun and a period) is examined to the next comma or other clause terminator, such as a semicolon, period, colon, question mark, or exclamation point. If the phrase so enclosed contains a noun phrase but no verb phrase, as in our example above, then the phrase is considered an apposition and the leading comma is marked with the "capo" IE tag. If the apposition terminator is a comma, then that comma is marked with the "cape" IE tag. However, if the leading comma had been previously marked as a coordination comma (IE tag "ccrd"), the closing comma is marked for the coordination. This ensures that the apposition is included together with its antecedent noun phrase in any subsequent coordination split. Moreover, if the leading comma had been previously marked as a coordination comma but the phrase terminator is not a comma, then a coordination comma is inserted at the phrase termination point to preserve the coordination split options.
4.1.10 Commas Introducing Adverbials

The input text may contain adverbial phrases and clauses that are introduced by a comma followed by an adverb. For example, consider the sentence fragment: "The flag, once 42 feet by 30 feet but now reduced to 33 by 30, ...." Here, the comma followed by the adverb "once" introduces an adverbial phrase. The IE marks the comma that precedes the adverb with the "cavo" tag if it has not previously been tagged. If the adverbial phrase is terminated by another unmarked comma, then that comma is tagged "cavo".

4.1.11 Indirect Quotes

An indirect quote is a reference to an utterance, such as in the following sentence: "Not only would he have to wait for treatment, the doctor said, but he would have to wait for another doctor to confirm he needed treatment, for a total of about two and a half years." In this sentence, the clause "the doctor said" is itself a complete sentence, as it contains both noun ("the doctor") and verb ("said") phrases. However, because from a semantic viewpoint it would not make sense to split it off from the rest of the sentence, the IE detects and marks such situations.

The IE maintains a list of present and past tense indirect quote verbs, such as "stated", "says", "asserted", and "claimed". The IE scans an input sentence for such verb forms and, if found, marks the verb with the "m" tag prefix. If a comma preceded the
phrase or clause in which such verb was found, and if no noun phrase was observed within
the phrase or clause, then the leading comma is also tagged "cqi" to indicate an indirect
quotation. Since indirect quotations, unlike direct quotations, do not ordinarily contain a
comma following the indirect quotation verb and the description of the matter quoted, no
search is made for a following comma and none is marked by this rule.

4.1.12 "Wh-" Relative Clauses

A "wh-" relative clause is a subordinate clause introduced by a "wh-" relative
pronoun such as "what", "which", "who", "whose", "whom", and their compounds
("whoever", etc.). The word "that" is also considered a relative pronoun. Consider the
sentence: "Estelle, who lived until 1909, led a tragic life quite nobly." In this sentence,
"who" is a "wh-" relative pronoun and is preceded by a comma. The IE detects and marks
such situations by examining all sentence tokens that have a Brill tag of "WP"
(wh-pronoun) or "WDT" (wh-determiner). If any such token is preceded by an otherwise
unmarked comma, then that comma is tagged with IE tag "crel" to indicate that introduces
a "wh-" relative phrase or clause.
4.1.13 Prepositions and Subordinate Conjunctions

As discussed in Section 3.2, the Brill tagger uses the Penn Treebank "IN" tag for both prepositions and subordinate conjunctions. It is important to distinguish between the two because a subordinate conjunction introduces a clause which is a candidate for splitting off, while a preposition does not.

The IE heuristic rule for making this determination is as follows: First, only "IN" tagged tokens that start the sentence or follow commas, semicolons, colons, or commas plus quotation marks, are considered. Thus, IN-tagged tokens that do not introduce phrases or clauses are not examined. Next, the token is checked against a list of 15 subordinate conjunctions maintained by the IE. If the token is on the list, then the phrase or clause in which it appears is examined for noun phrases, gerund verb forms, and other verb phrases. If a noun phrase or a gerund is found and if it precedes a verb phrase that is also observed, then the "IN" token receives the IE tag "scnj" to indicate that it is a subordinate conjunction, and the phrase or clause terminator, if it is a previously unmarked comma, receives the "csub" IE tag.

If the token is not on the list, or if a noun phrase or gerund is not found preceding a verb phrase, the token receives the "prep" IE tag to indicate that it is a preposition, and the phrase or clause in which it appears is not considered further. The IE then proceeds to search for more "IN" tagged tokens following the phrase or clause terminator.
4.1.14 Unknown Commas

For the sake of consistency in the tagging of commas with letter-based tag codes, all commas not otherwise tagged are marked with the "cf" IE tag. This rule is applied after all other rules that could affect comma tags are applied.

4.1.15 Parenthetical Expressions

The IE marks all open parentheses with the "prno" tag, and all close parentheses with the "prnc" tag. Open and close parentheses must be evenly matched. If an open parenthesis is detected and the IE detects the end of the sentence before the corresponding close parenthesis is encountered, the IE will insert a close parenthesis immediately before the period that terminates the sentence and will mark it with the "prnc" IE tag. If the input file contains a close parenthesis that is not preceded by an open parenthesis, it is tagged but otherwise ignored.

4.1.16 Expressions Separated by Dashes

Phrases and clauses that are set off by the double-dash symbol "--" can appear at the beginning or end of a sentence. Since open and close dashes use the same symbol, the IE considers the first dash in a sentence as an open dash. Subsequent dashes alternate.
between closed and open. Like parentheticals, if an open dash was observed but the end of the sentence is encountered before a closing dash is observed, a closing dash is invented immediately before the terminal period.

4.2 Sentence Splitting Rules

Sentence splitting heuristic rules are applied after all post-tagging of words and punctuation marks is completed. These rules make use of all three available knowledge sources: (i) the word tags obtained from the Brill tagger, as augmented by the IE post-tagging rules, (ii) the punctuation marks, as tagged by the IE post-tagging rules, and (iii) in certain instances, the words themselves. The sentence splitting rules are set forth in the following sections.

4.2.1 Comma-"Which" Splits

When the word "which" is used following a comma, it often introduces a nonrestrictive relative clause, that is, a subordinate clause that modifies or adds detail to its antecedent, but which if omitted, does not destroy the essential meaning of the sentence. For example, consider the following sentence, taken from the IE test corpus:

Every patient in Britain knows about the notorious National Health Service waiting list, which tells how many people are waiting for hospital treatment for anything from varicose vein removal to hip replacement surgery.
In this example, the which-clause adds detail to the prior clause, but if omitted, the prior clause stands on its own without loss of meaning.

The IE heuristic rule for determining whether a comma-which clause should be split off is as follows: First, the rule locates the occurrence of a comma followed by the word "which". For this purpose, such occurrences within direct quotes are not considered. Next, from that token, the sentence is examined until the next comma, semicolon, colon, question mark, exclamation point, or the end of the sentence is reached. Intervening apposition commas are ignored for this purpose. If a noun phrase or a gerund verb form is observed within the clause, and it precedes a modal or any verb form other than a gerund form, then the IE concludes that an independent clause has been observed.

In that case, if the comma-which clause ends prior to the end of the sentence, or if both a noun phrase and a verb phrase (i.e., a complete clause) had been previously observed in the main clause before the comma-which occurrence, the IE processes the comma-which clause for splitting. This is accomplished by marking the clause terminating comma, if any, with the "x-wh" IE end split tag, but if the terminator is not a comma, a comma is inserted with this tag. The IE will also replace the word "which" with the "/p-wh" IE begin split token and insert the token "[prior]/nnwh" at the end of the split off clause to indicate that the object of the split off clause refers to either the entire antecedent clause or to the immediately preceding noun phrase. The IE does not distinguish between these two cases.
The rule also processes a split if only a verb phrase is observed or if a verb phrase precedes a noun phrase within the comma-which clause, provided the comma-which clause ends prior to the end of the sentence, or both a noun phrase and a verb phrase (i.e., a complete clause) had been previously observed in the main clause before the comma-which occurrence. In this case, the terminal comma is marked or a terminal comma is inserted, and the "which" is replaced by the open split tag, as above. However, in this case the "[prior]/nnwh" is inserted at the beginning of the split off clause to indicate that the prior clause or noun phrase, as the case may be, serves as the subject of the split off clause. In accordance with this part of the rule, the example sentence above is correctly split as follows:

Every/DT patient/NN in/IN Britain/NNP knows/VBZ about/IN the/DT notorious/JJ National/NNP Health/NNP Service/NNP waiting/VBG list/NN ,/crel ,/p-wh ./

[prior]/nnwh tells/mVBZ how/WRB many/JJ people/NNS are/VBP waiting/VBG for/IN hospital/NN treatment/NN for/IN anything/NN from/IN varicose/NN vein/NN removal/NN to/TO hip/NN replacement/NN surgery/NN ,/x-wh

4.2.2 Comma-Preposition-"Which" Splits

As with comma-which clauses, a comma-preposition-which occurrence often signals the beginning of a complete clause that can be split off. For example, consider the following sentence, taken from the IE test corpus:
Some scholars here argue that from ancient times until perhaps 150 years ago, virtually all Japanese learned to walk in a special style called the namba, in which the right arm and leg swing forward at the same time, and then the left arm and leg swing forward.

Here again, the clause introduced by the ",,-in-which" explains and adds detail to the preceding clause, but if omitted, the preceding clause can stand alone without a change in its meaning.

The IE heuristic rule for determining whether a comma-preposition-which clause should be split off is as follows: First, the rule locates the occurrence of a comma followed by token tagged with "prep" and then the word "which". For this purpose, such occurrences within direct quotes are not considered. Next, from that token, the sentence is examined until the next comma, semicolon, colon, question mark, exclamation point, or the end of the sentence is reached. Intervening apposition commas are ignored for this purpose. If a noun phrase or a gerund verb form is observed within the clause, and it precedes a modal or any verb form other than a gerund form, then the IE concludes that an independent clause has been observed.

In that case, if the comma-preposition-which clause ends prior to the end of the sentence, or if both a noun phrase and a verb phrase (i.e., a complete clause) had been previously observed in the main clause before the comma-preposition-which occurrence, the IE processes the comma-which clause for splitting. This is accomplished by marking the clause terminating comma, if any, with the "x-pw" IE end split tag, but if the terminator is not a comma, a comma is inserted with this tag. The IE will also replace the
preposition token with ",/p-pw" IE begin split token. It will also insert a copy of the preposition token (with its "prep" tag) and the token "[prior]/nnwh" at the end of the split off clause to indicate that the object of the split off clause refers to either the entire antecedent clause or to the immediately preceding noun phrase. The IE does not distinguish between these two cases. In accordance with this heuristic rule, the example sentence above is correctly split as follows:

Some/DT scholars/NNS here/RB argue/VBP that/IN from/IN ancient/JJ times/NNS until/IN perhaps/RB 150/CD years/NNS ago/RB [,]/zCC virtually/RB all/DT Japanese/NN learned/VBD to/TO walk/VB in/IN a/DT special/JJ style/NN called/VBD the/DT namba/NN ,/capo ,/p-pw ./

The/DT right/JJ arm/NN and/zCC leg/NN swing/NN forward/RB at/IN the/DT same/JJ time/NN ,/capc and/CC then/RB the/DT left/VBN arm/NN and/zCC leg/NN swing/NN forward/RB in/prep [prior]/nnpw ,/x-pw

4.2.3 Comma-"With"-Gerund Splits

The gerund verb form is a verb form constructed from the base verb form by adding the suffix "-ing" and which serves as a noun in a sentence, as in: "Peter likes swimming." When the word "with" immediately follows a comma, and a gerund is contained in the clause introduced thereby, such an occurrence often signals the beginning of an independent clause that can be split off. For example, consider the following sentence, taken from the IE test corpus:
But using a large fishing boat, tribal members managed to refloat it several hours later and tow it ashore, with two hunters standing atop the whale in a triumphal pose.

In this example, the clause introduced by the comma-"with" and containing the gerund "standing" can clearly stand alone, as can the antecedent clause, and so should be split off.

The IE heuristic rule for determining whether a comma-with-gerund clause should be split off is as follows: First, the rule locates the occurrence of a comma followed by the word "with". For this purpose, such occurrences within direct quotes are not considered. Next, from that token, the sentence is examined until the next comma, semicolon, colon, question mark, exclamation point, or the end of the sentence is reached. Intervening apposition commas are ignored for this purpose. If a gerund verb form (Brill tag "VBG") is observed within the clause, then the IE concludes that an independent clause has been observed.

In that case, if the comma-with-gerund clause ends prior to the end of the sentence, or if both a noun phrase and a verb phrase (i.e., a complete clause) had been previously observed in the main clause before the comma-preposition-which occurrence, the IE processes the comma-which clause for splitting. This is accomplished by marking the clause terminating comma, if any, with the "x-wi" IE end split tag, but if the terminator is not a comma, a comma is inserted with this tag. The IE will also replace the "with" token with "/p-wi" IE begin split token. It will also convert the gerund verb form to the past tense form and tag it with "vbd" to indicate that the tense was changed by the IE. In
accordance with this heuristic rule, the example sentence above is correctly split as follows:

But/CC using/VBG a/DT large/JJ fishing/NN boat/NN ./ccrd tribal/JJ members/NNS managed/VBD to/TO refloat/VB it/PRP several/JJ hours/NNS later/RB and/CC tow/NN it/PRP ashore/RB ./cf ./p-wi ./.

Two/CD hunters/NNS stood/vbd atop/IN the/DT whale/NN in/IN a/DT triumphal/JJ pose/NN ./x-wi

4.2.4 Subordinate Clauses

Subordinating conjunctions, such as "because", "although", and "until", introduce subordinate clauses and connect them to main clauses. In the IE, a post-tagging rule determined whether an "IN"-tagged token was a subordinating conjunction or a preposition. This heuristic rule determines whether the subordinate clause introduced by a subordinating conjunction should be split off. For example, consider this sentence, taken from the IE test corpus:

As the men circled the whale shortly before 7 o'clock this morning, many Makah gathered to watch the event on television.

The word "as" in this sentence is a subordinating conjunction that introduces the clause ending with the comma.

The IE heuristic rule for determining whether to split off such a clause operates as follows: First, the sentence is scanned to identify the next subordinating conjunction,
which was tagged "scnj" by the IE post-tagging rule, but not including the conjunction "if". Next, the IE locates the corresponding comma that ends the clause, previously tagged "csub". This comma may not exist if the clause ends with the end of sentence period. The clause identified by these two endpoints is processed for splitting if either of the following conditions hold:

(a) The subordinating conjunction is not the first word token of the sentence and both a noun phrase (or gerund) and a verb phrase precede the subordinating conjunction, but not if any of the words "that", "neither", or "nor" were observed without an accompanying indirect quote verb; or

(b) If the subordinating conjunction is the first word token (it may follow a close quotation mark), but not if any of the words "that", "neither", or "nor" were observed without an accompanying indirect quote verb.

In situation (a), above, the subordinating conjunction is replaced by the IE split marker token "./p-cj" and the comma, if any, that closes the clause is marked with the "x-cj" IE end split tag. In situation (b), above, the leading subordinating conjunction is deleted from the sentence buffer entirely, and the closing comma, if any, is tagged with the "p-cj" IE split tag. Applying rule (b) above to our example, it is correctly split as follows:

```
The/DT men/NNS circled/VBD the/DT whale/NN shortly/RB before/IN 7/CD o/IN 'clock/VBG this/DT morning/NN ./p-cj

Many/JJ Makah/NNP gathered/VBD to/TO watch/VB the/DT event/NN on/IN television/NN ./
```
4.2.5 Comma-"But" Splits

The word "but" is a coordinating conjunction that often, but not always, introduces a subordinate clause, particularly when it follows a comma. For example, consider the following sentence, taken from the IE test corpus:

Jokes aside, Dole said he had been "a little bit concerned" that Mrs. Dole's campaign had a slow start, but that people should be patient given that Mrs. Dole, a two-time Cabinet secretary who is 62, had been until recently the president of the American Red Cross.

In this sentence, the clause preceding the "but" is independent and can stand on its own. Similarly, the clause beginning with the "but" and ending with the end of the sentence, including all embedded commas, can also stand on its own.

The IE heuristic rule for identifying and splitting off such occurrences operates as follows: First, an occurrence of the word "but" following a comma and not within a direct quote and not within an idiomatic expression or collocation (such as "but never") is identified. The remainder of the sentence is examined up until the next exclamation point, comma, semicolon, token within direct quotes, or the end of the sentence is reached. Intervening apposition commas are ignored. The clause identified by these two endpoints is processed for splitting off if either of the following two conditions holds:

(a) If both a noun phrase (or gerund) and a verb phrase are found in the subordinate clause, and if the noun phrase precedes the verb phrase (including modals), and either the clause does not terminate with the end of the sentence, or if it does, then both a noun phrase (or gerund) and verb phrase (including modals) was observed in the part of the sentence preceding the word "but"; or
(b) If only a verb phrase (including modals) was found in the subordinate clause, or if the verb phrase preceded any noun phrase (including gerunds) and the clause ends prior to the end of the sentence.

In both situations above, the IE rule replaces the "but" token with the ".//p-bt" IE split marker if it was previously tagged as a coordinating conjunction. Otherwise, such as where the word "but" is part of an idiomatic expression or collocation, a split marker is inserted before it and the word "but" will remain a part of the split off sentence.

Moreover, in case (b), where only a verb phrase was found, the place holder token "[Subj]/p-bt" is inserted at the beginning of the split off clause to indicate that the subject of the clause is the same as the subject for the previous clause. Applying situation (a) to our example correctly splits it into:

Jokes/NNS aside/RB ./ccrd Dole/NNP said/mVBD he/PRP had/VBD been/VBN "/qo a/qDT little/qJJ bit/qNN concerned/qVBN "/qc that/DT Mrs/NNP ./pd Dole/NNP 's/POS campaign/NN had/VBD a/DT slow/JJ start/NN ./cf ./p-bt

People/NNS should/MD be/VB patient/NN given/VBN that/DT Mrs/NNP ./pd Dole/NNP ./cf a/DT two-time/JJ Cabinet/NNP secretary/NN who/WP is/VBZ 62/CD ./cf had/VBN been/VBN until/IN recently/RB the/DT president/NN of/IN the/DT American/NNP Red/NNP Cross/NNP ./

4.2.6 Dashed Expressions

IE post-tagging rules mark open and close dashes, including adding close dashes at the end of a sentence to match a prior open dash. This makes the corresponding sentence
splitting rule straightforward. For example, consider the following sentence, taken from
the IE test corpus:

But days after the interview with her, Mrs. Dole called for a ban on assault
weapons -- something her husband had opposed in the Senate.

In this example, only an open dash appears in the original text. The clause set off by the
dash contains both a noun phrase and a verb phrase, and should be split off.

The IE heuristic rule for determining whether to split a clause enclosed by dashes
operates as follows: First, the clause is identified by its open and close IE tags. Recall
that the IE post-tagging rule guaranteed that for every open dash in the sentence there
would be a close dash. The words within the clause are then examined: if a noun phrase
or gerund is identified, and if a verb phrase, including modals, is also found prior to any
occurrence of the word "to", and if the noun phrase precedes the verb phrase, then the
dashed expression is processed for splitting off. In that case, the open dash tag is replaced
by the "p-ds" split dash tag and the close dash is replaced by the "x-ds" end split tag.

Applying this rule to the sample sentence produces the following correct split:

But/CC days/NNS after/IN the/DT interview/NN with/IN her/PRP$ ./cf
Mrs/NNP ./pd Dole/NNP called/VBD for/IN a/DT ban/NN on/IN
assault/NN weapons/NNS --/p-ds ./

Something/NN her/PRP$ husband/NN had/VBD opposed/VBN in/IN
the/DT Senate/NNP --/x-ds

42
Parenthesized expressions are handled in the same manner as dashed expressions, due to the enforcement of matching close parentheses and dashes by the IE post-tagging rules. For example, consider the following sentence, taken from the IE test corpus:

A top graduate of Stanford Law School in 1952 (her classmate, William H. Rehnquist, went on to a Supreme Court clerkship, an opportunity that was not then open to women), Sandra O'Connor applied to law firms only to receive job offers as a secretary.

In this example, the clause enclosed by the parentheses contains both a noun phrase and a verb phrase, and should be split off.

The IE heuristic rule for determining whether to split a clause enclosed by parentheses operates as follows: First, the clause is identified by its open and close IE tags. Recall that the IE post-tagging rule guaranteed that for every open parenthesis in the sentence there would be a close parenthesis. The words within the clause are then examined: if a noun phrase or gerund is identified, and if a verb phrase, including modals, is also found prior to any occurrence of the word "to", and if the noun phrase precedes the verb phrase, then the parenthesized expression is processed for splitting off. In that case, the open parenthesis tag is replaced by the "p-pr" split dash tag and the close dash is replaced by the "x-pr" end split tag. Applying this rule to the sample sentence produces the following correct split:

A/DT top/JJ graduate/NN of/IN Stanford/NNP Law/NNP School/NNP in/IN 1952/CD (/p-pr /cf Sandra/NNP O/NNP 'Connor/NN applied/VBN
Coordinations involve the linking of parallel grammatical structures, such as verb phrases or independent clauses, by the use of specific words that express the relationship between the structures. Where more than two structures are linked, the structures are commonly separated by commas, with only one instance of the coordinating conjunction (connecting word) usually appearing between the last two structures listed. For example, consider this sentence, taken from the IE test corpus:

Under the system Ms. Elliott devised, Salinas' third wife, Paulina Castanon, would pick up cashier's checks in pesos at Mexican banks, carry them to the Citibank subsidiary in Mexico City, convert them to dollars and wire them to New York, using the name Patricia Rios, a first name that she did not use combined with her mother's maiden name.

This sentence involves verb phrase coordination, with parallel verb phrase structures commencing with the words "pick", "carry", "convert", and "wire". The verb phrases are separated by commas, except for the last, for which the coordinating conjunction "and" appears immediately before the word "wire". The sample sentence also contains a modal verb, "would", before the verb "pick" in the main clause. Accordingly, the sample
sentence should be split before each of the verbs, and the modal form carried forward to each of the split-off sentences, in order to maintain the parallelism of the sentence structures.

The IE heuristic rule for determining sentence and verb phrase splits is relatively complex, as it includes the logic necessary to propagate modal verb forms, infinitives, and other multiple verb constructions into the split-off sentences. The basic framework of the rule, however, is straightforward. The rule operates as follows: The first clause of the sentence, which the IE takes to be the main clause, is examined. The main clause is considered to run from the beginning of the sentence up to the next coordinating conjunction, colon, exclamation point, coordination comma, semicolon, split marker, or end of sentence, whichever comes first.

The sentence is not considered for splitting at all if the main clause, as defined above, contains any of the words "that", "neither", or "nor", or if the main clause does not contain a noun phrase that precedes a verb phrase. For this evaluation, a gerund, the existential "there" (Brill tag "EX"), and cardinal numbers count as noun phrases, and any verb form other than the gerund or present participle (Brill tag "VBG"), but not modals, count as verb phrases. While examining the main clause, the heuristic rule takes note whether modals and infinitive verb forms were observed. In addition, as each successive verb is encountered, a multiple verb construction list is regenerated, containing the set of verb forms which, possibly together with adverbs, appear in sequence immediately before
such verb. Thus, when the main clause is fully examined, the list contains the set of verb forms associated with the last verb encountered in the clause.

Once the above conditions are satisfied, the rule shifts attention to the clause or clauses that begin after the main clause, if any. Starting from the end of the main clause, the end of the next clause is determined to be the next succeeding coordinating conjunction, colon, exclamation point, coordination comma, semicolon, end of sentence, or split marker. This clause is examined for noun phrases, with the same inclusions as for the main clause, and for verb phrases. For this purpose, however, the rule is different: modals and all verb forms are included, except for present participles or gerunds, and except for past participles. The exclusion of past participles was made because past participles are frequently used as adjectives. The rule also keeps track of modals, infinitives, and multiple verb constructions in the same manner as for the main clause.

If both a noun phrase and a verb phrase were observed in the clause, and if the noun phrase preceded the verb phrase, and if the clause is not otherwise marked for splitting, and if the words "that", "neither", or "nor" are not found within the clause, and if the word "to" is not found before any verb form, then the clause is processed for splitting as a sentence coordination. In such event, a ".p-sc" IE split token is inserted at the beginning of the clause.

Otherwise, if a verb phrase was observed in the clause, and if the clause has not already been marked for splitting, and if the word "to" was not found before any verb form in the clause, then the clause is processed for splitting as a verb phrase coordination. In
this case, a "/p-svp" IE split token is inserted at the beginning of the clause. The token
"[Subj]/p-vp" is inserted immediately following the split token, to indicate that the subject
of the split off sentence is the same subject as in the main clause. Additionally, if the main
clause contained a modal and the first verb in the split off clause does not, then a copy of
the modal, with the IE tag "md", is inserted immediately following the subject token.
Alternatively, if an infinitive was detected in the main clause (by finding the word "to"
occurring between the last two verbs observed in that clause), then the main clause verb
preceding the word "to", with a lower case tag to indicate that it was inserted by the IE,
and the word "to", with the IE tag "to", are inserted immediately after the subject token.
Also alternatively, if the last verb of the main clause was the head noun of a multiple verb
construction, and if the first verb of the split off clause contained fewer such "auxiliary"
verbs, then as many verbs as the difference between the two counts, are propagated to the
split off sentence, with lower case tags to indicate that they were added by the IE.

Whether or not a given clause is split off, the rule's attention shifts to the next
clause in line, applying the above rules once more, and so on, until all clauses of the
sentence are examined.

Applying the above heuristic rule to the sample sentence above, the IE produces:

Under/prep the/DT system/NN Ms/NNS/.pd Elliott/NNP devised/VBD
/cf Salinas/NNP '/POS third/JJ wife/NN ./capo Paulina/NNP
Castanon/NNP ./capc would/MD pick/VB up/IN cashier/NN's/POS
checks/NNS in/IN pesos/NNS at/IN Mexican/JJ bank/d/W.

[Subj]/p-vp would/md carry/VB them/PRP to/TO the/DT Citibank/NNP
subsidiary/NN in/IN Mexico/NNP City/NNP ./ccrd ./p-vp
This is the correct result, taking into account the Brill tagger's mis-tagging of the verb "wire" as a noun ("NN"), which, together with the presence of the word "that" in the last clause, precluded splitting that clause. Parenthetically, an experiment was conducted in which the tag for "wire" was changed to "VB" to be the same base form as for the other verb structures, and in which the word "that" was eliminated. Operating on the resultant sentence, the IE correctly made all three splits and correctly propagated the modal to all split off sentences.

4.3 Utility Rules

In addition to the post-tagging and sentence splitting rules, there are numerous additional algorithms that are employed by the IE, mostly in support roles. However, some of those utility algorithms rise to the level of heuristic rules in their own right, and are described in the following sections.
4.3.1 Paragraph Spanning Direct Quotations

Direct quotations that span paragraph boundaries have the additional complication that a quotation mark appears, in the original plain text file, at the beginning of each paragraph within the quotation, thereby destroying the one-to-one correspondence between open and close quotation marks. Moreover, in passing the input text file through the Brill tagger, the output is an unbroken sequence of tagged sentences, one per line. As a result, all paragraph boundary information is lost.

The IE detects and corrects this situation by applying a simple rule prior to passing the input file through the Brill tagger. The rule operates as follows: As the input file is pre-processed for submission to the tagger, the open and closed status of quotations is maintained. The status is initially closed, but upon encountering the first quotation mark, the status is changed to open. Succeeding quotation marks then toggle the status marker between closed and open status. The rule then provides that if at any time the quotation status is open and a quotation mark is encountered as the first token of a sentence buffer, then that token is ignored. By removing the leading quotation marks for continuing quotations across paragraph boundaries in this manner, the input file to the tagger is transformed to contain matching open and close quotes in all cases.
4.3.2 Converting Gerund Verb Form to Past Tense

To support the processing of comma-"with"-gerund splits, a simple rule is employed for transforming the gerund verb form to the simple past tense. The IE maintains a list of common irregular verb gerunds, together with their third person singular past tense forms. A gerund on the list is converted to the past tense form specified on the list. For gerunds not on the list, the past tense form is obtained by stripping of the "-ing" suffix and appending an "-ed" suffix.

4.3.3 Periods That Do Not End Sentences

Inasmuch as the purpose of the IE is to split sentences, it has been designed to process tagged input sentences with their associated punctuation marks. Accordingly, it is essential for the IE to determine where sentence boundaries are. This is not a trivial task, since the Brill tagger tags all periods the same. Thus, periods embedded within decimal numbers, or following abbreviations or initials, are tagged the same as periods that end sentences.

The IE solves this problem in two stages. First, as discussed in Sections 4.1.1 and 4.1.3, the periods embedded within decimal numbers or multiple initial identifiers are removed from consideration by embedding them within their respective non-period tokens.
Second, the IE maintains a list of over 200 entries that commonly precede periods that do not mark the ends of sentences. Entries include the letters of the alphabet (initials), the months of the year, corporate insignia such as "Corp", forms of address such as "Mrs", offices such as "Treas", military rank such as "Col", directions such as "So", titles such as "Rev" and "Dr", miscellaneous items such as "vs" and "etc", and legislative offices such as "Sen" and "R-Ariz".

As the Brill tagger output is read into the IE, upon encountering a period, the rule is invoked. The rule examines the token preceding the period, and if it is on the list, then the IE concludes that a sentence-ending period has not been found. Otherwise, it concludes that the end of the sentence was found.

4.3.4 Outputting Sentences

To facilitate the analysis of IE output, the various split and end-split tags, as well as the occurrence of multiple successive punctuation marks inserted by the heuristic rules, are retained for output. However, to enhance readability, certain leading tokens are dropped on output. The heuristic rule that determines which tokens are dropped, is an integral part of the sentence splitting heuristic rule set, because without its operation, many split off sentences would not appear to have been split properly.

To this end, the IE maintains a list of initial adverbs that are not to be deleted, such as "Almost", "Some", and "Never". The first token of each sentence, whether main clause
or split off clause, is checked against the listed adverbs. If it is on the list, then no tokens are deleted from the sentence. Otherwise, if it is a leading subordinate conjunction, it is removed from the sentence buffer. Alternatively, if it is a leading sequence of adverbs and Wh-adverbs, they are all deleted, including any trailing comma, if any, provided they are followed by a noun phrase preceding a verb phrase. Also, if the initial token is a coordinating conjunction followed by a sequence of adverbs and Wh-adverbs, they and any trailing comma are all deleted if they are followed by a noun phrase preceding a verb phrase. But if the first token has a tag of "IN", then it is deleted if it is followed by a noun phrase preceding a verb phrase. Finally, the rule deletes all spurious initial tokens before the first word, including quotation marks and parentheses.

For ease of reading, once all deletions are made, sentences are output with the leading alphabetic character capitalized, and with each main sentence or split off sentence beginning on a separate line.

4.4 The Augmented Tagset

The various IE post-tagging, sentence splitting and utility heuristics introduce a number of tags that are not included in the Penn Treebank tagset shown in Table 1. For convenient reference, these extensions are listed in the following tables.

Table 2 lists the basic word and punctuation mark tags introduced by the IE. These are tags that generally replace tags produced by the Brill tagger or are tags for new
tokens introduced by the IE and serve as grammatical objects, indicated by the token identifiers "[prior]" and "[Subj]".

Table 2
Intelligent Editor Augmented Tagset

<table>
<thead>
<tr>
<th>IE Tag</th>
<th>Tag Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>capo, capc</td>
<td>open and close apposition commas</td>
</tr>
<tr>
<td>cavo, cavc</td>
<td>open and close adverbial clause commas</td>
</tr>
<tr>
<td>ccrd</td>
<td>coordination comma</td>
</tr>
<tr>
<td>cf</td>
<td>unknown (futile) comma</td>
</tr>
<tr>
<td>cqdo, cqdc</td>
<td>open and close direct quote commas</td>
</tr>
<tr>
<td>cqi</td>
<td>leading comma before indirect quote</td>
</tr>
<tr>
<td>crel</td>
<td>comma preceding a wh-relative clause</td>
</tr>
<tr>
<td>csub</td>
<td>closing comma of subordinate clause</td>
</tr>
<tr>
<td>dsho, dshc</td>
<td>open and close dashes</td>
</tr>
<tr>
<td>md</td>
<td>modal verb form</td>
</tr>
<tr>
<td>nnpa</td>
<td>proper noun not part of name (e.g., a title)</td>
</tr>
<tr>
<td>[prior]/nnpw</td>
<td>prior object for comma-prep-which clause</td>
</tr>
<tr>
<td>[prior]/nnwh</td>
<td>prior object for comma-which clause</td>
</tr>
<tr>
<td>pd</td>
<td>period that does not mark end of sentence</td>
</tr>
<tr>
<td>pq</td>
<td>question or exclamation mark before quote</td>
</tr>
<tr>
<td>prep</td>
<td>preposition</td>
</tr>
<tr>
<td>prno, prnc</td>
<td>open and close parentheses</td>
</tr>
<tr>
<td>qo, qc</td>
<td>open and close quotes</td>
</tr>
<tr>
<td>scnj</td>
<td>subordinating conjunction</td>
</tr>
<tr>
<td>[Subj]/p-bt</td>
<td>prior subject of comma-but clause</td>
</tr>
<tr>
<td>[Subj]/p-vp</td>
<td>prior subject of verb phrase coordination</td>
</tr>
<tr>
<td>that</td>
<td>the word <em>that</em></td>
</tr>
<tr>
<td>to</td>
<td>the word <em>to</em></td>
</tr>
</tbody>
</table>
Table 3 lists the prefixes that are attached to other Brill and IE tags by some of the IE heuristic rules. It also records that the IE modifies certain Brill tags by rendering them in lower case when propagating tokens from main clauses to split off sentences, so that the output indicates clearly that such tokens were added by the IE.

Table 3
Intelligent Editor Tag Prefixes and Modified Brill Tags

<table>
<thead>
<tr>
<th>IE Tag Prefix or Modified Brill Tag</th>
<th>Tag Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>i[tag]</td>
<td>idiomatic expression or collocation</td>
</tr>
<tr>
<td>m[tag]</td>
<td>indirect quote verb</td>
</tr>
<tr>
<td>q[tag]</td>
<td>token is within a direct quote</td>
</tr>
<tr>
<td>z[tag]</td>
<td>non-sentence splitting coordinating conjunction</td>
</tr>
<tr>
<td>[lower case Brill tag]</td>
<td>verb form propagated from one clause to another when propagate infinitive, modal, or multiple verb phrase</td>
</tr>
</tbody>
</table>

Table 4 displays the sentence split tags introduced by the IE. The reader should note that the sentence and verb phrase coordination splits, as well as the comma-but split, do not have end-split tags. This is because the clauses introduced by splits of these kinds do not ordinarily occur as clauses contained entirely within other clauses.
Table 4
Intelligent Editor Sentence Split Tags

<table>
<thead>
<tr>
<th>IE Split Tag</th>
<th>Tag Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-bt</td>
<td>split off comma-but clause</td>
</tr>
<tr>
<td>p-cj, x-cj</td>
<td>begin and end subordinate clause split</td>
</tr>
<tr>
<td>p-ds, x-ds</td>
<td>begin and end dashed clause split</td>
</tr>
<tr>
<td>p-pr, x-pr</td>
<td>begin and end parenthesized clause split</td>
</tr>
<tr>
<td>p-pw, x-pw</td>
<td>begin and end comma-prep-which split</td>
</tr>
<tr>
<td>p-sc</td>
<td>sentence coordination split</td>
</tr>
<tr>
<td>p-vp</td>
<td>verb phrase coordination split</td>
</tr>
<tr>
<td>p-wh, x-wh</td>
<td>begin and end comma-which split</td>
</tr>
<tr>
<td>p-wi, x-wi</td>
<td>begin and end comma-with-gerund split</td>
</tr>
</tbody>
</table>
5. INTELLIGENT EDITOR IMPLEMENTATION

5.1 Software Architecture

The Intelligent Editor (IE) was implemented as a collection of C language programs running on a Windows95 PC environment with the GNU C compiler and the Netscape Navigator 4.05 browser. As implemented, the IE consists of three programs: (1) utility program IEPrep, which prepares text files for input to the Brill tagger, (2) utility program IEPrep2, which reformats the Brill tagger output for input to the principal IE program, and (3) NLP, the principal IE program, which executes the heuristic rules described in Chapter 4. Of the approximately 4,780 lines of commented source code, over 4,500 lines are for program NLP.

Program IEPrep scans the text file version of a New York Times online article, locates the start of the body of the article, and prepares the tokens for input to the Brill tagger. This preparation, called "lemmatizing", consists of: (a) preceding all apostrophes with spaces, (b) adding spaces before and after all punctuation marks, and (c) identifying and ignoring all leading quotation marks before new paragraphs in continuing quotations, as discussed in Section 4.3.1, above.

Program IEPrep2 reformats the Brill tagger output so that it can be viewed conveniently on the monitor and printed out. This function is necessary since the Brill
tagger generates successive output token/tag combinations on the same line, proceeding to
the next line only when encountering a period. As a result, the tag files are difficult to
view on the monitor and awkward to print out. IEPrep2 merely takes the tag file and
reformats it to a maximum line length of 70 characters.

IE functional components, other than the above described utility programs, are
depicted in Figure 1. As shown in the figure, after some initial input file cleanup as
described in Sections 4.1.1 through 4.1.3, the real work of the IE is done in a large
"while" control structure: For each sentence, the end of which is determined as described
in Section 4.3.3, the IE first executes all of the post-tagging heuristic rules that apply
augmented tags and tag punctuation marks. Then, the sentence splitting heuristics are
applied in order. After application of the heuristic rules, the resultant sentence, with all
split-offs, is output with their leading tokens cleaned up as described in Section 4.3.4. The
next sentence is analyzed and output in the same manner, and the process repeats until
there are no more sentences to process.

The "while" control structure permitted a smooth development cycle for the IE.
Individual heuristic rules were implemented as separate C functions and were added one at
a time to the appropriate component of the while loop for test and checkout. This
modular structure permits replacing existing modules or adding additional post-tagging
and sentence splitting heuristic rules with minimal impact on the existing software.
Figure 1
Intelligent Editor Functional Components

- Merge Multiple Token Identifiers
  - numbers with decimal points and commas
  - time of day
  - multiple-initial identifiers (e.g., "U.S.")

- Get Next Sentence

- Apply Augmented Tags and Tag Punctuation Marks
  - idioms
  - non-splitting CCs
  - coordination commas
  - quotation marks
  - proper noun appositions
  - appositions enclosed by commas
  - adverbial clause commas
  - indirect quote verbs
  - commas before wh-relative
  - resolve IN tag (prep/sub-conj)
  - unknown commas
  - clauses within parentheses
  - clauses within dashes

- Determine Sentence Splits
  - comma-which clause
  - comma-prep-which clause
  - comma-with-gerund clause
  - subordinate clause
  - comma-but clause
  - clause within parentheses
  - clause within dashes
  - sentence and verb phrase coordination

- Clean Up Leading Tokens and Print Sentence to Output File
5.2 Running the Intelligent Editor

The interplay among the browser, the Brill tagger, and the three IE program components is shown in Figure 2, which depicts the file transformations that occur as a New York Times online article, originally in HTML format, is processed by the various programs to produce the Intelligent Editor output file.

As shown in the figure, the browser on hand, in this case Netscape, is used to access the New York Times online articles in the first instance. Using the built-in capability of the browser, the original HTML format file is saved in text file format. Then, the IEPrep utility program is run to lemmatize the body of the article. The Brill tagger is then run on the lemmatized article to produce a tag file, which is then reformatted by the IEPrep2 utility program. The reformatted tag file is then read by the Intelligent Editor, which generate the IE output file.

For convenience, several DOS batch (script) files are provided to make the process of invoking the various programs easier. Batch file ie1.bat consists of the commands:

\[ \text{IEPrep} \%1.txt \%1.\text{lem} \\
\text{Brilltag} \%1.\text{lem} \%1.\text{tag} \]

where Brilltag invokes the Brill tagger through the single command:

\[ \text{tagger LEXICON } \%1 \text{ BIGRAMS LEXICALRULEFILE CONTEXTUALRULEFILE > } \%2 \]
Figure 2
Intelligent Editor Data Flow

HTML format New York Times article

Browser

Article in text file format

IEPrep

Lemmatized article

Brill Tagger

Tagged article

IEPrep2

Reformatted tagged article

Intelligent Editor

Intelligent Editor output file

Batch file "ie1" executes these modules

Batch file "ie2" executes these modules

Batch file "ie" executes only the Intelligent Editor
In order for iel to operate correctly, the text file version of the article must have a file extension of "txt". To run iel on text formatted article "Article.txt", the user merely types "iel \ Article" at the command prompt.

Similarly, batch file "iel2.bat" consists of the commands:

```
IELPrep2 %1.tag %1.inp
NLP %1.inp %1.out
```

which invokes the IELPrep2 utility program to reformat the Brill tagger output, and then executes the IEL main component (program NLP) on the reformatted tag file to produce the IEL output file.

As also shown in Figure 2, the separate batch file "iel.bat" executes only the IEL through the command: "NLP %1.inp %1.out". This batch file is most convenient when testing new modules or reverifying performance after a bug fix, where the reformatted tag files do not change between successive IEL runs against the same articles.

Of course, it is not necessary to use the batch files described above in order to run the IEL. Programs IELPrep, IELPrep2, NLP, and the Brill tagger can be executed separately. The syntax for doing so is contained in the batch files described above, which are provided only for the convenience of not needing to type file extensions or repeating file identifiers.
5.3 File Conventions

The NLP program component of the IE is furnished in five files: nlp.h, nlp.c, nlp1.c, nlp2.c, and nlp3.c. These files may be compiled and linked using the IE Makefile included in Appendix D.

Programs IEPrep and IEPrep2 are stand-alone program modules that can be compiled and linked using the commands: "gcc -o IEPrep IEPrep.c" and "gcc -o IEPrep2 IEPrep2.c".

The Brill tagger is furnished in a number of files and is in fact three programs: "sttag", "fintag", and "tagger". These programs may be compiled and linked using the Makefiles included in Appendix D. The Makefiles also indicate which files must be included in the default directory for compiling and linking these programs.

As discussed previously, batch files "ie1.bat" and "ie2.bat" require that their input files have a file extensions of "txt" and "inp", respectively. Given such a file, say "Article.txt", running "ie1.bat" followed by running "ie2.bat" will produce the following text files, which can be viewed or edited as desired: (1) "Article.lem", the lemmatized output of IEPrep, (2) "Article.tag", the tagged output of the Brill tagger, (3) "Article.inp", the reformatted tag file produced by IEPrep2, and (4) "Article.out", the IE output file. Different file names and extensions may be used if desired, but it is necessary to execute the various programs one at a time in order to do so.
6. TEST AND EVALUATION

6.1 Test Methodology

The testing approach adopted for the IE parallels the scope and emphasis of the IE design goals. The purpose of the test and evaluation effort was to evaluate IE performance against its primary design goal of identifying and effecting sentence splits.

The IE was evaluated only with respect to the types of sentence splits which it attempted to perform, as set forth in Section 2.1.3. Thus, splits which could be made within direct quotes, for example, but which were not even attempted by the IE, were not counted against it. However, the IE's design choice not to attempt to split sentences (in some cases) or clauses (in other cases) that contained certain words, such as "that", did not insulate the IE from being marked down for not identifying a split of one of the types which it does attempt to perform. In scoring splits, a split was taken to be correct if both its beginning point and its ending point were correctly identified by the IE. Moreover, if a split should be made as a verb coordination split, for example, but is done as a sentence coordination split, it is counted as an incorrect split. On the other hand, while some effort was made to propagate modals, infinitives, and complex verb forms, performance in this regard is secondary, and a split with correct endpoints is still regarded as a correct split even if the verb forms in the split off sentences are not quite correct.
6.2 Test Data Selection and Preparation

A test corpus of 25 articles from the New York Times online database was selected for the test program. The articles selected are identified by title, author, date of publication, and a shorthand identifier, in Table 5. Articles from a number of different subject areas, from international to domestic, political, scientific, and environmental, were purposely selected, in order better to approximate IE's goal of handling unrestricted text.

The input files were originally obtained through the Netscape web browser and saved to the local disk in HTML format. After disconnecting from the Internet, the files were read up again and saved in text file format, making use of the browser's built-in utility for doing so.

Over the course of the six months of this research, the New York Times changed the article formats somewhat. In particular, the double-dash near the byline, which is used by IEPrep to detect the beginning of the article body, was not always present. Where it was not present, it was added manually. Moreover, the more recent articles seemed to have advertisements attached to them at the margins, even in the text file versions. These were erased manually for all articles.
<table>
<thead>
<tr>
<th>Article</th>
<th>Date of Publication</th>
<th>Author and Full Title of Article</th>
</tr>
</thead>
<tbody>
<tr>
<td>Britain</td>
<td>4-18-99</td>
<td>Sarah Lyall, &quot;Britain's Prescription for health Care: Take a Seat&quot;</td>
</tr>
<tr>
<td>Cosmic</td>
<td>5-25-99</td>
<td>John Noble Wilford, &quot;In Cosmic Blasts, Clues to Black Holes&quot;</td>
</tr>
<tr>
<td>Degas</td>
<td>5-17-99</td>
<td>Bruce Weber, &quot;In a Degas Show, Impression of New Orleans' Tangled Past&quot;</td>
</tr>
<tr>
<td>Dole</td>
<td>5-17-99</td>
<td>Richard L. Berke, &quot;As Political Spouse, Bob Dole Is Admirer, Coach, and Critic&quot;</td>
</tr>
<tr>
<td>Drought</td>
<td>6-1-99</td>
<td>Sam Dillon, &quot;Parched Region Invokes God and Blames Humans&quot;</td>
</tr>
<tr>
<td>Everglades</td>
<td>2-22-99</td>
<td>William K. Stevens, &quot;Some Scientists Attack Plan to Restore Everglades&quot;</td>
</tr>
<tr>
<td>Film</td>
<td>2-22-99</td>
<td>James Sterngold, &quot;Coming Attractions: Digital Projectors Could Change Film Industry&quot;</td>
</tr>
<tr>
<td>Flag</td>
<td>5-25-99</td>
<td>Irvin Molotsky, &quot;New Dawn for Flag That Was Still There&quot;</td>
</tr>
<tr>
<td>Govs</td>
<td>2-22-99</td>
<td>Robert Pear, &quot; Governors Insist That Washington Return to Issues&quot;</td>
</tr>
<tr>
<td>Gunsales</td>
<td>5-13-99</td>
<td>Frank Bruni, &quot;Senate Rejects Background Checks at Gun Shows&quot;</td>
</tr>
<tr>
<td>Hubble</td>
<td>5-26-99</td>
<td>John Noble Wilford, &quot;Hubble Telescope Yields Data for Recalculating Age of Universe&quot;</td>
</tr>
<tr>
<td>Italy</td>
<td>12-4-98</td>
<td>Alessandra Stanley, &quot;The Italian Military Is Under Siege, in a Museum&quot;</td>
</tr>
<tr>
<td>Japanese</td>
<td>4-18-99</td>
<td>Nicholas D. Kristof, &quot;Walk This Way, or How the Japanese Kept in Step&quot;</td>
</tr>
<tr>
<td>Kosovo</td>
<td>4-18-99</td>
<td>Frank Bruni, &quot;Two Dueling Views of Reality Vying on the Airwaves&quot;</td>
</tr>
<tr>
<td>Launch</td>
<td>5-28-99</td>
<td>Beth Dickey, &quot;Shuttle Bound for Space Station, Supplies in Tow&quot;</td>
</tr>
<tr>
<td>Merger</td>
<td>6-5-99</td>
<td>Claudia H. Deutsch, &quot;Allied Signal and Honeywell Said to Be in Talks&quot;</td>
</tr>
<tr>
<td>Nireland</td>
<td>12-4-98</td>
<td>James F. Clarity, &quot;Ulster Talks Stall again, Unraveling This Time on Details&quot;</td>
</tr>
<tr>
<td>Nuclear</td>
<td>5-4-99</td>
<td>Matthew L. Wald, &quot;Group Warns of Likely Radiation Danger&quot;</td>
</tr>
<tr>
<td>Oconnor</td>
<td>5-26-99</td>
<td>Linda Greenhouse, &quot;A Conservative Voice, but Clearly a Woman's&quot;</td>
</tr>
<tr>
<td>Rockets</td>
<td>5-12-99</td>
<td>Warren E. Leary, &quot;Series of Rocket Failures Unnerves U.S. Space Launching Industry&quot;</td>
</tr>
<tr>
<td>Rubin</td>
<td>5-13-99</td>
<td>Richard W. Stevenson, &quot;A Key Architect of Prosperity&quot;</td>
</tr>
<tr>
<td>Salinas</td>
<td>12-4-98</td>
<td>Tim Golden, &quot;U. S. Report Says Salinas's Banker Ignored Safeguards&quot;</td>
</tr>
<tr>
<td>Submarine</td>
<td>5-31-99</td>
<td>Deborah Sontag, &quot;The Lost Sub Is Found, and Israelis Can Grieve&quot;</td>
</tr>
<tr>
<td>Whaling</td>
<td>5-18-99</td>
<td>Sam Howe Verhovek, &quot;Joy and Anger as Tribe Kills a Gray Whale&quot;</td>
</tr>
<tr>
<td>Wolves</td>
<td>6-1-99</td>
<td>William K. Stevens, &quot;Timber Wolf Unlikely to Return to Northeast on Its Own&quot;</td>
</tr>
</tbody>
</table>
A small amount of manual correction of the article body was also needed. In approximately three situations, sentences ended with abbreviations, such as for a state name or for the abbreviation "Corp". And in at least one case, an errant single quotation mark was replaced by a double quotation mark in order to close out the direct quotation. Except for such minor corrections, the test corpus was not manipulated.

Table 6 contains some vital statistics for the IE test corpus. From that table, we observe that the test corpus contains almost one thousand two hundred sentences, containing in excess of 32,000 words and punctuation marks. The average sentence length across the entire test corpus was 27 tokens.

Table 6
Test Corpus Composition

<table>
<thead>
<tr>
<th>Article</th>
<th>Number of sentences</th>
<th>Number of words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Britain</td>
<td>49</td>
<td>1,325</td>
</tr>
<tr>
<td>Cosmic</td>
<td>78</td>
<td>2,110</td>
</tr>
<tr>
<td>Degas</td>
<td>98</td>
<td>2,560</td>
</tr>
<tr>
<td>Dole</td>
<td>92</td>
<td>1,840</td>
</tr>
<tr>
<td>Drought</td>
<td>39</td>
<td>1,132</td>
</tr>
<tr>
<td>Everglades</td>
<td>53</td>
<td>1,391</td>
</tr>
<tr>
<td>Film</td>
<td>76</td>
<td>1,831</td>
</tr>
<tr>
<td>Flag</td>
<td>45</td>
<td>1,428</td>
</tr>
<tr>
<td>Govs</td>
<td>56</td>
<td>1,308</td>
</tr>
<tr>
<td>Gunsales</td>
<td>39</td>
<td>984</td>
</tr>
<tr>
<td>Hubble</td>
<td>42</td>
<td>1,200</td>
</tr>
<tr>
<td>Italy</td>
<td>50</td>
<td>1,169</td>
</tr>
</tbody>
</table>
The IE was run on the articles in the test corpus and performance was evaluated as described in Section 6.1. Table 7 shows the impact of the IE heuristics on the average sentence length. As the table shows, average sentence length across the entire test corpus was reduced by 16.45%, even though the maximum sentence length was not always improved. This confirms that sufficient splits are taking place with the algorithms at hand to have a significant impact on downstream parsing.

<table>
<thead>
<tr>
<th>Article</th>
<th>Number of sentences</th>
<th>Number of words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Japanese</td>
<td>38</td>
<td>1,028</td>
</tr>
<tr>
<td>Kosovo</td>
<td>40</td>
<td>1,312</td>
</tr>
<tr>
<td>Launch</td>
<td>10</td>
<td>349</td>
</tr>
<tr>
<td>Merger</td>
<td>34</td>
<td>859</td>
</tr>
<tr>
<td>NIreland</td>
<td>16</td>
<td>367</td>
</tr>
<tr>
<td>Nuclear</td>
<td>24</td>
<td>747</td>
</tr>
<tr>
<td>Oconnor</td>
<td>37</td>
<td>1,238</td>
</tr>
<tr>
<td>Rockets</td>
<td>66</td>
<td>2,042</td>
</tr>
<tr>
<td>Rubin</td>
<td>39</td>
<td>1,184</td>
</tr>
<tr>
<td>Salinas</td>
<td>41</td>
<td>1,275</td>
</tr>
<tr>
<td>Submarine</td>
<td>42</td>
<td>1,075</td>
</tr>
<tr>
<td>Whaling</td>
<td>43</td>
<td>1,340</td>
</tr>
<tr>
<td>Wolves</td>
<td>45</td>
<td>1,353</td>
</tr>
<tr>
<td>Totals</td>
<td>1,192</td>
<td>32,447</td>
</tr>
</tbody>
</table>

6.3 Experimental Results
### Table 7
Sentence Length Reduction

<table>
<thead>
<tr>
<th>Test corpus article</th>
<th>Input</th>
<th>No. of sentences</th>
<th>Max length</th>
<th>Average length</th>
<th>Output</th>
<th>No. of sentences</th>
<th>Max length</th>
<th>Average length</th>
<th>Percent reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Britain</td>
<td>49</td>
<td>70</td>
<td>27.04</td>
<td></td>
<td>59</td>
<td>54</td>
<td>22.64</td>
<td></td>
<td>16.27</td>
</tr>
<tr>
<td>Cosmic</td>
<td>78</td>
<td>60</td>
<td>27.05</td>
<td></td>
<td>95</td>
<td>58</td>
<td>22.49</td>
<td></td>
<td>16.86</td>
</tr>
<tr>
<td>Degas</td>
<td>98</td>
<td>71</td>
<td>26.13</td>
<td></td>
<td>120</td>
<td>62</td>
<td>21.61</td>
<td></td>
<td>17.3</td>
</tr>
<tr>
<td>Dole</td>
<td>92</td>
<td>60</td>
<td>20</td>
<td></td>
<td>100</td>
<td>51</td>
<td>18.87</td>
<td></td>
<td>5.65</td>
</tr>
<tr>
<td>Drought</td>
<td>39</td>
<td>56</td>
<td>29.03</td>
<td></td>
<td>48</td>
<td>51</td>
<td>23.71</td>
<td></td>
<td>18.33</td>
</tr>
<tr>
<td>Everglades</td>
<td>53</td>
<td>55</td>
<td>26.25</td>
<td></td>
<td>62</td>
<td>55</td>
<td>22.56</td>
<td></td>
<td>14.06</td>
</tr>
<tr>
<td>Film</td>
<td>76</td>
<td>59</td>
<td>24.09</td>
<td></td>
<td>87</td>
<td>53</td>
<td>21.29</td>
<td></td>
<td>11.62</td>
</tr>
<tr>
<td>Flag</td>
<td>45</td>
<td>60</td>
<td>31.73</td>
<td></td>
<td>67</td>
<td>60</td>
<td>21.58</td>
<td></td>
<td>31.99</td>
</tr>
<tr>
<td>Govs</td>
<td>56</td>
<td>66</td>
<td>23.36</td>
<td></td>
<td>60</td>
<td>67</td>
<td>22.25</td>
<td></td>
<td>4.75</td>
</tr>
<tr>
<td>Gunsales</td>
<td>39</td>
<td>53</td>
<td>25.23</td>
<td></td>
<td>48</td>
<td>50</td>
<td>20.88</td>
<td></td>
<td>17.34</td>
</tr>
<tr>
<td>Hubble</td>
<td>42</td>
<td>58</td>
<td>28.57</td>
<td></td>
<td>46</td>
<td>59</td>
<td>26.35</td>
<td></td>
<td>7.77</td>
</tr>
<tr>
<td>Italy</td>
<td>50</td>
<td>54</td>
<td>23.38</td>
<td></td>
<td>62</td>
<td>44</td>
<td>19.16</td>
<td></td>
<td>18.04</td>
</tr>
<tr>
<td>Japanese</td>
<td>38</td>
<td>68</td>
<td>27.05</td>
<td></td>
<td>44</td>
<td>68</td>
<td>23.32</td>
<td></td>
<td>13.79</td>
</tr>
<tr>
<td>Kosovo</td>
<td>40</td>
<td>76</td>
<td>32.8</td>
<td></td>
<td>48</td>
<td>77</td>
<td>27.71</td>
<td></td>
<td>15.52</td>
</tr>
<tr>
<td>Launch</td>
<td>10</td>
<td>42</td>
<td>34.9</td>
<td></td>
<td>12</td>
<td>42</td>
<td>29.42</td>
<td></td>
<td>15.7</td>
</tr>
<tr>
<td>Merger</td>
<td>34</td>
<td>66</td>
<td>25.6</td>
<td></td>
<td>41</td>
<td>66</td>
<td>21.17</td>
<td></td>
<td>16.19</td>
</tr>
<tr>
<td>Nireland</td>
<td>16</td>
<td>44</td>
<td>22.94</td>
<td></td>
<td>18</td>
<td>36</td>
<td>20.61</td>
<td></td>
<td>10.16</td>
</tr>
<tr>
<td>Nuclear</td>
<td>24</td>
<td>54</td>
<td>31.12</td>
<td></td>
<td>31</td>
<td>47</td>
<td>24.45</td>
<td></td>
<td>21.43</td>
</tr>
<tr>
<td>Oconnor</td>
<td>37</td>
<td>95</td>
<td>33.46</td>
<td></td>
<td>44</td>
<td>96</td>
<td>28.32</td>
<td></td>
<td>15.36</td>
</tr>
<tr>
<td>Rockets</td>
<td>66</td>
<td>61</td>
<td>30.95</td>
<td></td>
<td>77</td>
<td>61</td>
<td>26.81</td>
<td></td>
<td>13.38</td>
</tr>
<tr>
<td>Rubin</td>
<td>39</td>
<td>60</td>
<td>30.36</td>
<td></td>
<td>43</td>
<td>56</td>
<td>27.53</td>
<td></td>
<td>9.32</td>
</tr>
<tr>
<td>Salinas</td>
<td>41</td>
<td>71</td>
<td>31.1</td>
<td></td>
<td>55</td>
<td>52</td>
<td>23.25</td>
<td></td>
<td>25.24</td>
</tr>
<tr>
<td>Submarine</td>
<td>42</td>
<td>51</td>
<td>25.6</td>
<td></td>
<td>50</td>
<td>48</td>
<td>21.76</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>Whaling</td>
<td>43</td>
<td>67</td>
<td>31.16</td>
<td></td>
<td>62</td>
<td>64</td>
<td>21.95</td>
<td></td>
<td>29.56</td>
</tr>
<tr>
<td>Wolves</td>
<td>45</td>
<td>69</td>
<td>30.07</td>
<td></td>
<td>65</td>
<td>54</td>
<td>20.89</td>
<td></td>
<td>30.53</td>
</tr>
</tbody>
</table>

Average Reduction 16.45
The measure of how well the IE performed splits that it attempted, which we refer to as "precision", is reported in Table 8. Referring to the table, the "total splits" column refers to the total number of splits made by the IE for the given input file. In line with the IE test methodology, "correct splits" refers to the number of such splits for which both the beginning and ending points of the split were correct, as determined by a manual analysis of the input file. The column for "botched splits" records those splits for which either or both endpoints is incorrect. The "bad tag" category refers to incorrect splits which were caused by incorrect Brill tags, such as, for example, tagging the verb "wire" as a noun, in the example cited in Section 4.2.8. Finally, the "spurious splits" category contains the number of splits which should not have been made at all.

Within this context, we define precision to be:

\[
\text{Precision} = \frac{(\# \text{ correct splits})}{(\# \text{ total splits}) - (\# \text{ bad tag splits})}
\]

Thus, for example, for the "Kosovo" article, where there were 8 total splits, of which 7 were correct and one was a bad tag split, the precision was nevertheless 100%. We use this measure of precision as we feel it appropriately measures how well the IE makes the splits that it should be making.

As Table 8 shows, average precision across the entire test corpus was 93.32 percent. The lowest precision value was 75% was recorded for the "Rubin" article where only one split out of four was erroneous.
Table 8

Sentence Split Precision

<table>
<thead>
<tr>
<th>Test corpus</th>
<th>Total splits</th>
<th>Correct splits</th>
<th>Botched splits</th>
<th>Bad tag splits</th>
<th>Spurious splits</th>
<th>Precision (percent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Britain</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Cosmic</td>
<td>17</td>
<td>16</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>94</td>
</tr>
<tr>
<td>Degas</td>
<td>22</td>
<td>18</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>82</td>
</tr>
<tr>
<td>Dole</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Drought</td>
<td>9</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>78</td>
</tr>
<tr>
<td>Everglades</td>
<td>9</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Film</td>
<td>11</td>
<td>9</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>82</td>
</tr>
<tr>
<td>Flag</td>
<td>22</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>91</td>
</tr>
<tr>
<td>Govs</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Gunsales</td>
<td>9</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>78</td>
</tr>
<tr>
<td>Hubble</td>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Italy</td>
<td>12</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Japanese</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Kosovo</td>
<td>8</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Launch</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Merger</td>
<td>7</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>86</td>
</tr>
<tr>
<td>Nireland</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Nuclear</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Oconnor</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Rockets</td>
<td>11</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Rubin</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>75</td>
</tr>
<tr>
<td>Salinas</td>
<td>14</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Submarine</td>
<td>8</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>88</td>
</tr>
<tr>
<td>Whaling</td>
<td>19</td>
<td>17</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>89</td>
</tr>
<tr>
<td>Wolves</td>
<td>20</td>
<td>18</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>90</td>
</tr>
</tbody>
</table>

Average Precision 93.32
We use the term "effectiveness" to measure how well the IE recognizes splits that should be made. Table 9 shows IE effectiveness across the articles in the test corpus. Referring to the table, "valid splits attempted" is the total number of splits made, less the number of splits which should not have been made ("spurious splits"), which are obtained from Table 8. The "number of splits missed" measures the number of splits which, upon manual review of the input article text, should have been made but which the IE did not attempt. The category "bad tag misses" counts the number of split misses that were due to erroneous tags provided by the Brill tagger, such as tagging a verb as a noun, as determined by manual review of the Brill tagger output for the article.

In this context, we define effectiveness to be:

\[
\text{Effectiveness} = \frac{\# \text{ attempted}}{\# \text{ attempted} + (\# \text{ missed} - \# \text{ bad tag misses})}
\]

Thus, for the "Salinas" article, for example, where there were 14 split attempts, with three misses, two of which were attributable to bad tags, the effectiveness was computed to be 93%. We use this measure of effectiveness as we feel it appropriately measures how well the IE detects splits that should be made.

From Table 9, the average effectiveness across the entire test corpus was 81.52%, with a large variation in values, ranging from a low of 53% for the "Doie" article to 100% for several other articles.
Table 9
Sentence Split Effectiveness

<table>
<thead>
<tr>
<th>Test corpus article</th>
<th>Valid splits attempted</th>
<th>Number of splits missed</th>
<th>Bad tag misses</th>
<th>Effectiveness (percent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Britain</td>
<td>10</td>
<td>2</td>
<td>0</td>
<td>83</td>
</tr>
<tr>
<td>Cosmic</td>
<td>17</td>
<td>6</td>
<td>1</td>
<td>77</td>
</tr>
<tr>
<td>Degas</td>
<td>20</td>
<td>8</td>
<td>0</td>
<td>71</td>
</tr>
<tr>
<td>Dole</td>
<td>8</td>
<td>7</td>
<td>0</td>
<td>53</td>
</tr>
<tr>
<td>Drought</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>90</td>
</tr>
<tr>
<td>Everglades</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>90</td>
</tr>
<tr>
<td>Film</td>
<td>11</td>
<td>2</td>
<td>0</td>
<td>85</td>
</tr>
<tr>
<td>Flag</td>
<td>20</td>
<td>5</td>
<td>0</td>
<td>80</td>
</tr>
<tr>
<td>Govs</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>67</td>
</tr>
<tr>
<td>Gunsales</td>
<td>8</td>
<td>5</td>
<td>0</td>
<td>62</td>
</tr>
<tr>
<td>Hubble</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Italy</td>
<td>12</td>
<td>2</td>
<td>0</td>
<td>86</td>
</tr>
<tr>
<td>Japanese</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>75</td>
</tr>
<tr>
<td>Kosovo</td>
<td>8</td>
<td>3</td>
<td>1</td>
<td>80</td>
</tr>
<tr>
<td>Launch</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Merger</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>78</td>
</tr>
<tr>
<td>Nireland</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Nuclear</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Oconnor</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>88</td>
</tr>
<tr>
<td>Rockets</td>
<td>11</td>
<td>3</td>
<td>0</td>
<td>79</td>
</tr>
<tr>
<td>Rubin</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>60</td>
</tr>
<tr>
<td>Salinas</td>
<td>14</td>
<td>3</td>
<td>2</td>
<td>93</td>
</tr>
<tr>
<td>Submarine</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>88</td>
</tr>
<tr>
<td>Whaling</td>
<td>18</td>
<td>8</td>
<td>3</td>
<td>78</td>
</tr>
<tr>
<td>Wolves</td>
<td>18</td>
<td>6</td>
<td>0</td>
<td>75</td>
</tr>
</tbody>
</table>

Average Effectiveness 81.52
7. CONCLUSIONS

7.1 Concept Evaluation

Based on the experimental results, it is clear that sentence splitting in order to facilitate downstream NLP components is feasible. The research conducted and the heuristic rules developed demonstrate that POS tags, punctuation, and the words themselves constitute rich sources of information for determining whether and how to split sentences without affecting their meanings. This research also demonstrates, through implementing and testing actual heuristic rules, that sentence splitting heuristics can be found that are both effective and operate with precision.

7.2 Limitations of Approach

This research has also demonstrated that the IE, as implemented, possesses certain limitations. First, at the lowest level, the types of splits that the IE attempts do not completely cover the field of all meaning-preserving splits that can ideally be made. Sentences within direct quotes and "that"-clauses fall within this category. Also, while the rules implemented to date work well, additional experimentation to refine them can be expected to result in even better performance.
Second, it is clear that verb form agreement for split off sentences is a complex task in its own right, which, if solved, would enhance IE performance. Although the IE included some rules for propagating infinitives, modals, and complex verb constructions, and while these heuristics worked well in some cases (as in the example shown in Section 4.2.8), there were other cases in which the rules produced, at best, awkward results. For example, the IE rule for comma-"with"-gerund splits that always converts the gerund to the past tense is not appropriate in many situations. Clearly, improvements in this area would improve overall IE performance.

Third, and most importantly, this research also demonstrated that the manual development of heuristic rules of this type is an arduous task, fraught with danger. For example, over the course of IE development, the C function that implemented the coordination splitting heuristic rule grew to include over 500 lines of code. Moreover, even within the limited rule set involved in the IE, the rules reached a level of complexity where subtle interactions among the rules occurred (through their impact on tagging and splitting) whenever any one rule was altered, however slightly. This is the well known problem of rule conflict. One approach for solving this problem is in the automated development of rules, a did Brill in the development of his tagger [2].
7.3 Future Research

The limitations described above point the way towards future research in this area. Again beginning at the lowest level, additional research can be conducted with the aim of further refining the existing rule set, or extending it to cover additional sentence splitting situations, such as "that"-clauses and sentences within direct quotes.

Second, research into verb form agreement would be expected to enhance overall performance. This can be expected to be a difficult area, since clauses in real world articles often contain multiple verbs not in the same form, and rules will need to be developed to determine which of the verb phrases within a split off clause must agree with which verb phrase in an antecedent clause. In this regard, also, the effort should include the resolution of the Brill tagger's "VBG" tag, which is used for both gerunds and present participles, and which the IE currently assumes for convenience always to be a gerund.

Finally, research should be conducted concerning the possibilities of automating the rule development process, perhaps in a manner similar to the development of the Brill tagger rule set, as described in Section 3.1. However, it is not at all certain that the complex rules needed for the IE can be developed in such a manner, as the rules developed to date often depend on many different types of conditions that must be satisfied in parts of the sentence far removed from the candidate split location. This is unlike the POS tagger rules, which operate only upon information of a limited type and
only within a small local window around the word being tagged at the moment. Only further research will determine the answer to this question.
Appendix A

"Salinas" Article Text, Tag, and Output Files
By TIM GOLDEN

-- Eager to do business with Raul Salinas de Gortari, a brother of the former President of Mexico, Citibank executives ignored some of the bank's own safeguards against the laundering of illicit funds, a congressional report says.

As the bankers took in millions of dollars from Salinas, they never asked for standard information on his financial background and made virtually no effort to verify the source of the money, the report said.

After Salinas was arrested for murder in 1995 and lawyers for the bank had begun monitoring his accounts, his personal banker in New York quietly advised Salinas' wife to move the money elsewhere, apparently without the consent of the legal department.

And even when Citibank finally warned federal officials about Salinas' suspicious transactions, and after Mrs. Salinas had been arrested as well, the bank failed to tell the government about the network of foreign shell companies and offshore accounts that the bank had set up to shield the Salinas fortune.

The disclosures, in a report by the General Accounting Office, the investigative arm of Congress, represent the most detailed accounting yet of how Salinas used a special Citibank unit reserved for the wealthiest customers to move up to $100 million out of Mexico secretly.

Salinas and the bank have repeatedly denied wrongdoing. Whether any U.S. laws were broken remains unclear.

Federal prosecutors in Manhattan are continuing to investigate the possibility that Citibank, a unit of Citigroup Inc., illegally laundered the money. Officials at the Justice Department and the Federal Reserve Bank refused to discuss the case with
The investigation underscores why federal regulators are stepping up scrutiny of the high-end services called private banking and why they have begun to propose steps for banks to track customers' financial movements and backgrounds more closely.

"We determined in the Salinas scenario that Citibank's voluntary controls did not work," the investigators wrote. "Citibank, while violating only one aspect of its then policies, facilitated a money-managing system that disguised the origin, destination and beneficial owner of the funds involved."

The study was issued weeks after Swiss authorities had moved to confiscate $114 million from Salinas, asserting that the funds were protection money paid by drug traffickers. Mexican officials also recently announced that they had frozen an additional $119 million in a maze of other accounts that Salinas controlled.

In a statement Thursday, the bank said the report "contains errors of fact and interpretation."

A spokesman for the bank, Richard Howe, would neither detail the errors nor address any specific issues in the case.

The report also noted that officials in the Office of the Comptroller of the Currency, which has not investigated the case, believed that the civil bank-secrecy statute had probably not been violated. The law says prosecutors can only prove that Salinas or the bank violated money-laundering statutes if they first show that the money was from an unlawful source. The prosecutors would then have to demonstrate that the bank knew or should have known that the money was illicit.

Law-enforcement officials familiar with the case said their principal challenge had been to amass enough evidence to prove in a criminal trial that Salinas had earned his money by one of the handful of crimes, like drug trafficking, that are covered abroad under the federal money-laundering statutes.

Swiss investigators, who faced a much lower evidentiary threshold to confiscate Salinas' deposits there, based their case in part on statements by convicted drug traffickers imprisoned in the United States. Although U.S. officials have dismissed some of those potential witnesses as unreliable, they said the prosecutors in New York had interviewed others whom they considered credible.

It is not clear whether any figures in the new group...
have gone before the grand jury in the case.

Testifying as a government witness in an earlier money-laundering trial, the Citibank executive who worked on Salinas' account, Amy Elliott, suggested that the bank's "know your customer" policies were fundamental to its efforts to avoid easing the way for illegal transactions.

That case, in 1994, involved a former gasoline-station attendant from northern Mexico who masqueraded as an upstanding executive to launder huge sums of money for one of the biggest drug traffickers.

Although Ms. Elliott stated that she and her colleagues had evaluated their potential customers carefully, checking their business dealings and credit backgrounds and visiting them up to 12 times a year, the congressional investigators found that she worked quite differently with Salinas.

"Citibank made no attempt to investigate Salinas' background before accepting him" as a customer in 1992, the report states.

It notes that Ms. Elliott, still an employee in good standing, filed neither a standard financial profile nor a financial background check. Nor, as bank policy required, did she ask to have the requirement for a profile waived.

Although Salinas had never been formally accused of wrongdoing, rumors of possible corruption were widespread in Mexican financial circles. Nonetheless, Ms. Elliott later told prosecutors in a deposition, she thought of her new customer as something akin to "a Rockefeller."

Ms. Elliott said in her statement that she believed that much of Salinas' money came from the sale of a construction company. But as the deposits flowed in, generating $1.1 million in fees, bank officials never learned the company's name.

Under the system Ms. Elliott devised, Salinas' third wife, Paulina Castanon, would pick up cashier's checks in pesos at Mexican banks, carry them to the Citibank subsidiary in Mexico City, convert them to dollars and wire them to New York, using the name Patricia Rios, a first name that she did not use combined with her mother's maiden name.

Congressional investigators, like Swiss detectives before them, were unable to establish the source of the pesos that Salinas kept in Mexican banks. But if he had received drug bribes, they would have almost certainly been paid in American dollars, the currency in which drugs are generally sold.
From Mexico, Salinas' money went to a Citibank account in New York that disguised its origins by mixing it with deposits from other banks and customers. The funds were then sent to Swiss and British accounts in the name of a Cayman Islands shell corporation, Trocca Ltd., that was run by three other offshore shell companies but secretly controlled by Salinas.

The congressional report states that after Salinas' arrest in February 1995, Ms. Elliott filed a brief financial profile and went to Mexico City without the knowledge or consent of the bank's legal representative to try to persuade Mrs. Salinas to close her husband's Citibank accounts.

Mrs. Salinas finally did try to consolidate his holdings, but was arrested in Switzerland that November. Only then did Citibank file a criminal referral form, the congressional report states, but it neglected to mention Trocca or the Swiss or British accounts.
Eager/JJ to/TO do/VB business/NN with/IN Raul/NNP Salinas/NNP de/FW Gortari/NNP ,/ a/DT brother/NN of/IN the/DT former/JJ President/JJ of/IN Mexico/NNP ,/ Citibank/NNP executives/NNS ignored/VBD some/DT of/IN the/DT bank/NN 's/POS own/JJ safeguards/NNS against/IN the/DT laundering/NN of/IN illicit/JJ funds/NNS ,/ a/DT congressional/JJ report/NN says/NNS ./ As/IN the/DT bankers/NNS took/VBD in/IN millions/NNS of/IN dollars/NNS from/IN Salinas/NNP ,/ they/PRP never/RB asked/VBD for/IN standard/JJ information/NN on/IN his/PRP$ financial/JJ background/NN and/CC made/VBN virtually/RB no/DT effort/NN to/TO verify/VB the/DT source/NN of/IN the/DT money/NN ,/ the/DT report/NN said/VBD ./ After/IN Salinas/NNP was/VBD arrested/VBN for/IN murder/NN in/IN 1995/CD and/CC lawyers/NNS for/IN the/DT bank/NN had/VBD begun/VBN monitoring/VBG his/PRP$ accounts/NNS ,/ his/PRP$ personal/JJ banker/NN in/IN New/NP York/NNP quietly/RB advised/VBD Salinas/NNP '/POS wife/NN to/TO move/VB the/DT money/NN elsewhere/RR ,/ apparently/RB without/IN the/DT consent/NN of/IN the/DT legal/JJ department/NN ./ And/CC even/RB when/WRB Citibank/NNP finally/RB warned/VBD federal/JJ officials/NNS about/IN Salinas/NNP '/POS suspicious/JJ transactions/NNS ,/ and/CC after/IN Mrs/NNP ./ Salinas/NNP had/VBD been/VBN arrested/VBN as/IN well/RB ,/ the/DT bank/NN failed/VBD to/TO tell/VB the/DT government/NN about/IN the/DT network/NN of/IN foreign/JJ shell/NN companies/NNS and/CC offshore/JJ accounts/NNS that/IN the/DT bank/NN had/VBD set/VP up/VB to/TO shield/VB the/DT Salinas/NNP fortune/NN ./. The/DT disclosures/NNS ,/ in/IN a/DT report/NN by/IN the/DT General/JJ NNP Accounting/NNP Office/NNP ,/ the/DT investigative/JJ arm/NN of/IN Congress/NNP ,/ represent/VB the/DT most/RBS detailed/VBN accounting/NN yet/RB of/IN how/WRB Salinas/NNP used/VBD a/DT special/JJ Citibank/NNP unit/NN reserved/VBN for/IN the/DT wealthiest/JJS customers/NNS to/TO move/VB up/in/IN to/TO $$ 100/CD million/CD out/IN of/IN Mexico/NNP secretly/RB ./ Salinas/NNP and/CC the/DT bank/NN have/VBP repeatedly/RB denied/VBN wrongdoing/NN ./ Whether/IN any/DT U/NNP ./ S/NPP ./ laws/NNS were/VBD broken/VBN remains/VBZ unclear/JJ ./ Federal/JJ prosecutors/NNS in/IN Manhattan/NNP are/VBP continuing/VBG to/TO investigate/VB the/DT possibility/NN that/IN Citibank/NNP ,/ a/DT unit/NN of/IN Citigroup/NNP Inc/NNP ./ ./ ,/ illegally/RB laundered/VBN the/DT money/NN ./ Officials/NNS at/IN the/DT Justice/NN Department/NNP and/CC the/DT Federal/NNP Reserve/NNP Bank/NNP refused/VBD to/TO discuss/VB the/DT case/NN with/IN congressional/JJ investigators/NNS ./ The/DT investigation/NN underscores/VBZ why/WRB federal/JJ regulators/NNS are/VBP stepping/VBG up/in/IN scrutiny/NN of/IN the/DT high-end/JJ services/NNS called/VBD private/JJ banking/NN and/CC why/WRB they/PRP have/VBP begun/VBN to/TO propose/VB steps/NNS for/IN banks/NNS to/TO track/VB customers/NNS '/POS financial/JJ movements/NNS and/CC backgrounds/NNS more/RBR closely/RB ./ " We/PRP determined/VBD in/IN the/DT Salinas/NNP scenario/NN that/IN Citibank/NNP 's/POS voluntary/JJ controls/NNS did/VBD not/RB work/VB ./ " the/DT investigators/NNS wrote/VBD ./ " Citibank/NNP ,/ while/IN violating/VBG only/RB one/CD aspect/NN of/IN its/PRP$ then/JJ policies/NNS ./ ,/ facilitated/VBN a/DT money-managing/JJ system/NN that/WDT disguised/VBD the/DT origin/NN ./ ,/ destination/NN and/CC
from IN northern/JJ Mexico/NNP who/VP masqueraded/VBD as/IN an/DT upstanding/JJ executive/NNS to/TO launder/VB huge/JJ sums/NNS of/IN money/NNS for/IN one/CD of/IN the/DT biggest/JJS drug/NNS traffickers/NNS /. However/IN Ms/NNP ./ Elliott/NNP stated/VBD that/IN she/PRP and/CC her/PRP$ colleagues/NNS had/VBD evaluated/VBN their/PRP$ potential/JJ customers/NNS carefully/RB ,/ checking/VBG their/PRP$ business/NN dealings/NNS and/CC credit/NN backgrounds/NNS and/CC visiting/VBG them/PRP up/IN to/TO 12/CD times/VBZ a/DT year/NN ./, the/DT congressional/JJ investigators/NNS found/VBD that/IN she/PRP worked/VBD quite/RB differently/RB with/IN Salinas/NNP ./) " / Citibank/NNP made/VBD no/DT attempt/NN to/TO investigate/VB Salinas/NNP ' /POS background/NN before/IN accepting/VBG him/PRP '" as/IN a/DT customer/NN in/IN 1992/CD ./, the/DT report/NN states/NNS ./. It/PRP notes/VBE that/DT Ms/NNP ./ Elliott/NNP ,/ still/RR an/DT employee/NN in/IN good/JJ standing/NN ./, filed/VBD neither/DT a/DT standard/JJ financial/JJ profile/NN nor/CC a/DT financial/JJ background/NN check/NN ./. Nor/CC ./, as/IN bank/NN policy/NN required/VBN ./, did/VBD she/PRP ask/VBE to/TO have/VB the/DT requirement/NN for/IN a/DT profile/NN waived/VBN ./. Although/IN Salinas/NNP had/VBD never/RB been/VBN formally/RB accused/VBN of/IN wrongdoing/NN ./, rumors/VBZ of/IN possible/JJ corruption/NN were/VBD widespread/JJ in/IN Mexican/JJ financial/JJ circles/NNS ./. Nonetheless/RB ./, Ms/NNP ./ Elliott/NNP later/RB told/VBD prosecutors/NNS in/IN a/DT deposition/NN ./, she/PRP thought/VBD of/IN her/PRP$ new/JJ customer/NN as/IN something/NN akin/JJ to/TO " / a/DT Rockefeller/NNP ./." ./ " Ms/NNP ./ Elliott/NNP said/VBD in/IN her/PRP$ statement/NN that/IN she/PRP believed/VBD that/IN much/NN of/IN Salinas/NNP ' /POS money/NN came/VBD from/IN the/DT sale/NN of/IN a/DT construction/NN company/NN ./. But/CC as/IN the/DT deposits/NNS flowed/VBD in/IN ./, generating/VBG $/S 1/CD ./. 1/CD million/CD in/IN fees/NNS ./, bank/NN officials/NNS never/RB learned/VBD the/DT company/NN ' s/POS name/NN ./. Under/IN the/DT system/NN Ms/NNP ./ Elliott/NNP devised/VBD ./, Salinas/NNP ' /POS third/JJ wife/NN ./, Paulina/NNP Castanon/NNP ./, would/MD pick/VB up/IN cashier/NN ' s/POS checks/NNS in/IN pesos/NNS at/IN Mexican/JJ banks/NNS ./, carry/VB them/PRP to/TO the/DT Citibank/NNP subsidiary/NN in/IN Mexico/NNP City/NNP ./, convert/VB them/PRP to/TO dollars/NNS and/CC wire/NN them/PRP to/TO New/NNP York/NNP ./, using/VBG the/DT name/NN Patricia/NNP Rios/NNP ./, a/DT first/JJ name/NN that/IN she/PRP did/VBD not/RB use/VB combined/VBN with/IN her/PRP$ mother/NN ' s/POS maiden/NN name/NN ./. Congressional/JJ investigators/NNS ./, like/IN Swiss/JJ detectives/NNS before/IN them/PRP ./, were/VBD unable/JJ to/TO establish/VB the/DT source/NN of/IN the/DT pesos/NNS that/IN Salinas/NNP kept/VBD in/IN Mexican/JJ banks/NNS ./. But/CC if/IN he/PRP had/VBD received/VBN drug/NN bribes/NNS ./, they/PRP would/MD have/VB almost/RB certain/RB be/VBN paid/VBN in/IN American/JJ dollars/NNS ./, the/DT currency/NN in/IN which/WDT drugs/NNS are/VBP generally/RB sold/VBN ./. From/IN Mexico/NNP ./, Salinas/NNP ' /POS money/NN went/VBD to/TO a/DT Citibank/NNP account/NN in/IN New/NNP York/NNP that/WDT disguised/VBD its/PRP$ origins/NNS by/IN mixing/VBG it/PRP with/IN deposits/NNS from/IN other/JJ banks/NNS and/CC customers/NNS ./. The/DT funds/NNS were/VBD then/RB sent/VBN to/TO Swiss/JJ
and/CC British/JJ accounts/NNS in/IN the/DT name/NN of/IN a/DT Cayman/NNP Islands/NNPS shell/NN corporation/NN ./, Trocca/NNP Ltd/NNP ./.,/ that/WDT was/VBD run/VBN by/IN three/CD other/JJ offshore/JJ shell/NN companies/NNS but/CC secretly/RB controlled/VBN by/IN Salinas/NNP ./. The/DT congressional/JJ report/NN states/NNS that/IN after/IN Salinas/NNP 's/POS arrest/NN in/IN February/NNP 1995/CD ./, Ms/NNP ./. Elliott/NNP filed/VBD a/DT brief/JJ financial/JJ profile/NN and/CC went/VBD to/TO Mexico/NNP City/NNP without/IN the/DT knowledge/NN or/CC consent/NN of/IN the/DT bank/NN 's/POS legal/JJ representative/NN to/TO try/VB to/TO persuade/VB Mrs/NNP ./. Salinas/NNP to/TO close/VB her/PRP$ husband/NN 's/POS Citibank/NNP accounts/NNS ./. Mrs/NNP ./. Salinas/NNP finally/RB did/VBD try/VB to/TO consolidate/VB his/PRP$ holdings/NNS ./, but/CC was/VBD arrested/VBN in/IN Switzerland/NNP that/DT November/NNP ./. Only/RB then/RB did/VBD Citibank/NNP file/VB a/DT criminal/JJ referral/NN form/NN ./, the/DT congressional/JJ report/NN states/NNS ./, but/CC it/PRP neglected/VBD to/TO mention/VB Trocca/NNP or/CC the/DT Swiss/JJ or/CC British/JJ accounts/NNS ./.
DEMETRIOS G. GLINOS

Master's Thesis: "An Intelligent Editor for Natural Language Processing of Unrestricted Text"

Output file for thesis program "nlp.exe"

Input file name > 120498salinas.inp

Eager/JJ to/TO do/VB business/NN with/IN Raul/NNP Salinas/NNP de/FW Gortari/NNP ,/cf a/DT brother/NN of/IN the/DT former/JJ President/NNP of/IN Mexico/NNP [,] /zCC Citibank/NNP executives/NNP ignored/VBD some/DT of/IN the/DT bank/NN 's/POS own/JJ safeguards/NNN against/IN the/DT laundering/NN of/IN illicit/JJ funds/NNS ,/capo a/DT congressional/JJ report/NN says/mNNN ./.

The/DT bankers/NNS took/VBD in/IN millions/NNS of/IN dollars/NNS from/IN Salinas/NNP ,/ccrd ./p-sc

They/PRP never/RB asked/VBD for/IN standard/JJ information/NN on/IN his/PRP$ financial/JJ background/NN and/CC made/VBN virtually/RB no/DT effort/NN to/TO verify/VB the/DT source/NN of/IN the/DT money/NN ,/cqi the/DT report/NN said/mVBD ./.

Salinas/NNP was/VBD arrested/VBN for/IN murder/NN in/IN 1995/CD ./p-sc

Lawyers/NNS for/IN the/DT bank/NN had/VBD begun/VBN monitoring/VBG his/PRP$ accounts/NNS ,/p-cj

His/PRP$ personal/JJ banker/NN in/IN New/NNP York/NNP quietly/RB advised/VBD Salinas/NNP 's/POS wife/NN to/TO move/VB the/DT money/NN elsewhere/RB [,] /zCC apparently/RB without/IN the/DT consent/NN of/IN the/DT legal/JJ department/NN ./.

Citibank/NNP finally/RB warned/VBD federal/JJ officials/NNS about/IN Salinas/NNP 's/POS suspicious/JJ transactions/NNS ,/cf ./p-sc
Mrs/NNP ./pd Salinas/NNP had/VBD been/VBN arrested/VBN as/IN well/RB ,/ccrd ./p-sc

The/DT bank/NN failed/VBD to/TO tell/VB the/DT government/NN about/IN the/DT network/NN of/IN foreign/JJ shell/NN companies/NNS and/CC offshore/JJ accounts/NNS that/IN the/DT bank/NN had/VBD set/VBN up/VB to/TO shield/VB the/DT Salinas/NNP fortune/NN ./.  

The/DT disclosures/NNS ,/ccpo in/prep a/DT report/NN by/IN the/DT General/nnpa ,/ccpo Accounting/NNP Office/NNP ,/ccpc ,/cf the/DT investigative/JJ arm/NN of/IN Congress/NNP ,/cccrd represent/VB the/DT most/RBS detailed/VBN accounting/NN yet/RB of/IN how/WRB Salinas/NNP used/VBD a/DT special/JJ Citibank/NNP unit/NN reserved/VBN for/IN the/DT wealthiest/JJS customers/NNS to/TO move/VB up/IN to/TO $/$ 100/CD million/CD out/IN of/IN Mexico/NNP secretly/RB ./.  

Salinas/NNP and/CC the/DT bank/NN have/VBP repeatedly/RB denied/VBN wrongdoing/NN ./.  

Whether/prep any/DT U.S./NNP laws/NNS were/VBD broken/VBN remains/VBZ unclear/JJ ./.  

Federal/JJ prosecutors/NNS in/IN Manhattan/NNP are/VBP continuing/VBG to/TO investigate/VB the/DT possibility/NN that/IN Citibank/NNP ,/ccpo a/DT unit/NN of/IN Citigroup/NNP Inc/NNP ./pd ,/ccpc illegally/RB laundered/VBN the/DT money/NN ./.

Officials/NNS at/IN the/DT Justice/NNP Department/NNP and/CC the/DT Federal/NNP Reserve/NNP Bank/NNP refused/VBD to/TO discuss/VB the/DT case/NN with/IN congressional/JJ investigators/NNS ./.  

The/DT investigation/NN underscores/VBZ why/WRB federal/JJ regulators/NNS are/VBP stepping/VBG up/IN scrutiny/NN of/IN the/DT high-end/JJ services/NNS called/VBD private/JJ banking/NN ./p-sc
They/PRP have/VBP begun/VBN to/TO propose/VB steps/NNS for/IN banks/NNS to/TO track/VB customers/NNS '/POS financial/JJ movements/NNS and/zCC backgrounds/NNS more/RBR closely/RB ./.

"/go We/PRP determined/qVBD in/qIN the/qDT Salinas/qNNP scenario/qNN that/qIN Citibank/qNNP 's/qPOS voluntary/qJJ controls/qNNNS did/qVBD not/qRB work/qVB ,/q, "/qc the/DT investigators/NNS wrote/VBD ./.

"/go Citibank/qNNP ,/qccrd while/qIN violating/qVBG only/qRB one/qCD aspect/qNN of/qIN its/qPRP$ then/qJJ policies/qNNNS ,/qccrd facilitated/qVBN a/qDT money-managing/qJJ system/qNNN that/qWDT disguised/qVBD the/qDT origin/qNNN [,]/qzCC destination/qNNN and/qzCC beneficial/qJJ owner/qNN of/qIN the/qDT funds/qNNNS involved/qVBN ./q. "/qc

The/DT study/NN was/VBD issued/VBN weeks/NNS after/IN Swiss/JJ authorities/NNS had/VBD moved/VBN to/TO confiscate/VB $/$ 114/CD million/CD from/IN Salinas/NNP ,/cf asserting/VBG that/IN the/DT funds/NNS were/VBD protection/NN money/NN paid/VBN by/IN drug/NN traffickers/NNS ./.

Mexican/JJ officials/NNS also/RB recently/RB announced/VBD that/IN they/PRP had/VBD frozen/VBN an/DT additional/JJ $/$ 119/CD million/CD in/IN a/DT maze/NN of/IN other/JJ accounts/NNS that/IN Salinas/NNP controlled/VBD ./.

In/prep a/DT statement/NN Thursday/NNP ,/qccrd the/DT bank/NN said/mVBD the/DT report/NN "/qc contains/qVBDZ errors/qNNNS of/qIN fact/qNN and/qzCC interpretation/qNNN ./q. "/qc

A/DT spokesman/NN for/IN the/DT bank/NN ,/capo Richard/NNP Howe/NNP ,/capc would/MD neither/DT detail/NN the/DT errors/NNS nor/CC address/VB any/DT specific/JJ issues/NNS in/IN the/DT case/NN ./.
The/DT report/NN also/RB noted/mVBD that/IN officials/NNS in/IN the/DT Office/NNP of/IN the/DT Comptroller/NNP of/IN the/DT Currency/NN ,/crel ,/p-wh ,/cf believed/VBD that/IN the/DT civil/JJ bank-secrecy/JJ statute/NN had/VBD probably/RB not/RB been/VBN violated/VBN ./.  

[prior]/nnwh has/VBZ not/RB investigated/VBN the/DT case/NN ,/x-wh

The/DT law/NN says/mVBJ prosecutors/NNS can/MD only/RB prove/VB that/IN Salinas/NNP or/CC the/DT bank/NN violated/VBD money-laundering/NN statutes/NNS if/IN they/PRP first/JJ show/NN that/IN the/DT money/NN was/VBD from/IN an/DT unlawful/JJ source/NN ./.  

The/DT prosecutors/NNS would/MD then/RB have/VB to/TO demonstrate/VB that/IN the/DT bank/NN knew/VBD or/CC should/MD have/VB known/VBN that/IN the/DT money/NN was/VBD illicit/JJ ./.  

Law-enforcement/JJ officials/NNS familiar/JJ with/IN the/DT case/NN said/mVBD their/PRP$ principal/JJ challenge/NN had/VBD been/VBN to/TO amass/VB enough/JJ evidence/NN to/TO prove/VB in/IN a/DT criminal/JJ trial/NN that/IN Salinas/NNP had/VBD earned/VBN his/PRP$ money/NN by/IN one/CD of/IN the/DT handful/NN of/IN crimes/NNS ,/capo like/prep drug/NN trafficking/NN ,/capc that/WDT are/VBP covered/VBN abroad/RB under/IN the/DT federal/JJ money-laundering/NN statutes/NNS ./.  

Swiss/JJ investigators/NNS ,/crel who/WP faced/VBD a/DT much/RB lower/JJR evidentiary/JJ threshold/NN to/TO confiscate/VB Salinas/NNP '/'pos deposits/NNS there/EX ,/cf based/VBD their/PRP$ case/NN in/IN part/NN on/IN statements/NNS by/IN convicted/VBN drug/NN traffickers/NNS imprisoned/VBN in/IN the/DT United/NNP States/NNPS ./.
U.S./NNP officials/NNS have/VBP dismissed/VBN some/DT of/IN those/DT potential/JJ witnesses/NNS as/IN unreliable/JJ ,/p-cj

They/PRP said/mVBD the/DT prosecutors/NNS in/IN New/NNP York/NNP had/VBD interviewed/VBN others/NNS whom/WP they/PRP considered/VBD credible/JJ ./.

It/PRP is/VBZ not/RB clear/JJ whether/IN any/DT figures/NNS in/IN the/DT new/JJ group/NN have/VBP gone/VBN before/IN the/DT grand/JJ jury/NN in/IN the/DT case/NN ./.

Testifying/VBG as/IN a/DT government/NN witness/NN in/IN an/DT earlier/JJR money-laundering/NN trial/NN ,/cf the/DT Citibank/NNP executive/NN who/WP worked/VBD on/IN Salinas/NNP '/POS account/NN ,/capo Amy/NNP Elliott/NNP ,/capc suggested/VBD that/IN the/DT bank/NN 's/POS "'/go know/qVB your/qPRP$ customer/qNN "/qc policies/NNS were/VBD fundamental/JJ to/TO its/PRP$ efforts/NNS to/TO avoid/VB easing/VBG the/DT way/NN for/IN illegal/JJ transactions/NNS ./.

That/DT case/NN ,/cf in/prep 1994/CD ,/cf involved/VBD a/DT former/JJ gasoline-station/JJ attendant/NN from/IN northern/JJ Mexico/NNP who/WP masqueraded/VBD as/IN an/DT upstanding/JJ executive/NN to/TO launder/VB huge/JJ sums/NNS of/IN money/NN for/IN one/CD of/IN the/DT biggest/JJS drug/NN traffickers/NNS ./.

Ms/NNP ./pd Elliott/NNP stated/mVBD that/IN she/PRP and/CC her/PRP$ colleagues/NNS had/VBD evaluated/VBN their/PRP$ potential/JJ customers/NNS carefully/RB ,/ccrd checking/VBG their/PRP$ business/NN dealings/NNS and/CC credit/NN backgrounds/NNS and/CC visiting/VBG them/PRP up/IN to/TO 12/CD times/VBZ a/DT year/NN ,/p-cj

The/DT congressional/JJ investigators/NNS found/VBD that/IN she/PRP worked/VBD quite/RB differently/RB with/IN Salinas/NNP ./.

"/go Citibank/qNNP made/qVBD no/qDT attempt/qNN to/qTO investigate/qVB Salinas/qNNP '/qPOS background/qNN before/qIN
accepting/qVBG him/qPRP "/qc as/IN a/DT customer/NN in/IN 1992/CD ,/cqi the/DT report/NN states/mNNS ./.

It/PRP notes/mVBZ that/DT Ms/NNP ./pd Elliott/NNP ,/capo still/RB an/DT employee/NN in/IN good/JJ standing/NN ,/capc filed/VBD neither/DT a/DT standard/JJ financial/JJ profile/NN nor/CC a/DT financial/JJ background/NN check/NN ./.

Nor/CC ,/cf as/scnj bank/NN policy/NN required/VBN ,/csub did/VBD she/PRP ask/VBP to/TO have/VB the/DT requirement/NN for/IN a/DT profile/NN waived/VBN ./.

Salinas/NNP had/VBD never/RB been/VBN formally/RB accused/VBN of/IN wrongdoing/NN ,/p-cj Rumors/VBZ of/IN possible/JJ corruption/NN were/VBD widespread/JJ in/IN Mexican/JJ financial/JJ circles/NNS ./.

Ms/NNP ./pd Elliott/NNP later/RB told/mVBD prosecutors/NNS in/IN a/DT deposition/NN ,/cf she/PRP thought/VBD of/IN her/PRP$ new/JJ customer/NN as/IN something/NN akin/JJ to/TO "/qc a/qDT Rockefeller/qNNP ./q. "/qc

Ms/NNP ./pd Elliott/NNP said/mVBD in/IN her/PRP$ statement/NN that/IN she/PRP believed/VBD that/IN much/NN of/IN Salinas/NNP '/POS money/NN came/VBD from/IN the/DT sale/NN of/IN a/DT construction/NN company/NN ./.

But/CC as/IN the/DT deposits/NNS flowed/VBD in/IN ,/cf generating/VBG "/$ 1.1/CD million/CD in/IN fees/NNS ,/cf bank/NN officials/NNS never/RB learned/VBD the/DT company/NN 's/POS name/NN ./.
Under prep the DT system NN Ms NNP ./pd Elliott NNP devised VBD ,/cf Salinas NNP '/POS third JJ wife NN ,/capo Paulina NNP Castanon NNP ,/capc would MD pick VB up IN cashier NN s/POS checks NNS in IN pesos NNS at IN Mexican JJ banks NNS ,/ccrd ./p-vp

[Subj] p-vp would MD carry VB them PRP to TO the DT Citibank NNP subsidiary NN in IN Mexico NNP City NNP ,/ccrd ./p-vp

[Subj] p-vp would MD convert VB them PRP to TO dollars NNS and CC wire NN them PRP to TO New NNP York NNP ,/capo using VBG the DT name NN Patricia NNP Rios NNP ,/capc a DT first JJ name NN that IN she PRP did VBD not RB use VB combined VBN with IN her PRP$ mother NN s/POS maiden NN name NN ./.

Congressional JJ investigators NNS ,/capo like prep Swiss JJ detectives NNS before IN them PRP ,/capc were VBD unable JJ to TO establish VB the DT source NN of IN the DT pesos NNS that IN Salinas NNP kept VBD in IN Mexican JJ banks NNS ./.

But CC if IN he PRP had VBD received VBN drug NN bribes NNS ,/cf they PRP would MD have VB almost RB certainly RB been VBN in IN American JJ dollars NNS ,/cf the DT currency NNS in IN which WDT drugs NNS are VBP generally RB sold VBN ./.

From prep Mexico NNP [,] /zCC Salinas NNP '/POS money NN went VBD to TO a DT Citibank NNP account NN in IN New NNP York NNP that WDT disguised VBD its PRP$ origins NNS by IN mixing VBG it PRP with IN deposits NNS from IN other JJ banks NNS and zCC customers NNS ./.

The DT funds NNS were VBD then RB sent VBN to TO Swiss JJ and zCC British JJ accounts NNS in IN the DT name NN of IN a DT Cayman NNP Islands NNP$ shell NN corporation NN ,/capo Trocca NNP Ltd NNP ./pd ,/capc that WDT was VBD run VBN by IN three CD other JJ offshore JJ shell NN companies NNS but CC secretly RB controlled VBN by IN Salinas NNP ./.
The DT congressional JJ report NN states mNNS that IN after IN Salinas NNP / POS arrest NN in IN February NNP 1995 CD , / ccrd Ms NNP . / pd Elliott NNP filed VBD a DT brief JJ financial JJ profile NN and CC went VBD to TO Mexico NNP City NNP without IN the DT knowledge NN or zCC consent NN of IN the DT bank NN 's POS legal JJ representative NN to TO try VB to TO persuade VB Mrs NNP . / pd Salinas NNP to TO close VB her PRP$ husband NN 's POS Citibank NNP accounts NNS . /.

Mrs NNP . / pd Salinas NNP finally RB did VBD try VB to TO consolidate VB his PRP$ holdings NNS , / cf . / p bt

[Subj] / p bt was VBD arrested VBN in IN Switzerland NNP that DT November NNP . /.

Only RB then RB did VBD Citibank NNP file VBD a DT criminal JJ referral NN form NN , / capo the DT congressional JJ report NNP states mNNS , / capc . / p bt

It PRP neglected VBD to TO mention VB Trocca NNP or zCC the DT Swiss JJ or zCC British JJ accounts NNS . /.

Sentence splitting statistics:

Number of input sentences > 41
Minimum input sentence length > 9
Maximum input sentence length > 71
Average input sentence length > 31.10

Number of output sentences > 55
Minimum output sentence length > 7
Maximum output sentence length > 52
Average output sentence length > 23.25

Program editing statistics:

Number of nonterminating periods found > 11
Number of '? ' and '!' before close quote > 0
Number of idioms collocations marked > 0
Number of non splitting CCs commas marked > 11
Number of quotation marks marked > 12
Number of pn appositions w o commas > 1
Number of pn appositions w commas > 13
Number of prep conjunctions resolved > 14
Number of 'that' tokens marked > 0
Number of numbers in text fixed > 1

93
<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of multi-initial identifiers fixed</td>
<td>&gt; 2</td>
</tr>
<tr>
<td>Number of direct quotation commas marked</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Number of indirect quotation commas marked</td>
<td>&gt; 2</td>
</tr>
<tr>
<td>Number of indirect quotation verbs marked</td>
<td>&gt; 14</td>
</tr>
<tr>
<td>Number of adverbial commas marked</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Number of sub. conj. commas marked</td>
<td>&gt; 5</td>
</tr>
<tr>
<td>Number of coordination commas marked</td>
<td>&gt; 14</td>
</tr>
<tr>
<td>Number of wh-word commas marked</td>
<td>&gt; 2</td>
</tr>
<tr>
<td>Number of futile commas marked</td>
<td>&gt; 18</td>
</tr>
<tr>
<td>Number of open &amp; close parentheses marked</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Number of open &amp; close dashes marked</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Number of modals propagated</td>
<td>&gt; 2</td>
</tr>
<tr>
<td>Number of infinitives propagated</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Number of auxiliaries propagated</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Number of comma-which splits</td>
<td>&gt; 1</td>
</tr>
<tr>
<td>Number of comma-prep-which splits</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Number of but-clause splits</td>
<td>&gt; 2</td>
</tr>
<tr>
<td>Number of dash clause splits</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Number of parenthetical clause splits</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Number of comma + 'with' + gerund splits</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Number of subordinate clause splits</td>
<td>&gt; 4</td>
</tr>
<tr>
<td>Number of verb phrase coordination splits</td>
<td>&gt; 2</td>
</tr>
<tr>
<td>Number of sentence coordination splits</td>
<td>&gt; 5</td>
</tr>
</tbody>
</table>

94
Appendix B

"Italy" Article Text, Tag, and Output Files
CASERTA, Italy -- When a fire broke out recently in an attic of the Royal Palace here, the flames were quickly detected and extinguished.

But that accident reignited a simmering controversy over the way the palace is being used that is far more difficult to stamp out.

The Reggia di Caserta, begun in the mid-18th century for Charles III, the Bourbon king of Naples, to rival Versailles, is one of the most imposing palaces in Europe.

President Clinton dined under its gold ceilings during the Group of Seven summit meeting held in Naples in 1994. Film director George Lucas chose its late Baroque marble staircase for a scene in his coming "Stars Wars" movie, "Episode One: The Phantom Menace." The palace has 1,200 rooms (Versailles has 700) and is surrounded by a 250-acre park that includes a 256-foot cascade.

Unlike Versailles, however, the palace in Caserta is not solely a museum. Only 44 rooms are open to the public. Eight hundred rooms are now under military occupation -- specifically, the Italian air force's training school for noncommissioned officers.

The museum director, the Culture Ministry and art lovers want the military out. But the general in command at Caserta and his men would prefer to stay, and they argue that they have no other place to go. Local officials are caught in the middle: They want the military to leave the palace, but they also want them to stay in Caserta, where they provide 300 jobs and spend about $9 million a year.

The emotional, many-sided battle being fought in Caserta, moreover, is decades old, and far from unique. It took nearly 50 years, but the Culture Ministry finally secured an agreement with the Defense and Finance ministries in 1997 to move an officers' club out of the Palazzo Barberini, one of Italy's national galleries, in Rome.
Retired officers are still having genteel lunch here under their 17th-century dining room's ceiling fresco, but the museum is renovating a smaller 1930s villa in the back of the property in the expectation that the officers will move someday.

In Italy, a country overflowing with art treasures and historical monuments, countless buildings have cultural significance. The Italian military owns or occupies some of the finest properties, including a 14th-century convent in Perugia now used by the army as a foreign language school. But police academies, tax inspectors and dentists also work out of historic buildings.

The incident at Caserta, which occurred shortly after the new culture minister, Giovanna Melandri, was appointed, lighted a fire in Rome.

Mrs. Melandri, 36, vowed to expel the military from Caserta and from dozens of other cultural sites. She said the air force had agreed to find a new location by Feb. 2.

"Our patrimony is so rich and enormous, it would be impossible to turn it all into museums," Mrs. Melandri said in an interview in her office. "But we have to look at improper use of historical buildings. In Caserta, we need to find a use for the available space that is more compatible with a museum. A military school is incompatible."

In Caserta, from opposite wings of a palace spread over four courtyards, the soldiers and the museum workers seem as compatible as hostile neighbors in a crowded Neapolitan high rise. Vincenzo Zuccaro, a museum public relations official, indignantly pointed out an 18th-century vault fresco by Mariano Rossi that had to be restored because of water damage Zuccaro said was caused by cadets' showers above.

The museum director, Livio Ricciardi, complained about cooking smells from the school kitchens. But he confessed that he had no plans or proposals on how to use the additional rooms if and when the school moves out.

"We hope they'll leave soon, but when they do, we'll be at a loss," he said. "For one thing, I don't know what rooms they have and what they look like."

Asked why not, he replied, "I've never been invited."

The commander of the school, which was established here in 1948, said that as a soldier, he would obey orders to retreat from the palace.

"Caserta is in our hearts," the commander, Gen. Alessio Santicchi, said. "Of course we would be unhappy to
leave but that is a sentimental discourse. On a practical level, if they told us, leave so we can do this this and this, I would understand. But to be told just to go away, that I do not understand."

Like other officers there, General Santicchi argued that the school spends $800,000 a year to maintain its part of the building, and is a useful tenant. He argued that the fire was detected, and put out, only because his men were living nearby.

"Our fear is that if we leave, nothing will take our place," he said. "It's like seeing a home turn into an abandoned house."

But his case is complicated by the fact that 10 years ago, the air force decided that the palace was getting too small for the school, and began building a $125 million complex eight miles away in Capua, with most of the money coming from the European Economic Community.

It took 10 years to build, and by the time it was close to completion, cuts in the military budget had whittled down enrollment at the school from 2,000, to 900 today. Now, the air force argues, Capua is too large for the school, and would be too costly to run.

Mrs. Melandri said it was the military's problem to find a new site by Feb. 2. Santicchi noted that even if the air force chose Capua, that site was not ready for habitation. "They haven't built the barracks, and that could take another year or two," he said.

Caserta Mayor Luigi Falco, who said he wants to convert other unused military buildings in town for the school, laughed merrily at Mrs. Melandri's February deadline.

"That," he said, "is not a realistic deadline. It's an Italian deadline."
When/W RB a/DT fire/NN broke/VBD out/IN recently/RB in/IN an/DT attic/NN of/IN the/DT Royal/NNP Palace/NNP here/RB ,/ , the/DT flames/NNS were/VBD quickly/RB detected/VBN and/cc extinguished/VBN ./ . But/CC that/DT accident/NN reignited/VBD a/DT simmering/VBG controversy/NN over/IN the/DT way/NN the/DT palace/NN is/VBZ being/VBG used/VBN that/WD T is/VBZ far/RR more/RB difficult/JJ to/TO stamp/VB out/RR ./ . The/DT Reggia/NNP di/NRP Caserta/NNP ,/ , begun/VBN in/IN the/DT mid-18th/~~ century/NN for/IN Charles/NNP III/NNP ,/ , the/DT Bourbon/NNP king/NN of/IN Naples/NNP ,/ , to/TO rival/VB Versailles/NNP ,/ , is/VBZ one/CD of/IN the/DT most/RRBS imposing/VBG palaces/NNS in/IN Europe/NP ./ . President/WNP Clinton/NNP dined/VBD under/IN its/PRP$ gold/NN ceilings/NNS during/IN the/DT Group/NNP of/IN Seven/NP allies/NN meeting/NN held/VBN in/IN Naples/NNP in/IN 1994/CD ./ . Film/NNP director/NN George/NNP Lucas/NNP chose/VBD its/PRP$ late/JJ Baroque/JJ marble/NN staircase/NN for/IN a/DT scene/NN in/IN his/PRP$ coming/NN "/" Stars/NNP Wars/NNPS "/" movie/NN ,/ , "/" Episode/NNP One/CD :/: The/DT Phantom/NNP Menace/NNP ./ . "/" The/DT palace/NN has/VBZ 1/CD ,/ , 200/CD rooms/NNS (( Versailles/NNP has/VBZ 700/CD )/SYM and/cc is/VBZ surrounded/VBN by/IN a/DT 250-acre/JJ park/NN that/WDT includes/VBZ a/DT 256-foot/JJ cascade/NN ./ . Unlike/IN Versailles/NNP ./ , however/RR ./ , the/DT palace/NN in/IN Caserta/NNP is/VBZ not/RB solely/RR a/DT museum/NN ./ . Only/RR 44/CD rooms/NNS are/VBP open/JJ to/TO the/DT public/NN a./. Eight/CD hundred/CD rooms/NNS are/VBP now/RR under/IN military/JJ occupation/NN --: specifically/RR ,/ , the/DT Italian/JJ air/NN force/NN 's/POS training/NN school/NN for/IN noncommissioned/JJ officers/NNS ./ . The/DT museum/NN director/NN ,/ , the/DT Culture/NNP Ministry/NNP and/cc art/NN lovers/NNS want/VBP the/DT military/NN out/RR ./ . But/CC the/DT general/NN in/IN command/NN at/IN Caserta/NNP and/cc his/PRP$ men/NNS would/MD prefer/VB to/TO stay/VB ,/ , and/cc they/PRP argue/VBP that/IN they/PRP have/VBP no/DT other/JJ place/NN to/TO go/VB ./ . Local/JJ officials/NNS are/VBP caught/VBN in/IN the/DT middle/NN :/: They/PRP want/VBP the/DT military/JJ to/TO leave/VB the/DT palace/NN ,/ , but/CC they/PRP also/RR want/VBP them/PRP to/TO stay/VB in/IN Caserta/NNP ,/ , where/WRB they/PRP provide/VBP 300/CD jobs/NNS and/cc spend/VB about/IN RB $/$ 9/CD million/CD a/DT year/NN ./ . The/DT emotional/JJ ,/ , many-sided/JJ battle/NN being/VBG fought/VBN in/IN Caserta/NNP ,/ , moreover/RR ,/ , is/VBZ decades/NNS old/JJ ,/ , and/cc far/RR from/IN unique/NN ./ . It/PRP took/VBD nearly/RR 50/CD years/NNS ,/ , but/CC the/DT Culture/NNP Ministry/NNP finally/RR secured/VBD an/DT agreement/NN with/IN the/DT Defense/NNP and/cc Finance/NNP ministries/NNS in/IN 1997/CD to/TO move/VB an/DT officers/NNS 'POS club/NN out/IN of/IN the/DT Palazzo/NNP Barberini/NNP ,/ , one/CD of/IN Italy/NNP 's/POS national/JJ galleries/NNS ,/ , in/IN Rome/NNP ./ . Retired/JJ officers/NNS are/VBP still/RR having/VBG genteel/JJ lunch/NN here/RR under/IN their/PRP$ 17th-century/JJ dining/NN room/NN 's/POS ceiling/NN fresco/NN ,/ , but/CC the/DT museum/NN is/VBZ renovating/VBG a/DT smaller/JJR 1930s/CD villa/NN in/IN the/DT back/RR of/IN the/DT property/NN in/IN the/DT expectation/NN that/IN the/DT officers/NNS will/MD move/VB someday/RR ./ . In/IN Italy/NNP ./ , a/DT country/NN overflowing/VBG with/IN art/NN treasures/NNS and/cc historical/JJ monuments/NNS ,/,
countless/NJJ buildings/NNS have/VBP cultural/JJ significance/NN /
. The/DT Italian/JJ military/NN owns/VBZ or/CC occupies/VBZ
some/DT of/IN the/DT finest/JJS properties/NNS ,/,
including/VBG a/DT 14th-century/JJ convent/NN in/IN Perugia/NNP
now/RB used/VBD by/IN the/DT army/NN as/IN a/DT foreign/JJ
language/NN school/NN ./. But/CC police/NN academies/NNS ,/,
tax/NN inspectors/NNS and/CC dentists/NNS also/RB work/VB
out/IN of/IN historic/JJ buildings/NNS ./. The/DT incident/NN
at/IN Caserta/NNP ,/, which/WDT occurred/VBD shortly/RB
after/in the/DT new/JJ culture/NN minister/NN ,/, Giovanna/NNP
Melandri/NNP ,/, was/VBD appointed/VBN ,/, lighted/VBD a/DT
fire/NN in/IN Rome/NNP ./. Mrs/NNP ./. Melandri/NNP ,/, 36/CD
/, vowed/VBD to/TO expel/VB the/DT military/NN from/IN
Caserta/NNP and/CC from/IN dozens/NNS of/IN other/JJ
cultural/JJ sites/NNS ./. She/PRP said/VBD the/DT air/NN
force/NN had/VBD agreed/VBN to/TO find/VB a/DT new/JJ
location/NN by/IN Feb/NNP ./. 2/CD ./. "/" Our/PRP$
patrimony/NN is/VBZ so/RB rich/JJ and/CC enormous/JJ ,/, it/PRP
would/MD be/VB impossible/JJ to/TO turn/VB it/PRP all/DT
into/IN museums/NNS ,/, "/" Mrs/NNP ./. Melandri/NNP said/VBD
in/IN an/DT interview/NN in/IN her/PRP$ office/NN ./. "/"
But/CC we/PRP have/VBP to/TO look/VB at/IN improper/JJ use/NN
of/in historical/JJ buildings/NNS ./. In/IN Caserta/NNP ,/, we/
PRP need/VBD to/TO find/VB a/DT use/NN for/IN the/DT
available/JJ space/NN that/WDT is/VBZ more/RBR compatible/JJ
with/in a/DT museum/NN ./. A/DT military/NN school/NN is/VBZ
incompatible/JJ ./. "/" In/IN Caserta/NNP ,/, from/IN
opposite/JJ wings/NNS of/in a/DT palace/NN spread/NN over/IN
four/CD courtyards/NNS ;/, the/DT soldiers/NNS and/CC the/DT
museum/NN workers/NNS seem/VBP as/RB compatible/JJ as/IN
hostile/JJ neighbors/NNS in/IN a/DT crowded/VBN Neapolitan/NNP
high/JJ rise/NN ./. Vincenzo/NNP Zuccaro/NNP ,/, a/DT museum/NN
public/JJ relations/NN official/NN ,/, indignantly/RB
pointed/VBD out/IN an/DT 18th-century/JJ vault/NN fresco/NN
by/IN Mariano/NNP Rossi/NNP that/WDT had/VBD to/TO be/VB
restored/VBN because/in of/in water/NN damage/NN Zuccaro/NNP
said/VBD was/VBD caused/VBN by/in cadets/NNS '/POS showers/NNS
above/IN ./. The/DT museum/NN director/NN ,/, Livio/NNP
Ricciardi/NNP ,/, complained/VBD about/IN cooking/NN smells/VBZ
from/in the/DT school/NN kitchens/NNS ./. But/CC he/PRP
confessed/VBD that/IN he/PRP had/VBD no/DT plans/NNS or/CC
proposals/NNS on/IN how/WRB to/TO use/VB the/DT additional/JJ
rooms/NNS if/IN and/CC when/WRB the/DT school/NN moves/NNS
out/IN ./. "/" We/PRP hope/VBP they/PRP '11/MD leave/VB soon/IN
/, but/CC when/WRB they/PRP do/VBP ,/, we/PRP '11/MD be/VB
at/IN a/DT loss/NN ,/, "/" he/PRP said/VBD ./. "/" For/IN
one/CD thing/NN ,/, I/PRP don/IN VBP 't/IN VBG know/NN what/WP
rooms/NNS they/PRP have/VBP and/CC what/WP they/PRP look/VBP
like/IN ./. "/" Asked/VBN why/WRB not/IN RB ,/, he/PRP replied/VBD
,/, "/" I/PRP 've/VBP never/IN RB been/VBN invited/VBN ./. "/"
The/DT commander/NN of/in the/DT school/NN ,/, which/WDT
was/VBD established/VBN here/IN 1948/CD ,/, said/VBD
that/IN as/in a/DT soldier/NN ,/, he/PRP would/MD obey/VB
orders/NNS to/TO retreat/VB from/IN the/DT palace/NN ./. "/"
Caserta/NNP is/VBZ in/IN our/PRP$ hearts/NNS ,/, "/" the/DT
commander/NN ,/, Gen/NNP ./. Alessio/NNP Saticchi/NNP ,/
said/VBD ./. "/" Of/IN course/NN we/PRP would/MD be/VB
unhappy/JJ to/TO leave/VB but/CC that/DT is/VBZ a/DT
sentimental/JJ discourse/NN ./ On/IN a/DT practical/JJ level/NN ./ if/IN they/PRP told/VBD us/PRP ./, leave/VB so/IN we/PRP can/MD do/VB this/DT this/DT and/CC this/DT ./, I/PRP would/MD understand/VB ./, But/CC to/TO be/VB told/VBN just/RB to/TO go/VB away/RB ./, that/IN I/PRP do/VB not/RB understand/VB ./, "/" Like/IN other/JJ officers/NNS there/EX ./, General/NNP Santucci/NNP argued/VBD that/IN the/DT school/NN spends/VBZ $$/CD ./, 000/CD a/DT year/NN to/TO maintain/VB its/PRP$ part/NN of/IN the/DT building/NN ./, and/CC is/VBZ a/DT useful/JJ tenant/NN ./, He/PRP argued/VBD that/IN the/DT fire/NN was/VBD detected/VBN ./, and/CC put/VB out/IN ./, only/RB because/IN his/PRP$ men/NNS were/VBD living/VBG nearby/JJ ./, "/" Our/PRP$ fear/NN is/VBZ that/IN if/IN we/PRP leave/VBP ./, nothing/NN will/MD take/VB our/PRP$ place/NN ./, "/" he/PRP said/VBD ./, "/" It/PRP 's/VBZ like/IN seeing/VBG a/DT home/NN turn/IN into/IN an/DT abandoned/VBN house/NN ./, "/" But/CC his/PRP$ case/NN is/VBZ complicated/VBN by/IN the/DT fact/NN that/IN 10/CD years/NNS ago/RR ./, the/DT air/NN force/NN decided/VBD that/IN the/DT palace/NN was/VBD getting/VBG too/RR small/JJ for/IN the/DT school/NN ./, and/CC began/VBD building/VBG a/DT $$/125/CD million/CD complex/JJ eight/CD miles/NNS away/IN Capua/NNP ./, with/IN most/RRS of/IN the/DT money/NN coming/VBG from/IN the/DT European/NNP Economic/NNP Community/NNP ./, It/PRP took/VBD 10/CD years/NNS to/TO build/VB ./, and/CC by/IN the/DT time/NN it/PRP was/VBD close/RR to/TO completion/NN ./, cuts/VBZ in/IN the/DT military/JJ budget/NN had/VBD whittled/VBN down/JJ enrollment/NN at/IN the/DT school/NN from/IN 2/CD ./, 000/CD ./, to/TO 900/CD today/NN ./, Now/RR ./, the/DT air/NN force/NN argues/VBZ ./, Capua/NNP is/VBZ too/RR large/JJ for/IN the/DT school/NN ./, and/CC would/MD be/VB too/RR costly/JJ to/TO run/VB ./, Mrs/NNP ./, Melandri/NNP said/VBD it/PRP was/VBD the/DT military/JJ 's/POS problem/NN to/TO find/VB a/DT new/JJ site/NN by/IN Feb/NNP ./, 2/CD ./, Santicchi/NNP noted/VBD that/IN even/RR if/IN the/DT air/NN force/NN chose/VBD Capua/NNP ./, that/DT site/NN was/VBD not/RR ready/JJ for/IN habitation/NN ./, "/" They/PRP haven/NN 't/NN built/VBD the/DT barracks/NN ./, and/CC that/DT could/MD take/VB another/DT year/NN or/CC two/CD ./, "/" he/PRP said/VBD ./, Caserta/NNP Mayor/NNP Luigi/NNS Falco/NNP ./, who/WP said/VBD he/PRP wants/VBZ to/TO convert/VB other/JJ unused/JJ military/JJ buildings/NNS in/IN town/NN for/IN the/DT school/NN ./, laughed/VBD merrily/RR at/IN Mrs/NNP ./, Melandri/NNP 's/POS February/NNP deadline/NN ./, "/" That/DT ./, "/" he/PRP said/VBD ./, is/VBZ not/RR a/DT realistic/JJ deadline/NN ./, It/PRP 's/VBZ an/DT Italian/JJ deadline/NN ./, ""
DEMETRIOS G. GLINOS

Master's Thesis: "An Intelligent Editor for Natural Language Processing of Unrestricted Text"

Output file for thesis program "nlp.exe"

Input file name > 120498italy.inp

A/D T fire/NN broke/VBD out/RP recently/RB in/IN an/DT attic/NN of/IN the/DT Royal/NNP Palace/NNP here/RB ./ccrd ./p-sc

The/DT flames/NNS were/VBD quickly/RB detected/VBN and/zCC extinguished/VBN ./.

But/CC that/DT accident/NN reignited/VBD a/DT simmering/VBG controversy/NN over/IN the/DT way/NN the/DT palace/NN is/VBZ being/VBG used/VBN that/WD T is/VBZ far/RB more/RBR difficult/JJ to/TO stamp/VB out/RB ./.

The/DT Reggia/NNP di/NNP Caserta/NNP ,/cf begun/VBN in/IN the/DT mid-18th/JJ century/NN for/IN Charles/NNP III/NNP ,/capo the/DT Bourbon/NNP king/NN of/IN Naples/NNP ,/capc to/TO rival/VB Versailles/NNP ,/cf is/VBZ one/CD of/IN the/DT most/RBS imposing/VBG palaces/NNS in/IN Europe/NNP ./.

President/nnpa ,/capo Clinton/NNP ,/capc dined/VBD under/IN its/PRP$ gold/NN ceilings/NNS during/IN the/DT Group/NNP of/IN Seven/CD summit/NN meeting/NN held/VBN in/IN Naples/NNP in/IN 1994/CD ./.

Film/NNP director/NN George/NNP Lucas/NNP chose/VBD its/PRP$ late/JJ Baroque/JJ marble/NN staircase/NN for/IN a/DT scene/NN in/IN his/PRP$ coming/NN "/qc Stars/qNNP Wars/qNNPS "/qc movie/NN ,/cqdo "/qc Episode/qNNP One/qCD :/q: The/qDT Phantom/qNNP Menace/qNNP ./q. "/qc
The/DT palace/NN has/VBZ 1,200/CD rooms/NNS (/p-pr and/CC is/VBZ surrounded/VBN by/IN a/DT 250-acre/JJ park/NN that/WDT includes/VBZ a/DT 256-foot/JJ cascade/NN ./.

Versailles/NNP has/VBZ 700/CD )/x-pr

Unlike/prep Versailles/NNP ,/cavo however/RB ,/cavc the/DT palace/NN in/IN Caserta/NNP is/VBZ not/RB solely/RB a/DT museum/NN ./.

Only/RB 44/CD rooms/NNS are/VBP open/JJ to/TO the/DT public/NN ./.

Eight/CD hundred/CD rooms/NNS are/VBP now/RB under/IN military/JJ occupation/NN --/dsho specifically/RB ,/cf the/DT Italian/JJ air/NN force/NN 's/POS training/NN school/NN for/IN noncommissioned/JJ officers/NNS --/dshc ./.

The/DT museum/NN director/NN ,/cf the/DT Culture/NNP Ministry/NNP and/CC art/NN lovers/NNS want/VBP the/DT military/NN out/RB ./.

But/CC the/DT general/NN in/IN command/NN at/IN Caserta/NNP and/CC his/PRP$ men/NNS would/MD prefer/VB to/TO stay/VB ,/cf and/CC they/PRP argue/VBP that/IN they/PRP have/VBP no/DT other/JJ place/NN to/TO go/VB ./.

Local/JJ officials/NNS are/VBP caught/VBN in/IN the/DT middle/NN :/: ./p-sc

They/PRP want/VBP the/DT military/JJ to/TO leave/VB the/DT palace/NN ,/cf ./p-bt
They/PRP also/RB want/VBP them/PRP to/TO stay/VB in/IN Caserta/NNP ,/ccrd ./p-sc

They/PRP provide/VBP 300/CD jobs/NNS ./p-vp

[Subj]/p-vp spend/VB about/RB $/$ $ 9/CD million/CD a/DT year/NN ./.

The/DT emotional/JJ [,]/zCC many-sided/JJ battle/NN being/VBG fought/VBN in/IN Caserta/NNP ,/cavo moreover/RB ,/ccrd is/VBZ decades/NNS old/JJ ,/cavc and/CC far/RB from/IN unique/NN ./.

It/PRP took/VBD nearly/RB 50/CD years/NNS ,/cf ./p-bt

The/DT Culture/NNP Ministry/NNP finally/RB secured/VBD an/DT agreement/NN with/IN the/DT Defense/NNP and/zCC Finance/NNP ministries/NNS in/IN 1997/CD to/TO move/VB an/DT officers/NNS 'POS club/NN out/IN of/IN the/DT Palazzo/NNP Barberini/NNP ,/capo one/CD of/IN Italy/NNP 's/POS national/JJ galleries/NNS ,/capo in/prep Rome/NNP ./.

Retired/JJ officers/NNS are/VBP still/RB having/VBG genteel/JJ lunch/NN here/RB under/IN their/PRP$ 17th-century/JJ dining/NN room/NN 's/POS ceiling/NN fresco/NN ,/cf ./p-bt

The/DT museum/NN is/VBZ renovating/VBG a/DT smaller/JJR 1930s/CD villa/NN in/IN the/DT back/RB of/IN the/DT property/NN in/IN the/DT expectation/NN that/IN the/DT officers/NNS will/MD move/VB someday/RB ./.

In/prep Italy/NNP ,/capo a/DT country/NN overflowing/VBG with/IN art/NN treasures/NNS and/CC historical/JJ monuments/NNS ,/ccrd countless/JJ buildings/NNS have/VBP cultural/JJ significance/NN ./.

The/DT Italian/JJ military/NN owns/VBZ or/zCC occupies/VBZ some/DT of/IN the/DT finest/JJS properties/NNS ,/cf including/VBG a/DT 14th-century/JJ convent/NN in/IN Perugia/NNP now/RB used/VBD by/IN the/DT army/NN as/IN a/DT foreign/JJ language/NN school/NN ./.
But/CC police/NN academies/NNS ,/cf tax/NN inspectors/NNS and/zCC dentists/NNS also/RB work/VBP out/IN of/IN historic/JJ buildings/NNS ./.

The/DT incident/NN at/IN Caserta/NNP ,/crel ,/p-wh ,/cf lighted/VBD a/DT fire/NN in/IN Rome/NNP ./.

[prior]/nnwh occurred/VBD shortly/RB after/IN the/DT new/JJ culture/NN minister/NN ,/capo Giovanna/NNP Melandri/NNP ,/capc was/VBD appointed/VBN ,/x-wh

Mrs/NNP ./pd Melandri/NNP ,/cf 36/CD ,/ccrd vowed/VBD to/TO expel/VB the/DT military/NN from/IN Caserta/NNP and/CC from/IN dozens/NNS of/IN other/JJ cultural/JJ sites/NNS ./.

She/PRP said/mVBD the/DT air/NN force/NN had/VBD agreed/VBN to/TO find/VB a/DT new/JJ location/NN by/IN Feb/NNP ./pd 2/CD ./.

"/qo Our/qPRP$ patrimony/qNN is/qVBZ so/qRB rich/qJJ and/qzCC enormous/qJJ ,/q, it/qPRP would/qMD be/qVB impossible/qJJ to/qTO turn/qVB it/qPRP all/qDT into/qIN museums/qNNNS ,/q, "/qc Mrs/NNP ./pd Melandri/NNP said/mVBD in/IN an/DT interview/NN in/IN her/PRP$ office/NN ./.

"/qo But/qCC we/qPRP have/qVBP to/qTO look/qVB at/qIN improper/qJJ use/qNN of/qIN historical/qJJ buildings/qNNNS ./q. "/qc

"/qo In/qIN Caserta/qNNP ,/q, we/qPRP need/qVBP to/qTO find/qVB a/qDT use/qNN for/qIN the/qDT available/qJJ space/qNN that/qWDT is/qVBZ
more qRBR compatible qJJ with qIN a qDT museum qNN . /q. "/qc

"/qO A qDT military qJJ school qNN is qVBZ incompatible qJJ . /q. "/qc

In prep Caserta NNP , /capo from prep opposite JJ wings NNS of IN a DT palace NN spread NN over IN four CD courtyards NNS , /capc the DT soldiers NNS and CC the DT museum NN workers NNS seem VBP as RB compatible JJ as IN hostile JJ neighbors IN in /IN a /DT crowded VBN Neapolitan NNP high JJ rise NN . /.

Vincenzo NNP Zuccaro NNP , /capo a DT museum NN public JJ relations NNS official NN , /capc indignantly RB pointed VBD out RP an /DT 18th PP century JJ vault NN fresco NN by IN Mariano NNP Rossi NNP that WDT had VBD to TO be VB restored VBN because IN of IN water NNN damage NN Zuccaro NNP said mVBD was VBD caused VBN by IN cadets NNS '/POS showers NNS above RB . /.

The DT museum NN director NN , /capo Livio NNP Ricciardi NNP , /capc complained VBD about IN cooking NN smells VBZ from IN the DT school NN kitchens NNS . /.

But CC he PRP confessed VBD that IN he PRP had VBD no DT plans NNS or zCC proposals NNS on IN how WRB to TO use VB the DT additional JJ rooms NNS if IN and CC when WRB the DT school NN moves NNS out RB . /.

"/qO We qPRP hope qVBP they qPRP 'll qMD leave qVB soon qRB , /q, but qCC when qWRB they qPRP do qVBP , /q, we qPRP 'll qMD be qVB at qIN a qDT loss qNN , /q, "/qc he PRP said mVBD . /

"/qO For qIN one qCD thing qNN , /qccrd I qPRP don qVBP 't qVBG
know/NN what/PP rooms/NNS they/PRP have/VBP and/CC what/PP they/PRP look/VBP like/IN ./q "/qc

Asked/VBN why/WRB not/RB ,/cf he/PRP replied/VBD ,/cqdo "/q I/PRP 've/VBP never/RB been/VBN invited/VBN ./q "/qc

The/DT commander/NN of/IN the/DT school/NN ,/crel ,/p-wh ,/cf said/mVBD that/IN as/IN a/DT soldier/NN ,/cf he/PRP would/MD obey/VB orders/NNS to/TO retreat/VB from/IN the/DT palace/NN ./

[prior]/nnwh was/VBD established/VBN here/RB in/IN 1948/CD ,/x-wh

"/q Caserta/NNP is/VBP in/IN our/PRP$ hearts/NNNS ./q, "/qc
the/DT commander/NN ,/capo Gen/NNP ./pd Alessio/NNP Santicchi/NNP ,/capc said/mVBD ./.

"/q Of/IN course/NN we/PRP would/qMD be/VB unhappy/qJJ to/qTO leave/VB but/CC that/qDT is/VBZ a/DT sentimental/qJJ discourse/NN ./q "/qc

"/q On/IN a/DT practical/JJ level/NN ,/qccrd if/qIN they/PRP told/mqVBD us/PRP ,/qccrd leave/VB so/qIN we/PRP can/qMD do/qVB this/qDT this/qDT and/qZCC this/qDT ./q, I/PRP would/qMD understand/qVB ./q "/qc

"/q But/CC to/qTO be/VB told/mqVBN just/qRB to/qTO go/VB away/qRB ./q, that/qIN I/PRP do/qVBF not/qRB understand/qVBN ./q "/qc

Like/prep other/JJ officers/NNS there/EX ,/capo General/nnpa ,/capo Santicchi/NNP ,/ccrd argued/VBD that/IN the/DT school/NN spends/VBZ
$800,000/CD a/DT year/NN to/TO maintain/VB its/PRP$ part/NN of/IN the/DT building/NN ,/cf and/CC is/VBZ a/DT useful/JJ tenant/NN ./.

He/PRP argued/VBD that/IN the/DT fire/NN was/VBD detected/VBN ,/cf and/CC put/VB out/IN ,/cavo only/RB because/IN his/PRP$ men/NNS were/VBD living/VBG nearby/JJ ./.

"/qo Our/qPRP$ fear/qNN is/qVBZ that/qIN if/qIN we/qPRP leave/qVBP ,/q, nothing/qNN will/qMD take/qVB our/qPRP$ place/qNN ,/q, "/qc he/PRP said/mVBD ./.

"/qo It/qPRP 's/qVBZ like/qIN seeing/qVBG a/qDT home/qNN turn/qNN into/qIN an/qDT abandoned/qVBG house/qNN ./q. "/qc

But/CC his/PRP$ case/NN is/VBZ complicated/VBN by/IN the/DT fact/NN that/IN 10/CD years/NNS ago/RB ,/ccrd the/DT air/NN force/NN decided/VBD that/IN the/DT palace/NN was/VBD getting/VBG too/RB small/JJ for/IN the/DT school/NN ,/cf and/CC began/VBD building/VBG a/DT $/$ 125/CD million/CD complex/JJ eight/CD miles/NNS away/RB in/IN Capua/NNP ,/capo ./p-wi ./.

Most/RBS of/IN the/DT money/NN came/vbd from/IN the/DT European/NNP Economic/NNP Community/NNP ,/x-wi

It/PRP took/VBD 10/CD years/NNS to/TO build/VB ,/cf ./p-sc

By/iIN the/iDT time/iNN it/PRP was/VBD close/RB to/TO completion/NN ,/cf cuts/VBZ in/IN the/DT military/JJ budget/NN had/VBD whittled/VBN down/JJ enrollment/NN at/IN the/DT school/NN from/IN 2,000/CD ,/cf to/TO 900/CD today/NN ./.

The/DT air/NN force/NN argues/VBZ ,/ccrd Capua/NNP is/VBZ too/RB large/JJ for/IN the/DT school/NN ,/cf and/CC would/MD be/VB too/RB costly/JJ to/TO run/VB ./.
Mrs/NNP . /pd Melandri/NNP said/mVBD it/PRP was/VBD the/DT military/NN 's/POS problem/NN to/TO find/VB a/DT new/JJ site/NN by/IN Feb/NNP ./pd 2/CD ./. 

Santicchi/NNP noted/mVBD that/IN even/RB if/IN the/DT air/NN force/NN chose/VBD Capua/NNP ,/cf that/DT site/NN was/VBD not/RB ready/JJ for/IN habitation/NN ./. 

"/qo They/qPRP haven/qNN 't/qNN built/qVBD the/qDT barracks/qNN ,/q, and/qCC that/qDT could/qMD take/qVB another/qDT year/qNN or/qCC two/qCD ,/q, "/qc he/PRP said/mVBD ./. 

Caserta/NNP Mayor/NNP Luigi/NNS Falco/NNP ,/crel who/WP said/mVBD he/PRP wants/VBZ to/TO convert/VB other/JJ unused/JJ military/JJ buildings/NNS in/IN town/NN for/IN the/DT school/NN ,/cf laughed/VBD merrily/RB at/IN Mrs/NNP ./pd Melandri/NNP 's/POS February/NNP deadline/NN ./. 

"/qo That/qDT ,/q, "/qc he/PRP said/mVBD ,/cf is/VBZ not/RB a/DT realistic/JJ deadline/NN ./. 

It/PRP 's/VBZ an/DT Italian/JJ deadline/NN ./. 

Sentence splitting statistics:

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of input sentences</td>
<td>&gt; 50</td>
</tr>
<tr>
<td>Minimum input sentence length</td>
<td>&gt; 6</td>
</tr>
<tr>
<td>Maximum input sentence length</td>
<td>&gt; 54</td>
</tr>
<tr>
<td>Average input sentence length</td>
<td>&gt; 23.38</td>
</tr>
<tr>
<td>Number of output sentences</td>
<td>&gt; 62</td>
</tr>
</tbody>
</table>
Minimum output sentence length > 4
Maximum output sentence length > 44
Average output sentence length > 19.18

Program editing statistics:

Number of nonterminating periods found > 7
Number of '?' and '!' before close quote > 0
Number of idioms/collocations marked > 1
Number of non-splitting CCs/commas marked > 8
Number of quotation marks marked > 34
Number of pn appositions (w/o commas) > 2
Number of pn appositions (w/commas) > 13
Number of prep/conjunctions resolved > 7
Number of 'that' tokens marked > 0
Number of numbers in text fixed > 3
Number of multi-initial identifiers fixed > 0
Number of direct quotation commas marked > 2
Number of indirect quotation commas marked > 0
Number of indirect quotation verbs marked > 14
Number of adverbial commas marked > 5
Number of sub. conj. commas marked > 0
Number of coordination commas marked > 12
Number of wh-word commas marked > 3
Number of futile commas marked > 25
Number of open & close parentheses marked > 2
Number of open & close dashes marked > 2

Number of modals propagated > 0
Number of infinitives propagated > 0
Number of auxiliaries propagated > 0

Number of comma-which splits > 2
Number of comma-prep-which splits > 0
Number of but-clause splits > 3
Number of dash clause splits > 0
Number of parenthetical clause splits > 1
Number of comma + 'with' + gerund splits > 1
Number of subordinate clause splits > 0
Number of verb phrase coordination splits > 1
Number of sentence coordination splits > 4
Appendix C

"Flag" Article Text, Tag, and Output Files
WASHINGTON -- Using technology as new as space age fabrics and as ordinary as the chemicals from the corner dry-cleaning shop, scientists and historians here have taken on the job of cleaning and preserving the Star-Spangled Banner, the flag that flew over Fort McHenry in Baltimore in 1814, so that it might last at least another 185 years.

"A lot of our visitors do not know that there was a flag that inspired 'The Star-Spangled Banner,'" said Lonn Taylor, a historian at the Smithsonian Institution's National Museum of American History and one of the experts working on the flag that Francis Scott Key saw by dawn's early light.

Suzanne Thomassen-Krauss, the senior textile conservator for the project, said the first phase would involve separating the flag from its linen backing, which was applied in 1914 with an estimated 1.7 million stitches. They will be removed one at a time so that the big woolen flag is not damaged, and beginning Friday, visitors may watch the process at the museum, separated from workers by a glass wall.

The Smithsonian project has many elements of a detective story and, with new research,
many clues have been found to solve old mysteries like the presence of a "V" on one stripe.

One thing that is known is that the flag, which signaled to Key that the British had failed to capture Fort McHenry in the War of 1812, survived all these years in part because it was kept away from the light, which over time causes the fabric to deteriorate.

As for the stripes, the researchers have determined that the red comes from cochineal, a dye made from the dried bodies of female cochineal insects found in Mexico and the American Southwest, plus madder, or Rubia tinctorum, a plant probably grown in the Netherlands or France.

A similar dye was used for the linen threads that attached the red parts of the flag to the backing, but because linen does not hold dye as well as wool, "it creates a netlike effect on the flag," Ms. Thomassen-Krauss said.

In a square-foot area where the threads have been removed, the red is brighter because the haze effect of the threads is gone.

The same will hold true when the blue threads are removed. Many of the flag's threads are cracked, a result of flapping in the wind, age and the small amounts of light it has been exposed to.

The flag, once 42 feet by 30 feet but now reduced to 33 by 30 from what Ms. Thomassen-Krauss describes as "souvenir ing," has been rolled onto a tube resembling a huge paper towel holder. The next step will be to roll the flag out slowly, as museum visitors watch, and snip away the threads holding it to the linen backing.

Then, said Anthony Maher, a project manager with the architectural and engineering firm of KCF-SHG, which designed the laboratory in the Smithsonian where the work is to be done, the flag will be laid on an inert modern fabric called Tyvec and rolled back on the tube.

Next, it will be unrolled slowly, and dry-cleaning fluid will be dripped through the fabric as it comes off the tube.
For the past 35 years, the flag has been on display at the American history museum and before that in the Smithsonian's Art and Industries Building, all the while attracting more dirt and grime despite efforts on a smaller scale to clean it.

Ms. Thomassen-Krauss said that the fluid probably would be chlorinated hydrocarbons, the same substance that is effective in removing lipstick stains from blouses. And, like lipstick, much of the flag's soil has an oil base, brought by the air from the exhaust of cars passing outside the museum or from elevators and escalators inside.

Some of the soil is exotic, as revealed last year by a camera that the National Aeronautics and Space Administration usually uses to peer into the heavens.

"We had a photograph that showed pollen grains, clay and fibers," Ms. Thomassen-Krauss said. "One of my colleagues said that it looked like Bourbon Street after Mardi Gras."

Chlorinated hydrocarbons are a health danger, but the technicians will work for short periods in a well-ventilated enclosed area.

And, the hydrocarbons are deemed preferable to the alternatives of acetone and petroleum distillates, which are highly flammable and could wind up doing what the British tried to do.

Once the cleaning is completed, the flag will be displayed anew, although officials have not decided exactly how.

A glass case is being considered, but it would have to be as big as the facade of a three-story house and filled with a gas like argon or nitrogen that would displace oxygen, the culprit along with light in the deterioration of wool.

The project's cost is estimated at $18 million, of which $10 million has been donated by Polo Ralph Lauren and $3 million by the Pew Charitable Trusts.

According to a Smithsonian history, "The Star-Spangled Banner" entered American
folklore after Key, a Washington lawyer, went to a British truce ship on Sept. 13, 1814, to try to negotiate the release of Dr. John Beanes, who had been seized by the invading British in Maryland.

While Key was aboard the ship, a heavy rain fell and the British bombarded Fort McHenry.

The American cannons returned the fire, leading the British to detain Key on the ship overnight, although he was a negotiator, not a prisoner.

He spent the night worrying that Fort McHenry was falling to the British.

The British had already sacked Washington, burning the White House in the process, and many feared that if the British captured Baltimore, it would lead to the undoing of the American Revolution.

At dawn, the rain stopped and the British warships departed. Maj. George Armistead, Fort McHenry's commander, lowered a smaller storm flag that he had flown during the rain, and in its place he flew the huge flag that he had ordered earlier from a Baltimore flag maker, Mary Pickersgill.

It was then that Key, still aboard the truce ship, saw the flag and was so inspired by the sight that he took a letter from his pocket and wrote a poem on its back. He called his poem "The Defense of Fort McHenry," but he soon renamed it "The Star-Spangled Banner."

It was set to the popular English tune "To Anacreon in Heaven," which was about a Greek poet, and it was first performed in Baltimore in 1814. Americans have struggled to sing it ever since.

After the battle, the widow of a soldier killed in the bombardment asked Armistead for a piece of the flag as a remembrance, and he compiled, a military practice that continued until the end of the Civil War. That explains why the flag has been reduced in size, but no one is sure how it lost one of its 15 stars, which is represented today by a later embroidery.

Another mystery apparently is close to being solved by careful examination of the flag. For years, people have been puzzled by the letter "V" sewn onto one of the stripes. "I'm
reasonably sure that is the beginning of the letter 'A,' " Taylor, the historian, said.

The "A" would have stood for Armistead, whose descendants kept the flag, presumably away from the light, until donating it to the Smithsonian in 1912.

"I think it is an 'A' without the crossbar," Taylor said. "It was either never on or it fell off."

{Are you living with HIV?}
Using/VBG technology/NN as RB new/JJ as IN space/NN age/NN fabrics/NNs and CC as RB ordinary/JJ as IN the DT chemicals/NNs from/IN the DT corner/NN dry-cleaning/JJ shop/NN, /, scientists/NNs and CC historians/NNs here/RB have/VBP taken/VBN on/IN the DT job/NN of IN cleaning/NN and CC preserving/VBG the/DT Star-Spangled/NNS Banner/NNS, /, the DT flag/NN that/WD TDT flew/VBD over/IN Fort/NNS McHenry/NNS in/IN Baltimore/NNS in/IN 1814/CD, /, so/RR/BR that/IN it/PRP might/MD last/VB at/IN least/JJS another/IN DT 185/CD years/NNS/. "/" A/DT lot/NNS of IN our/PRP$ visitors/NNs do/VBP not/RB know/VB that/IN there/EX was/VBD a/DT flag/NN that/WD TDT inspired/VBD 'The/NNS Star-Spangled/NNS Banner/NNS, /, 'POS '" said/VBD Lonn/NNS Taylor/NNS, /, a DT historian/NN at/IN the DT Smithsonian/NNS Institution/NNS 's/POS National/NNS Museum/NNS of/IN American/JJ History/NNS and CC one/CD of/IN the DT experts/NNS working/VBG on/IN the DT flag/NN that/IN Francis/NNS Scott/NNS Key/NNS saw/VBD by/IN dawn/NN 's/POS early/JJ light/NN. "/.
Suzanne/NNS Thomassen-Krauss/NNS, /, DT senior/JJ textile/NN conservator/NN for/IN the DT project/NN, /, said/VBD the/DT first/JJ phase/NN would/MD involve/VB separating/VBG the/DT flag/NN from/IN its/PRP$ linen/NN backing/VBG, /, which/WDT was/VBD applied/VBN in/IN 1914/CD with/IN an/DT estimated/VBN 1/CD/CD. 7/CD million/CD stitches/NNS/. They/PRP will/MD be/VB removed/VBN one/CD at/IN a DT time/NN so/RR/BR that/IN the DT big/JJ woolen/JJ flag/NN is/VBD not/RB damaged/VBD, /, and CC beginning/VBG Friday/NNS, /, visitors/NNS may/MD watch/VB the DT process/NN at/IN the DT museum/NNS, /, separated/VBN from IN workers/NNS by/IN a DT glass/NN wall/NN. The/DT Smithsonian/NNS project/NN has/VBZ many/JJ elements/NNs of/IN a DT detective/NN story/NN and CC, /, with/IN new/JJ research/NN, /, many/JJ clues/NNS have/VBP been/VBN found/VBN to/TO solve/VB old/JJ mysteries/NNS like/IN the DT presence/NN of/IN a DT "'/" V/NN '" on/IN one/CD stripe/NN. /, One/CD thing/NN that/WDT is/VBZ known/VBN is/VBZ that/IN the DT flag/NN, /, which/WDT signaled/VBD to/TO Key/NNS that/IN the DT British/JJ had/VBD failed/VBN to/TO capture/VB Fort/NNS McHenry/NNS in/IN the DT War/NNS of/IN 1812/CD, /, survived/VBD all/PDT these DT years/NNS in/IN part/NN because/IN it/PRP was/VBD kept/VBN away/RB from/IN the DT light/NN, /, which/WDT over/IN time/NN causes/VBZ the DT fabric/NN to/TO deteriorate/VB. /, As/IN for/IN the DT stripes/NNS, /, the DT researchers/NNS have/VBP determined/VBN that/IN the DT red/JJ comes/NNS from/IN cochineal/JJ, /, a DT dye/NN made/VBN from/IN the DT dried/VBN bodies/NNS of/IN female/JJ cochineal/JJ insects/NNS found/VBN in/IN Mexico/NNS and CC American/NNS Southwest/NNS, /, plus/CC madder/NN, /, or CC Rubia/NNS tinctorum/NN, /, a DT plant/NN probably/RB grown/VBD in/IN the DT Netherlands/NNS or/CC France/NNS. /.
A/DY similar/JJ dye/NN was/VBD used/VBN for/in the DT linen/NN threads/NNS that/WDT attached/VBD the DT red/JJ parts/NNS of/IN the DT flag/NN to/TO the DT backing/NN, /, but/CC because/IN linen/NN does/VBZ not/RB hold/VB dye/NN as/RB well/RB as/IN wool/NN, /, "'/" it/PRP creates/VBZ a DT netlike/NN effect/NN on/IN the DT flag/NN, /, "'/" Ms/NNS. /.
Thomassen-Krauss/NNS said/VBD ./.
In/IN a DT square-foot/JJ area/NN where/WRB the DT threads/NNS have/VBP been/VBN removed/VBN, /, the DT red/NN is/VBZ brighter/JJR because/IN the DT haze/NN effect/NN of/IN the DT threads/NNS is/VBZ gone/VBN. ./ The DT same/JJ will/MD
hold/VB true/JJ when/WRB the/DT blue/JJ threads/NNS are/VBP removed/VBN ./ Many/JJ of/IN the/DT flag/NN 's/POS threads/NNS are/VBP cracked/VBN ./ a/DT result/NN of/IN flapping/VBG in/IN the/DT wind/NN ./ age/NN and/CC the/DT small/JJ amounts/NNS of/IN light/NN it/PRP has/VBZ been/VBN exposed/VBN to/TO ./ The/DT flag/NN ./ once/RB 42/CD feet/NNS by/IN 30/CD feet/NNS but/CC now/RB reduced/VBN to/TO 33/CD by/IN 30/CD from/IN what/WP Ms/NNP ./ Thomassen-Krauss/NNP describes/VBZ as/IN "/" souveniring/VBG ./ "/" has/VBZ been/VBN rolled/VBN onto/IN a/DT tube/NN resembling/VBG a/DT huge/JJ paper/NN towel/NN holder/NN ./ The/DT next/JJ step/NN will/MD be/VE to/TO roll/VE the/DT flag/NN out/IN slowly/RR ./ as/IN museum/NN visitors/NNS watch/VBP ./ and/CC snip/VB away/RR the/DT threads/NNS holding/VBG it/PRP to/TO the/DT linen/NN backing/VBG ./ Then/RB ./ said/VBD Anthony/NNP Maher/NNP ./ a/DT project/NN manager/NN with/IN the/DT architectural/JJ and/CC engineering/NN firm/NN of/IN KCF-SHG/NNP ./ which/WDT designed/VBD the/DT laboratory/NN in/IN the/DT Smithsonian/NNP where/WRB the/DT work/NN is/VBZ to/TO be/VE done/VBN ./ the/DT flag/NN will/MD be/VE laid/VBN on/IN an/DT inert/JJ modern/JJ fabric/NN called/VBN Tyvec/NNP and/CC rolled/VBD back/RR on/IN the/DT tube/NN ./ Next/JJ ./ it/PRP will/MD be/VE unrolled/JJ slowly/RR ./ and/CC dry-cleaning/VBG fluid/NN will/MD be/VE dripped/VBD through/IN the/DT fabric/NN as/IN it/PRP comes/VBZ off/IN the/DT tube/NN ./ For/IN the/DT past/JJ 35/CD years/NNS ./ the/DT flag/NN has/VBZ been/VBN on/IN display/NN at/IN the/DT American/JJ history/NN museum/NN and/CC before/IN that/DT in/IN the/DT Smithsonian/NNP 's/POS Art/NNP and/CC Industries/NNP Building/NNP ./ all/PDT the/DT while/NN attracting/VBG more/JJR dirt/NN and/CC grime/NN despite/IN efforts/NNS on/IN IN a/DT smaller/JJR scale/NN to/TO clean/VB it/PRP ./ Ms/NNP ./ Thomassen-Krauss/NNP said/VBD that/IN the/DT fluid/NN probably/RB would/MD be/VE chlorinated/VBN hydrocarbons/NNS ./ the/DT same/JJ substance/NN that/WDT is/VBZ effective/JJ in/IN removing/VBG lipstick/NN stains/NNS from/IN blouses/NNS ./ And/CC ./ like/IN lipstick/NN ./ much/NN of/IN the/DT flag/NN 's/POS soil/NN has/VBZ an/DT oil/NN base/NN ./ brought/VBN by/IN the/DT air/NN from/IN the/DT exhaust/NN of/IN cars/NNS passing/VBG outside/IN the/DT museum/NN or/CC from/IN elevators/NNS and/CC escalators/NNS inside/RR ./ Some/DT of/IN the/DT soil/NN is/VBZ exotic/JJ ./ as/IN revealed/VBN last/JJ year/NN by/IN a/DT camera/NN that/IN the/DT National/NNP Aeronautics/NNP and/CC Space/NNP Administration/NNP usually/RB uses/VBZ to/TO peer/VE into/IN the/DT heavens/NNS ./ "/" We/PRP had/VBD a/DT photograph/NN that/WDT showed/VBD pollen/NN grains/NNS ./ clay/NN and/CC fibers/NNS ./ "/" Ms/NNP ./ Thomassen-Krauss/NNP said/VBD ./ One/CD of/IN my/PRP$ colleagues/NNS said/VBD that/IN it/PRP looked/VBD like/IN Bourbon/NNP Street/NNP after/IN Mardi/NNP Gras/NNP ./ "/" Chlorinated/JJ hydrocarbons/NNS are/VBP a/DT health/NN danger/NN ./ but/CC the/DT technicians/NNS will/MD work/VB for/IN short/JJ periods/NNS in/IN a/DT well-ventilated/RB enclosed/VBN area/NN ./ And/CC ./ the/DT hydrocarbons/NNS are/VBP deemed/VBN preferable/JJ to/TO the/DT alternatives/NNS of/IN acetone/NN and/CC petroleum/NN distillates/NNS ./ which/WDT are/VBP highly/RB flammable/JJ and/CC could/MD wind/VB up/IN doing/VBG what/WP the/DT British/JJ tried/VBD to/TO do/VB ./ Once/RB the/DT cleaning/VBG
is/VBZ completed/VBN /, the/DT flag/NN will/MD be/VB displayed/VBN anew/RB /, although/IN officials/NNS have/VBP not/RB decided/VBN exactly/RB how/WRB ./. A/DT glass/NN case/NN is/VBZ being/VBG considered/VBN /, but/CC it/PRP would/MD have/VB to/TO be/VB as/RB big/JJ as/IN the/DT facade/NN of/IN a/DT three-story/JJ house/NN and/CC filled/VBN with/IN a/DT gas/NN like/IN argon/NN or/CC nitrogen/NN that/IN WDT would/MD displace/VB oxygen/NN /, the/DT culprit/NN along/IN with/IN light/NN in/IN the/DT deterioration/NN of/IN wool/NN ./. The/DT project/NN 's/POS cost/NN is/VBZ estimated/VBN at/IN $/$$ 18/CD million/CC /, of/IN which/WDT S/$$ 10/CD million/CD has/VBZ been/VBN donated/VBN by/IN Polo/NNP Ralph/NNP Lauren/NNP and/CC $/$$ 3/CD million/CD by/IN the/DT Pew/NNP Charitable/JJ Trusts/NNS ./. According/VBG to/TO a/DT Smithsonian/NNP history/NN /, "/" The/DT Star-Spangled/NNP Banner/NNP "/" entered/VBD American/JJ folklore/NN after/IN Key/NN /, a/DT Washington/NNP lawyer/NN /, went/VBD to/TO a/DT British/JJ truce/NN ship/NN on/IN Sept/NNP /, 13/CD /, 1814/CD /, to/TO try/VB to/TO negotiate/VB the/DT release/NN of/IN Dr/NNP ./. John/NNP Beanes/NNP /, who/IN had/VBD been/VBN seized/VBN by/IN the/DT invading/VBG British/JJ in/IN Maryland/NNP /, while/IN Key/NN was/VBD aboard/IN the/DT ship/NN /, a/DT heavy/JJ rain/NN fell/VBD and/CC the/DT British/JJ bombarded/VBD Fort/NNP McHenry/NNP /, the/DT American/JJ cannons/NNS returned/VBD the/DT fire/NN /, leading/VBG the/DT British/JJ to/to detain/VB Key/NN on/IN the/DT ship/NN overnight/JJ /, although/IN he/PRP was/VBD a/DT negotiator/NN /, not/RB a/DT prisoner/NN /, he/PRP spent/VBD the/DT night/NN worrying/VBG that/IN Fort/NNP McHenry/NNP was/VBD falling/VBG to/TO the/DT British/JJ ./. The/DT British/JJ had/VBD already/RB sacked/VBN Washington/NNP /, burning/VBG the/DT White/NNP House/NNP in/IN the/DT process/NN /, and/CC many/NN feared/VBD that/IN if/IN the/DT British/JJ captured/VBN Baltimore/NNP /, it/PRP would/MD lead/VB to/to the/DT undoing/NN of/IN the/DT American/NNP Revolution/NN /, at/IN dawn/NN /, the/DT rain/NN stopped/VBD and/CC the/DT British/JJ warships/NNS departed/VBD ./. Maj/NNP ./. George/NNP Armistead/NNP /, Fort/NNP McHenry/NNP 's/POS commander/NN /, lowered/VBD a/DT smaller/JJR storm/NN flag/NN that/IN he/PRP had/VBD flown/during/IN the/DT rain/NN /, and/CC in/IN its/PRP$ place/NN he/PRP flew/VBD the/DT huge/JJ flag/NN that/IN he/PRP had/VBD ordered/VBN earlier/RBR from/IN a/DT Baltimore/NNP flag/NN maker/NN /, Mary/NNP Pickersgill/NNP ./. It/PRP was/VBD then/RB that/IN Key/NN /, still/RB aboard/IN the/DT truce/JJ ship/NN /, saw/VBD the/DT flag/NN and/CC was/VBD so/RB inspired/VBN by/IN the/DT sight/NN that/IN he/PRP took/VBD a/DT letter/NN from/IN his/PRP$ pocket/NN and/CC wrote/VBD a/DT poem/NN on/IN its/PRP$ back/RB /, he/PRP called/VBD his/PRP$ poem/NN "/" The/DT Defense/NNP of/IN Fort/NNP McHenry/NNP /, "/" but/CC he/PRP soon/RB renamed/VBD it/PRP "/" The/DT Star-Spangled/NNP Banner/NNP ./. "/" It/PRP was/VBD set/VBN to/TO the/DT popular/JJ English/NNP tune/NN "/" To/TO Anacreon/NNP in/IN Heaven/NNP /, "/" which/WDT was/VBD about/IN a/DT Greek/NNP poet/NN /, and/CC it/PRP was/VBD first/RB performed/VBN in/IN Baltimore/NNP in/IN 1814/CD /, Americans/NNS have/VBP struggled/VBN to/to sing/VB it/PRP ever/RB since/RB ./. After/IN the/DT battle/NN /, the/DT widow/NN of/IN a/DT soldier/NN killed/VBN in/IN the/DT
bombardment/NN asked/VBD Armstead/NNP for/IN a/DT piece/NN of/IN the/DT flag/NN as/IN a/DT remembrance/NN ,/, and/CC he/PRP complied/VBD ,/, a/DT military/JJ practice/NN that/WDT continued/VBD until/IN the/DT end/NN of/IN the/DT Civil/NNP War/NNP ./. That/DT explains/VBZ why/WRB the/DT flag/NN has/VBZ been/VBN reduced/VBN in/IN size/NN ,/, but/CC no/DT one/CD is/VBZ sure/JJ how/WRB it/PRP lost/VBD one/CD of/IN its/PRP$ 15/CD stars/NNS ,/, which/WDT is/VBZ represented/VBN today/NN by/IN a/DT later/JJ embroidery/NN ./. Another/DT mystery/NN apparently/RB is/VBZ close/RB to/TO being/VBG solved/VBN by/IN careful/JJ examination/NN of/IN the/DT flag/NN ./. For/IN years/NNS ,/, people/NNS have/VBP been/VBN puzzled/VBN by/IN the/DT letter/NN "/" V/NN "/" sewn/VBN onto/IN one/CD of/IN the/DT stripes/NNS ./. "/" I/PRP 'm/VBP reasonably/RB sure/JJ that/WDT is/VBZ the/DT beginning/NN of/IN the/DT letter/NN 'A/NN ,/, 'POS "/" Taylor/NNP ,/, the/DT historian/NN ,/, said/VBD ./. The/DT "/" A/DT "/" would/MD have/VB stood/VBN for/IN Armstead/NNP ,/, whose/WP$ descendants/NNS kept/VBD the/DT flag/NN ,/, presumably/RB away/RB from/IN the/DT light/NN ,/, until/IN donating/VBG it/PRP to/TO the/DT Smithsonian/NNP in/IN 1912/CD ./. "/" I/PRP think/VBP it/PRP is/VBZ an/DT 'A/NN "/" POS without/IN the/DT crossbar/NN ,/, "/" Taylor/NNP said/VBD ./. "/" It/PRP was/VBD either/DT never/RB on/IN or/CC it/PRP fell/VBD off/RB ./. "/"
DEMETRIOS G. GLINOS

Master's Thesis: "An Intelligent Editor for Natural Language Processing of Unrestricted Text"

Output file for thesis program "nlp.exe"

Input file name > 052599flag.inp

Using/VBG technology/NNS as/IN RB new/JJ as/IN space/NNS age/NNS fabrics/NNS and/CC as/IN RB ordinary/JJ as/IN the/DT chemicals/NNS from/IN the/DT corner/NN dry-cleaning/JJ shop/NN ,/cf scientists/NNS and/CC historians/NNS here/RB have/VBP taken/VBN on/IN the/DT job/NN of/IN cleaning/NN and/CC preserving/VBG the/DT Star-Spangled/NNP Banner/NNP ,/cf the/DT flag/NN that/WDT flew/VBD over/IN Fort/NNP McHenry/NNP in/IN Baltimore/NNP in/IN 1814/CD ,/cavo so/RB that/IN it/PRP might/MD last/VB at/iIN least/iJJS another/DT 185/CD years/NNS ./. 

"/qo A/qDT lot/qNN of/qIN our/qPRPS visitors/qNNS do/qVBP not/qRB know/qVB that/qIN there/qEX was/qVBD a/qDT flag/qNN that/qWDT inspired/qVBD 'The/qNNP Star-Spangled/qNNP Banner/qNNP ,/qccrd '/qPOS "/qc said/mVBD Lonn/NNP Taylor/NNP ,/cf a/DT historian/NN at/IN the/DT Smithsonian/NNP Institution/NNP 's/POS National/NNP Museum/NNP of/IN American/JJ History/NN and/CC one/CD of/IN the/DT experts/NNS working/VBG on/IN the/DT flag/NN that/IN Francis/NNP Scott/NNP Key/NNP saw/VBD by/IN dawn/NN 's/POS early/JJ light/NN ./. 

Suzanne/NNP Thomassen-Krauss/NNP ,/capo the/DT senior/JJ textile/NN conservator/NN for/IN the/DT project/NN ,/capc said/mVBD the/DT first/JJ phase/NN would/MD involve/VB separating/VBG the/DT flag/NN from/IN its/PRPs linen/NN backing/VBG ,/crel ,/p-wh ./. 

[prior]nnwh was/VBD applied/VBN in/IN 1914/CD with/IN an/DT estimated/VBN 1.7/CD million/CD stitches/NNS ,/x-wh

They/PRP will/MD be/VB removed/VBN one/CD at/in/IN a/DT time/NN so/RB that/IN the/DT big/JJ woolen/JJ flag/NN is/VBZ not/RB damaged/VBN
The DT Smithsonian/NNP project/NN has/VPB many/JJ elements/NNS of/IN a/DT detective/NN story/NN ./p-sc

One/CD thing/NN that/WDT is/VPB known/VBN is/VPB that/IN the/DT flag/NN ./crel ./p-wh ./cf survived/VBD all/PDT these/DT years/NNS in/IN part/NN because/IN it/PRP was/VBD kept/VBN away/RB from/IN the/DT light/NN ./crel ./p-wh

[prior]/nnwh signaled/VBD to/TO Key/NNP that/IN the/DT British/JJ had/VBD failed/VBN to/TO capture/VB Fort/NNP McHenry/NNP in/IN the/DT War/NNP of/IN 1812/CD ./x-wh ./

Time/NN causes/VPB the/DT fabric/NN to/TO deteriorate/VB [prior]/nnwh ./x-wh

As/scnj for/IN the/DT stripes/NNS ./ccrd the/DT researchers/NNS have/VBP determined/VBN that/IN the/DT red/JJ comes/NNS from/IN cochineal/JJ ./ccrd a/DT dye/NN made/VBN from/IN the/DT dried/VBN bodies/NNS of/IN female/JJ cochineal/JJ insects/NNS found/VBD in/IN Mexico/NNP and/CC the/DT American/NNP Southwest/NNP ./capo plus/CC madder/NN ./capo or/CC Rubia/NNP tinctorum/NN ./capc a/DT plant/NN probably/RB grown/VBD in/IN the/DT Netherlands/NNP or/CC France/NNP ./

A/DT similar/JJ dye/NN was/VBD used/VBN for/IN the/DT linen/NN threads/NNS that/WDT attached/VBD the/DT red/JJ parts/NNS of/IN the/DT flag/NN to/TO the/DT backing/NN ./cf ./p-bt

Linen/NN does/VPB not/RB hold/VB dye/NN as/RB well/RB as/IN wool/NN ./cqdo "./qo it/qPRP creates/qVBP a/DT netlike/qNN effect/qNN on/qIN the/qDT flag/qNN ./q ./qc Ms/NNP ./pd Thomassen-Krauss/NNP said/mVBD ./

122
In Prep a DT square-foot JJ area NN where WRB the DT threads NNS have VBP been VBN removed VBN /cf the DT red NN is VBZ brighter JJR because IN the DT haze NN effect NN of IN the DT threads NNS is VBZ gone VBN ./. 

The DT same JJ will MD hold VB true JJ when WRB the DT blue JJ threads NNS are VBP removed VBN ./. 

Many JJ of IN the DT flag NN 's POS threads NNS are VBP cracked VBN ,/ccrd ./p-sc 

A DT result NN of IN flapping VBG in IN the DT wind NN [,] /zCC age NN and zCC the DT small JJ amounts NNS of IN light NN it PRP has VBZ been VBN exposed VBN to TO ./. 

The DT flag NN ,/cavo once RB 42 CD feet NNS by IN 30 CD feet NNS but CC now RB reduced VBN to TO 33 CD by IN 30 CD from IN what WP Ms NNP ./pd Thomassen Krauss NNP describes VBZ as IN "/go souveniring qVBG ,/q , "/gc has VBZ been VBN rolled VBN onto IN a DT tube NN resembling VBG a DT huge JJ paper NN towel NN holder NN ./.

The DT next JJ step NN will MD be VB to TO roll VB the DT flag NN out IN slowly RB ,/ccrd ./p-cj ./p-vp 

Museum NN visitors NNS watch VBP ,/x-cj 

[Subj]/p-vp will md snip VB away RB the DT threads NNS holding VBG it PRP to TO the DT linen NN backing VBG ./. 

Then RB ,/ccrd said mVBD Anthony NNP Maher NNP ,/capo a DT project NNP manager NN with IN the DT architectural JJ and CC engineering NNP firm NN of IN KCF SHG NNP ,/capc ,/p-wh ,/ccrd the DT flag NN will MD be VB laid VBN on IN an DT inert JJ modern JJ fabric NN called VBN Tyvec NNP and CC rolled VBD back RB on IN the DT tube NN ./.
Next/JJ ,/ccrd it/PRP will/MD be/VB unrolled/JJ slowly/RB ,/cf and/CC dry-cleaning/JJ fluid/NN will/MD be/VB dripped/VBD through/IN the/DTD fabric/NN as/IN it/PRP comes/VBZ off/IN the/DTD tube/NN ./.

For/prep the/DTD past/JJ 35/CD years/NNS ,/ccrd the/DTD flag/NN has/VBZ been/VBN on/IN display/NN at/IN the/DTD American/JJ history/NN museum/NN and/CC before/IN that/DTD in/NN the/DTD Smithsonian/NNP 's/POS Art/NNP and/CCZGC Industries/NNP Building/NNP ,/ccrd all/PDT the/DTD while/NN attracting/VBG more/JJR dirt/NN and/CCZCC grime/NN despite/IN efforts/NNS on/IN a/DTD smaller/JJR scale/NN to/TO clean/VB it/PRP ./.

Ms/NNP ,/pd Thomassen-Krauss/NNP said/mVBD that/IN the/DTD fluid/NN probably/RB would/MD be/VB chlorinated/VBN hydrocarbons/NNS ,/cf the/DTD same/JJ substance/NN that/WDT is/VBZ effective/JJ in/IN removing/VBG lipstick/NN stains/NNS from/IN blouses/NNS ./.

And/CC ,/ccrd like/prep lipstick/NN [,]/ZCC much/NN of/IN the/DTD flag/NN 's/POS soil/NN has/VBZ an/DTD oil/NN base/NN ,/ccrd brought/VBN by/IN the/DTD air/NN from/IN the/DTD exhaust/NN of/IN cars/NNS passing/VBG outside/IN the/DTD museum/NN or/CC from/IN elevators/NNS and/CCZCC escalators/NNS inside/RB ./.

Some/DTD of/IN the/DTD soil/NN is/VBZ exotic/JJ ,/ccrd as/prep revealed/VBN last/JJ year/NN by/IN a/DTD camera/NN that/IN the/DTD National/NNP Aeronautics/NNP and/ZCC Space/NNP Administration/NNP usually/RB uses/VBZ to/TO peer/VB into/IN the/DTD heavens/NNS ./.

"/go We/qPRP had/qVBD a/qDT photograph/qNN that/qWDT showed/qVBD pollen/qNN grains/qNNS ,/q, clay/qNN and/qCC fibers/qNNS ,/q, "/qc
Chlorinated hydrocarbons are highly flammable and could wind up doing what the British tried to do, but the technicians will work for short periods in a well-ventilated enclosure.

And the hydrocarbons are deemed preferable to alternatives of acetone and petroleum distillates. Chlorinated hydrocarbons are highly flammable and could cause wind-up doing what the British tried to do, but the time the cleaning is completed will be displayed anew.

Officials have not decided exactly how it would be.

It would have to be big as the facade of a three-story house and filled with gas like argon or nitrogen that would displace oxygen. The culprit along with light in the deterioration of wool.

The project's cost is estimated at $18.
Of/ prep which/ WDT $/$ $10/ CD million/ CD has/ VBZ been/ VBN donated/ VBN by/ IN Polo/ NNP Ralph/ NNP Lauren/ NNP and/ CC $/$ $3/ CD million/ CD by/ IN the/ DT Pew/ NNP Charitable/ JJ Trusts/ NNS . /.

According/ VBG to/ TO a/ DT Smithsonian/ NNP history/ NN ,/ cqdo "/ qo The/ qDT Star-Spangled/ qNNP Banner/ qNNP "/ qc entered/ VBD American/ JJ folklore/ NN after/ IN Key/ NNP ,/ capo a/ DT Washington/ NNP lawyer/ NN ,/ capc went/ VBD to/ TO a/ DT British/ JJ truce/ NN ship/ NN on/ IN Sept/ NNP .13/ CD [ ]/ zCC 1814/ CD ,/ cf to/ TO try/ VB to/ TO negotiate/ VB the/ DT release/ NN of/ IN Dr/ NNP . / pd John/ NNP Beanes/ NNP ,/ crel who/ WP had/ VBD been/ VBN seized/ VBN by/ IN the/ DT invading/ VBG British/ JJ in/ IN Maryland/ NNP . /.

Key/ NNP was/ VBD aboard/ IN the/ DT ship/ NN ,/ ccrd . / p-sc
A/ DT heavy/ JJ rain/ NN fell/ VBD . / p-vp

[Subj]/ p-vp the/ DT British/ JJ bombarded/ VBD Fort/ NNP McHenry/ NNP . /.

The/ DT American/ JJ cannons/ NNS returned/ VBD the/ DT fire/ NN ,/ cf leading/ VBG the/ DT British/ JJ to/ TO detain/ VB Key/ NNP on/ IN the/ DT ship/ NN overnight/ JJ ,/ cf . / p-cj

He/ PRP was/ VBD a/ DT negotiator/ NN ,/ capo not/ RB a/ DT prisoner/ NN . /

He/ PRP spent/ VBD the/ DT night/ NN worrying/ VBG that/ IN Fort/ NNP McHenry/ NNP was/ VBD falling/ VBG to/ TO the/ DT British/ JJ . /.

The/ DT British/ JJ had/ VBD already/ RB sacked/ VBN Washington/ NNP ,/ capo burning/ VBG the/ DT White/ NNP House/ NNP in/ IN the/ DT process/ NN ,/ ccrd and/ zCC many/ NN feared/ VBD that/ IN if/ IN the/ DT British/ JJ captured/ VBN Baltimore/ NNP ,/ cf it/ PRP would/ MD lead/ VB to/ TO the/ DT undoing/ NN of/ IN the/ DT American/ NNP Revolution/ NNP . /.

126
At dawn, the rain stopped and British warships departed.

Maj. Armistead lowered the flag that had flown during the rain from its place. He flew the huge flag that he had ordered earlier from Baltimore. Captain Mary Pickersgill made the flag and its back.

It was then that Key, still aboard the ship, saw the flag and wrote a poem inspired by it. He took the poem and wrote another.

He called his poem "The Defense of Fort McHenry". He soon renamed it "The Star-Spangled Banner".

It was set to the popular English tune "To Anacreon in Heaven" which was about a Greek poet.

It was first performed in Baltimore in 1814.

Americans have struggled to sing it ever since.
The/DN battle/DDN, ccrd the/DDN widow/DDN of/DDN a/DDN soldier/DDN killed/VBN in/IN the/DDN bombardment/DDN asked/VBD Armistead/NNP for/IN a/DDN piece/DDN of/DDN the/DDN flag/DDN as/IN a/DDN remembrance/DDN, /p-cj

And/CC he/PRP complied/VBD, /cf a/DDN military/JJ practice/DDN that/WDT continued/VBD until/IN the/DDN end/DDN of/IN the/DDN Civil/NNP War/NNP ./.  

That/DDN explains/VBZ why/WRB the/DDN flag/DDN has/VBZ been/VBN reduced/VBN in/IN size/DDN, /cf ./p-bt

[Subj]/p-bt no/DDN one/CD is/VBZ sure/JJ how/WRB it/PRP lost/VBD one/CD of/IN its/PRP$ 15/CD stars/NNS, /crel ,/p-wh ./.

[prior]/nnwh is/VBZ represented/VBN today/DDN by/IN a/DDN later/JJ embroidery/DDN, /x-wh

Another/DDN mystery/DDN apparently/RB is/VBZ close/RB to/TO being/VBG solved/VBN by/IN careful/JJ examination/DDN of/IN the/DDN flag/DDN ./.

For/prep years/DDN [,] zCC people/NNS have/VBP been/VBN puzzled/VBN by/IN the/DDN letter/DDN "/go V/qNN " /qc sewn/VBN onto/IN one/CD of/IN the/DDN stripes/NNS ./.

"/go I/qPRP 'm/qVBP reasonably/qRB sure/qJJ that/qWDT is/qVBZ the/qDT beginning/qNN of/qIN the/qDT letter/qNN 'A/qNN , /q, /qPPOS "/qqc Taylor/NNP ,/capo the/DDN historian/DDN ,/capc said/mVBD ./.

The/DDN "/go A/qDT "/qc would/MD have/VB stood/VBN for/IN Armistead/NNP, /crel whose/WPS descendants/NNS kept/VBD the/DDN flag/DDN ,/capo presumably/RB away/RB from/IN the/DDN light/DDN ,/capo until/prep donating/VBG it/PRP to/TO the/DDN Smithsonian/NNP in/IN 1912/CD ./.
"/qo I/qPRP think/qVBP it/qPRP is/qVBZ an/qDT 'A/qNN '/qPOS without/qIN the/qDT crossbar/qNN ,/q, "/qc Taylor/NNP said/mVBD ./.

"/qo It/qPRP was/qVBD either/qDT never/qRB on/qIN or/qCC it/qPRP fell/qVBD off/qRB ./q. "/qc

Sentence splitting statistics:

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of input sentences</td>
<td>&gt; 45</td>
</tr>
<tr>
<td>Minimum input sentence length</td>
<td>&gt; 9</td>
</tr>
<tr>
<td>Maximum input sentence length</td>
<td>&gt; 60</td>
</tr>
<tr>
<td>Average input sentence length</td>
<td>&gt; 31.73</td>
</tr>
<tr>
<td>Number of output sentences</td>
<td>&gt; 67</td>
</tr>
<tr>
<td>Minimum output sentence length</td>
<td>&gt; 4</td>
</tr>
<tr>
<td>Maximum output sentence length</td>
<td>&gt; 60</td>
</tr>
<tr>
<td>Average output sentence length</td>
<td>&gt; 21.58</td>
</tr>
</tbody>
</table>

Program editing statistics:

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of nonterminating periods found</td>
<td>&gt; 6</td>
</tr>
<tr>
<td>Number of '?' and '!' before close quote</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Number of idioms/collocations marked</td>
<td>&gt; 2</td>
</tr>
<tr>
<td>Number of non-splitting CCs/commas marked</td>
<td>&gt; 15</td>
</tr>
<tr>
<td>Number of quotation marks marked</td>
<td>&gt; 30</td>
</tr>
<tr>
<td>Number of pn appositions (w/o commas)</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Number of pn appositions (w/commas)</td>
<td>&gt; 13</td>
</tr>
<tr>
<td>Number of prep/conjunctions resolved</td>
<td>&gt; 15</td>
</tr>
<tr>
<td>Number of 'that' tokens marked</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Number of numbers in text fixed</td>
<td>&gt; 2</td>
</tr>
<tr>
<td>Number of multi-initial identifiers fixed</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Number of direct quotation commas marked</td>
<td>&gt; 2</td>
</tr>
<tr>
<td>Number of indirect quotation commas marked</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Number of indirect quotation verbs marked</td>
<td>&gt; 9</td>
</tr>
<tr>
<td>Number of adverbial commas marked</td>
<td>&gt; 2</td>
</tr>
<tr>
<td>Number of sub. conj. commas marked</td>
<td>&gt; 2</td>
</tr>
<tr>
<td>Number of coordination commas marked</td>
<td>&gt; 25</td>
</tr>
<tr>
<td>Number of wh-word commas marked</td>
<td>&gt; 6</td>
</tr>
<tr>
<td>Number of futile commas marked</td>
<td>&gt; 25</td>
</tr>
<tr>
<td>Number of open &amp; close parentheses marked</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Number of open &amp; close dashes marked</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Number of modals propagated</td>
<td>&gt; 1</td>
</tr>
<tr>
<td>Number of infinitives propagated</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Number of auxiliaries propagated</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Number of comma-which splits</td>
<td>&gt; 6</td>
</tr>
<tr>
<td>Number of comma-prep-which splits</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Number of but-clause splits</td>
<td>&gt; 4</td>
</tr>
<tr>
<td>Number of dash clause splits</td>
<td>&gt; 0</td>
</tr>
</tbody>
</table>
Number of parenthetical clause splits  > 0
Number of comma + 'with' + gerund splits  > 0
Number of subordinate clause splits  > 4
Number of verb phrase coordination splits  > 2
Number of sentence coordination splits  > 6
Appendix D

IE and Brill Tagger Makefiles for the PC
DEMETRIOS G. GLINOS

Master's Thesis: "An Intelligent Editor for Natural Language Processing of Unrestricted Text"

Makefile for the Intelligent Editor

***************************************************************************

EFILE  = nlp

OBJS   = nlp.o nlp1.o nlp2.o nlp3.o

HDR    = nlp.h

$(EFILE): $(OBJS)
    @echo "linking ..... "
    gcc -o $(EFILE) $(OBJS)

$(OBJS): $(HDR)
    @echo "compiling ..... "
    gcc -c $*.

132
# Makefile for Brill Tagger

EFILE = tagger

OBJS = tagger.o lex.o useful.o darray.o registry.o memory.o

HDR = lex.h useful.h darray.h registry.h memory.h

$(EFILE): $(OBJS)
  @echo "linking .... "
  gcc -o $(EFILE) $(OBJS)

$(OBJS): $(HDR)
  @echo "compiling .... "
  gcc -c *.c
# Makefile for Brill Tagger

EFILE = starttag

OBJS = starttag.o lex.o useful.o darray.o registry.o memory.o

HDR = lex.h useful.h darray.h registry.h memory.h

$(EFILE): $(OBJS)
    @echo "linking ..... "
    gcc -o $(EFILE) $(OBJS)

$(OBJS): $(HDR)
    @echo "compiling ..... "
    gcc -c *.c
# Makefile for Brill Tagger

EFILE = fintag

OBJE = fintag.o lex.o useful.o darray.o registry.o memory.o

HDR = lex.h useful.h darray.h registry.h memory.h

$(EFILE): $(OBJE)
   @echo "linking ..... "
   gcc -0 $(EFILE) $(OBJE)

$(OBJE): $(HDR)
   @echo "compiling ..... "
   gcc -c $*.c
Appendix E

Intelligent Editor Source Code
DEMETRIOS G. GLINOS

Master's Thesis: "An Intelligent Editor for Natural Language Processing of Unrestricted Text"

Header file "nlp.h"

#include <stdio.h>

int Znum;       // Number of tokens in the current sentence
int csub_count;
int vp_splits;
int that_count;
int mod_count;
int inf_count;
int aux_count;
int iqv_count;

int o_min, o_max, o_count, o_sum;

char buff[40], token[200][30], tag[200][7];

enum boolean {FALSE, TRUE};
enum boolean open_quote;
enum boolean end_of_sentence(int t_index);

void output_sentence(FILE *outfile, int num_tokens);
void fix_gerund(int gdex);

int mark_idioms(void);
int mark_ands(void);
int mark_quotes(void);
int mark_pn_appos(void);
int mark_pn_appos_commas(void);
int fix_prep_subconj(void);
int mark_d_quotes(void);
int mark_l_quotes(void);
int mark_adverbials(void);
int mark_coords(void);
int mark_wh_words(void);
int mark_futiles(void);
int mark_paren(void);
int mark_dashes(void);
int split_which(void);
int split_prep_which(void);
int split_subconj(void);
int split_but(void);
int split_dash(void);
int split_paren(void);
int split_svp_coord(void);
int split_with(void);
int d_strlen(char *s);
int shift_right(int first, int num, int places);
int shift_left(int first, int num, int places);
int merge(int first, int num);
DEMETRIOS G. GLINOS

Master's Thesis: "An Intelligent Editor for Natural Language Processing of Unrestricted Text"

Main program file "nlp.c"

*******************************************************************************/

#include <stdio.h>
#include <assert.h>
#include "nlp.h"

//
*******************************************************************************/
// Main program to analyze Brill tagger output
//
*******************************************************************************/

int main(int argc, char **argv)
{
    FILE *fin, *fout;
    int num, i, j;
    enum boolean quote;

    // Input sentence length statistics
    int i_min = 100,
        i_max = 0,
        i_count = 0,
        i_sum  = 0;

    // Editing statistics
    int Zpd = 0,
        Zpq = 0,
        Zidm = 0,
        Za = 0,
        Zquote = 0,
        Zapos1 = 0,
        Zapos2 = 0,
        Zfix = 0,
        Zfixnum = 0,
        Zfixmult = 0,
        Zdirect = 0,
        Zindirect = 0,
        Zadverb = 0,
        Zcoord = 0,
        Zwh = 0,
        Zfutile = 0,
        Zparen = 0,
        Zdash = 0,
        Zwhich = 0,
        Zpwhich = 0,
        Zwith = 0,
        Zsubconj = 0,
Zbut = 0,
Zds = 0,
Zpr = 0,
Zvp = 0,
Zsc = 0;

// Editing statistics declared in file nlp.h
csup_count = 0;
vp_splits = 0;
that_count = 0;
mod_count = 0;
inf_count = 0;
aux_count = 0;
iqv_count = 0;

// Output sentence length statistics declared in file nlp.h
o_min = 100;
o_max = 0;
o_count = 0;
o_sum = 0;

// ************************************************************
// Correct for periods and commas in numbers of form
// "nnn/CD /., nnn/CD", "nnn/CD ,/, nnn/CD", and "./. CD"
// Also correct for colons in times of form "nn/CD :/: nn/CD"
// Also correct for exclamation and question marks within quotes
// ************************************************************
assert(argc == 3);
fin = fopen(argv[1], "r");
assert(fin != NULL);
fout = fopen("temp.nlp", "w");
assert(fout != NULL);

num = 0;
printf("\n\nFixing numbers in file...\n");
while(fscanf(fin,"%s",buff) == 1){
    i = 0;
    // Extract the word part
    while((token[num][i] = buff[i]) != '/') {
        i++;
    }
    token[num][i] = '\0';
i++;
    // Extract the tag part
    j = 0;
    while((tag[num][j] = buff[i]) != '\0') {
        j++;
        i++;
    }
    tag[num][j] = '\0';
    num++;
    if(num == 3) {
    
}
// Merge "nnn/CD ./ nnn/CD" or ". nnn/CD ,//, nnn/CD" into 1 token if found
if( (token[0][0] == '0') && (token[0][0] <= '9') && (tag[0][0] == 'C') && (tag[0][1] == 'D') &&
    (tag[1][0] == '.') ||
    (tag[1][0] == ',') ||
    (tag[1][0] == ':') ) &&
    (token[2][0] == '0') && (token[2][0] <= '9') && (tag[2][0] == 'C') && (tag[2][1] == 'D'))
{
    merge(0, 3);
    tag[0][0] = 'C';
    tag[0][1] = 'D';
    tag[0][2] = '\0';
    num = 1;
    Zfixnum++;
}

// Else merge ". ./ nnn/CD" if found
else if((end_of_sentence(0) == TRUE) &&
    (tag[1][0] == '.') &&
    (token[2][0] == '0') && (token[2][0] <= '9') && (tag[2][0] == 'C') && (tag[2][1] == 'D'))
{
    merge(1, 2);
    tag[1][0] = 'C';
    tag[1][1] = 'D';
    tag[1][2] = '\0';
    num = 2;
    Zfixnum++;
}

// Else mark an exclamation or quotation mark if followed
// by a quotation mark and either a lower case letter or
// by a proper noun (from which we infer that it is within
// a direct quote)
else if ( ( (token[0][0] == '?') || (token[0][0] == '!') ) &&
    (token[1][0] == '' ) &&
    ( (token[2][0] == 'a') && (token[2][0] <= 'z') )
    ||
    ( (tag[2][0] == 'N') &&
    (tag[2][1] == 'N') &&
    (tag[2][2] == 'P') )
)
{
    tag[0][0] = 'p';
    tag[0][1] = 'q';
    tag[0][2] = '\0';
    Zpq++;

    fprintf(fout, "#s/#s\n", token[0], tag[0]);
    shift_left(1, 3, 1);
    num--;num = 2;
}

// Else no match found, so left shift into output file
else
{

}
fprintf(fout, "%s/%s\n", token[0], tag[0]);
  shift_left(1, 3, 1);
  num = 2;
}

} // end if num

} // end while

if(num >= 1) fprintf(fout, "%s/%s\n", token[0], tag[0]);
if(num == 2) fprintf(fout, "%s/%s\n", token[1], tag[1]);
if(num == 3) printf("\n\n\n*** Fixnum file write error ***\n\n")
fclose(fin);
fclose(fout);

// ********************************************
// Correct for multi-initial identifiers
// (such as "U.S.", "a.m." and "e.g.")
// ********************************************

fin = fopen("temp.nlp","r");
assert(fin != NULL);
fout = fopen("temp2.nlp","w");
assert(fout != NULL);

num = 0;
printf("\nFixing multi-initial identifiers in file...\n");
while(fscanf(fin,"%s",buff) == 1){
  i = 0;

  // Extract the word part
  while((token[num][i] = buff[i]) != '/') {
    i++;
  }
  token[num][i] = '\0';
i++;

  // Extract the tag part
  j = 0;
  while((tag[num][j] = buff[i]) != '\0') {
    j++;
    i++;
  }
  tag[num][j] = '\0';
  num++;

  // Merge initials two at a time
  if(num == 4) {
    // Merge "U/NNP ./. S/NNP ../" into 1 token if found
    if( (token[0][0] == 'U') && (token[0][1] == '\0') &&
        (token[1][0] == 'S') && (token[2][1] == '\0') &&
        (token[3][0] == 'I') )
      merge(0, 4);
  }
tag[0][0] = 'N';
tag[0][1] = 'N';
tag[0][2] = 'P';
tag[0][3] = '\0';
num = 1;
Zfixmult++;

// Else merge "a/.m/." or "p/.m/.".
// into 1 token if found
else if( ( (token[0][0] == 'a') || (token[0][0] == 'p') ) &&
        (token[0][1] == '\0') &&
        (token[1][0] == '.') &&
        (token[2][0] == 'm') && (token[2][1] == '\0') &&
        (token[3][0] == '.') )
{
    merge(0, 4);
tag[0][0] = 'C';
tag[0][1] = 'D';
tag[0][2] = '\0';
num = 1;
Zfixmult++;
}

// Else merge "A/.M/." or "P/.M/.".
// into 1 token if found
else if( ( (token[0][0] == 'A') || (token[0][0] == 'P') ) &&
        (token[0][1] == '\0') &&
        (token[1][0] == '.') &&
        (token[2][0] == 'M') && (token[2][1] == '\0') &&
        (token[3][0] == '.') )
{
    merge(0, 4);
tag[0][0] = 'C';
tag[0][1] = 'D';
tag[0][2] = '\0';
num = 1;
Zfixmult++;
}

// Else merge "B/.C/." into 1 token if found
else if( (token[0][0] == 'B') && (token[0][1] == '\0') &&
        (token[1][0] == '.') &&
        (token[2][0] == 'C') && (token[2][1] == '\0') &&
        (token[3][0] == '.') )
{
    merge(0, 4);
tag[0][0] = 'C';
tag[0][1] = 'D';
tag[0][2] = '\0';
num = 1;
Zfixmult++;
}

// Else merge "A/.D/." into 1 token if found
else if( (token[0][0] == 'A') && (token[0][1] == '\0') &&
        (token[1][0] == '.') &&
        (token[2][0] == 'D') && (token[2][1] == '\0') &&

143
(token[3][0] == '.')
{
    merge(0, 4);
    tag[0][0] = 'C';
    tag[0][1] = 'D';
    tag[0][2] = '\0';
    num = 1;
    Zfixmult++;
}

// Else merge "e/NN ./ g/NN ./" into 1 token if found
else if((token[0][0] == 'e') && (token[0][1] == '\0') &&
        (tag[0][0] == 'N') && (tag[0][1] == 'N') &&
        (tag[1][0] == '.') && (token[2][0] == 'g') &&
        (token[2][1] == '\0') &&
        (tag[2][0] == 'N') &&
        (tag[2][1] == 'N') && (tag[3][0] == '.') )
{
    merge(0, 4);
    tag[0][0] = 'T';
    tag[0][1] = 'N';
    tag[0][2] = '\0';
    num = 1;
    Zfixmult++;
}

// Else merge "i/NN ./ e/NN ./" into 1 token if found
else if((token[0][0] == 'i') && (token[0][1] == '\0') &&
        (tag[0][0] == 'N') && (tag[0][1] == 'N') &&
        (tag[1][0] == '.') && (token[2][0] == 'e') &&
        (token[2][1] == '\0') && (tag[2][0] == 'N') &&
        (tag[2][1] == 'N') && (tag[3][0] == '.') )
{
    merge(0, 4);
    tag[0][0] = 'I';
    tag[0][1] = 'N';
    tag[0][2] = '\0';
    num = 1;
    Zfixmult++;
}

// Else merge "a/NN ./ k/NN ./ a/NN ./" into 1 token,
// in two steps -- this is step #1: merge the leading "a.k." into two tokens: "a.k" and "."
else if((token[0][0] == 'a') && (token[0][1] == '\0') &&
        (tag[1][0] == '.') &&
        (token[2][0] == 'k') && (token[2][1] == '\0') &&
        (tag[3][0] == '.') )
{
    merge(0, 3);
    tag[0][0] = 'Z';
    tag[0][1] = 'C';
    tag[0][2] = 'C';
    tag[0][3] = '\0';

token[1][0] = '.';
token[1][1] = '\0';
tag[1][0] = '.';
tag[1][1] = '\0';

144
num = 2;

/*a.k.a.*/ step #2: merge "a.k." with "a."
else if((token[0][0] == 'a') && (token[0][1] == '.') &&
(token[0][2] == 'k') && (token[0][3] == 'o') &&
(tag[0][0] == 'z') && (tag[0][1] == 'c') &&
(tag[0][2] == 'c') && (tag[0][3] == 'o') &&
(tag[1][0] == '.') &&
(token[2][0] == 'a') && (token[2][1] == 'o') &&
(tag[3][0] == '.
{
merge(0, 4);
tag[0][0] = 'z';
tag[0][1] = 'c';
tag[0][2] = 'c';
tag[0][3] = 'o';
num = 1;
fixmult++;
and++;
}
// Else no match found, so left shift into output file
else {
fprintf(fout, "%s/%s
l",
shift_left(1, 4, 1);
num = 3;
}
} // end if num

} // end while

if(num >= 1) fprintf(fout, "%s/%s
", token[0], tag[0]);
if(num >= 2) fprintf(fout, "%s/%s
", token[1], tag[1]);
if(num == 3) fprintf(fout, "%s/%s
", token[2], tag[2]);
if(num == 4) printf("%s/%s
*** Fixmult file write error ***\n\n")
fclose(fin);
fclose(fout);

// *******************************************************
//
// Commence intelligent editing
//
// *******************************************************

fin = fopen("temp2.nlp", "r");
assert(fin != NULL);
fout = fopen(argv[2],"w");
assert(fout != NULL);

fprintf(fout,"**************************************
***********\n
"DEMETRIOS G. GLINOS

145
"Master's Thesis: "An Intelligent Editor for Natural Language Processing of Unrestricted Text"
"Output file for thesis program "nlp.exe"

"************************************************************
"Input file name > %s\n\narg\n
num = 0;
open_quote = FALSE;

// Read in the next token for the current sentence
while( fscanf(fin, "%s", buff) == 1 )
{
    // Echo it to the screen
    printf("%d %s\n", num, buff);

    // Propagate the open quote from the previous sentence if it
    // still open and not closed by first token of this sentence
    if( (open_quote == TRUE) && (num == 0) && (buff[0] != "'"))
    {
        token[0][0] = '"';
        token[0][1] = '"';
        tag[0][0] = '"';
        tag[0][1] = '"';
        num = 1;
    }

    // Skip the token if it is only the close quote from
    // the previous sentence
    else if( (open_quote == TRUE) && (num == 0) && (buff[0] == "'"))
    {
        open_quote = FALSE;
        num = -1;
    }

    // Else close out a quote if see a quotation mark and
    // there is an open quote
    else if( (open_quote == TRUE) && (buff[0] == '"') )
    {
        open_quote = FALSE;
    }

    // Else start a new quote if see a quotation mark and
    // there is no open quote
    else if( (open_quote == FALSE) && (buff[0] == '"') )
    {
        open_quote = TRUE;
    }

    // If not skipping the token, extract it
    if( num >= 0 )
    {
        // Extract the word part
i = 0;
while( (token[num][i] = buff[i]) != '/' )
{
    i++;
}
token[num][i] = '\0';
i++;

// Extract the tag part
j = 0;
while( (tag[num][j] = buff[i]) != '\0' )
{
    j++;
i++;
}tag[num][j] = '\0';

// Count the word/token pair
num++;nun++; // Analyze sentence if end of sentence reached
if( (tag[num - 1][0] == '.') &&
    (end_of_sentence(num - 2) == TRUE ))
{
    Znum = num;
i_count++;i_sum += Znum;
if( Znum < i_min ) i_min = Znum;
if( Znum > i_max ) i_max = Znum;

Zidm += mark_idioms();Zand += mark_and();Zcoord += mark_coords();Zquote += mark_quotes();Zdirect += mark_d_quotes();Zappos1 += mark_pn_appos();Zappos2 += mark_pn_appos_commas();Zadverb += mark_adverbials();Zindirect += mark_i_quotes();Zwh += mark_wh_words();Zfixin += fix_prep_subconj();Zfutile += mark_futiles();Zparen += mark_paren();Zdash += mark_dashes();Zwhich += split_which();Zpwhich += split_prep_which();Zwith += split_with();Zsubconj += split_subconj();Zbut += split_but();Zds += split_dash();Zpr += split_paren();
Zsc += split_svp_coord();

printf("\nAll marks and splits completed ...");

output_sentence(fout, Znum);
num = 0;

printf("\nSentence has been printed to output file\n\n");
}

// Else mark the period as not a sentence terminator
else if( tag[num - 1][0] == '.' )
{
    tag[num-1][0] = 'p';
    tag[num-1][1] = 'd';
    tag[num-1][2] = '\0';
    Zpd++;
}

else
{
    num = 0;
}

// end while fscanf

// Analyze the residue if the end of file is reached
// and there is more than one token in the buffer
if ( num > 1 )
{
    // Add a closing period
    token[num][0] = '.';
    token[num][1] = '\0';
    tag[num][0] = '.';
    tag[num][1] = '\0';
    num++;

    // And proceed with analysis as before
    Znum = num;

    i_count++;
    i_sum += Znum;
    if( Znum < i_min ) i_min = Znum;
    if( Znum > i_max ) i_max = Znum;

    Zidm += mark_idioms();
    Zand += mark_ands();
    Zcoord += mark_coords();
    Zquote += mark_quotes();
    Zdirect += mark_d_quotes();

    Zsc += split_svp_coord();

    printf("\nAll marks and splits completed ...");

    output_sentence(fout, Znum);
    num = 0;

    printf("\nSentence has been printed to output file\n\n");
}
printf("All marks and splits completed ...");

output_sentence(fout, Znum);
num = 0;

printf("Sentence splitting statistics:

\tNumber of input sentences > %d\n\tMinimum input sentence length > %d\n\tMaximum input sentence length > %d\n\tAverage input sentence length > %5.2f\n\tNumber of output sentences > %d\n\tMinimum output sentence length > %d\n\tMaximum output sentence length > %d\n\tAverage output sentence length > %5.2f\n\n\tNumber of non-splitting CCs/comas marked > %d\n\tNumber of pn appositions (w/o commas) > %d\n\tNumber of pn appositions (w/commas) > %d\n")

fprintf(fout,"\n
Program editing statistics:

\tNumber of nonterminating periods found > %d\n\tNumber of '?' and '!' before close quote > %d\n\tNumber of idioms/collocations marked > %d\n\tNumber of non-splitting CCs/comas marked > %d\n\tNumber of quotation marks marked > %d\n\tNumber of pn appositions (w/o commas) > %d\n\tNumber of pn appositions (w/commas) > %d\n")
\tNumber of prep/conjunctions resolved  > \d\n
\tNumber of 'that' tokens marked  > \d\n
\tNumber of numbers in text fixed  > \d\n
\tNumber of multi-initial identifiers fixed  > \d\n
\tNumber of direct quotation commas marked  > \d\n
\tNumber of indirect quotation commas marked  > \d\n
\tNumber of indirect quotation verbs marked  > \d\n
\tNumber of adverbial commas marked  > \d\n
\tNumber of sub. conj. commas marked  > \d\n
\tNumber of coordination commas marked  > \d\n
\tNumber of wh-word commas marked  > \d\n
\tNumber of futile commas marked  > \d\n
\tNumber of open & close parentheses marked  > \d

\tNumber of modals propagated  > \d\n
\tNumber of infinitives propagated  > \d\n
\tNumber of auxiliaries propagated  > \d\n
\tNumber of comma-which splits  > \d\n
\tNumber of comma-prep-which splits  > \d\n
\tNumber of but-clause splits  > \d\n
\tNumber of dash clause splits  > \d\n
\tNumber of parenthetical clause splits  > \d\n
\tNumber of comma + 'with' + gerund splits  > \d\n
\tNumber of subordinate clause splits  > \d\n
\tNumber of verb phrase coordination splits  > \d\n
\tNumber of sentence coordination splits  > \d\n

fclose(fout);
fclose(fin);

// Clean up temporary files on exiting program
i = remove("temp.nlp");
if(i < 0) printf("\nALERT: You must manually delete temporary file 'TEMP.NLP'");
i = remove("temp2.nlp");
if(i < 0) printf("\nALERT: You must manually delete temporary file 'TEMP2.NLP'");
return(0);
/*********************************************************************************/

DEMETRIOS G. GLINOS

Master's Thesis: "An Intelligent Editor for Natural Language Processing of Unrestricted Text"

Support file "nlpl.c"

*********************************************************************************/

#include "nlp.h"

/*********************************************************************************/

// Compute string length

int d_strlen(char *s)
{
    char *p = s;
    while(*p != '\0') p++;
    return(p-s);
}

/*********************************************************************************/

// Shift token buffer to right "places" number of places for tokens from first to num - 1

int shift_right(int first, int num, int places)
{
    int i = 0, j, count = 0;

    if(places <= 0) return(0);

    for(i = num - 1; i >= first; i--)
    {
        for(j = 0; j < d_strlen(token[i]); j++)
            token[i + places][j] = token[i][j];
        token[i + places][j] = '\0';

        for(j = 0; j < d_strlen(tag[i]); j++)
            tag[i + places][j] = tag[i][j];
        tag[i + places][j] = '\0';
        count++;
    }

    num += places;
    return(count);
}
// Shift token buffer to left "places" number of places for tokens from first to num - 1

int shift_left(int first, int num, int places)
{
    int i = 0, j, count = 0;

    if (places <= 0) return(0);

    for (i = first; i < num; i++)
    {
        for (j = 0; j < d_strlen(token[i]); j++)
            token[i - places][j] = token[i][j];
        token[i - places][j] = '\0';

        for (j = 0; j < d_strlen(tag[i]); j++)
            tag[i - places][j] = tag[i][j];
        tag[i - places][j] = '\0';
        count++;
    }

    num -= places;
    return(count);
}

// Merge "num" number of tokens (including "first") into location of "first"

int merge(int first, int num)
{
    int i, j, k, idex, count = 0;

    idex = d_strlen(token[first]);

    for (i = 1; i < num; i++)
    {
        j = d_strlen(token[first + i]);

        for (k = 0; k < j; k++)
        {
            token[first][idex] = token[first + i][k];
            idex++;
        }
        token[first][idex] = '\0';
        count++;
    }
}
void fix_gerund(int gdex) {
    char *pverb[15] = { "were", "brought", "bought", "came", "ate", "went", "had", "heard", "knew", "read", "saw", "sold", "sang", "stood", "thought" };
    int i, j, glist, glength;

    // Mark the tag for past participle
    tag[gdex][0] = 'v';
    tag[gdex][1] = 'b';
    tag[gdex][2] = 'd';
    tag[gdex][3] = '\0';

    // Search for the listed word on the list of irregular verbs
    glist = -1;
    for ( i = 0; i < 15; i++ )
    {
        j = 0;
        while ( token[gdex][j] == gverb[i][j] )
        {
            if ( gverb[i][j] == '\0' ) glist = i;
            j++;
        }
        if ( glist >= 0 ) break;
    }

    // If verb is on the list of irregular gerund forms,
    // use the corresponding list of past tense forms
    if ( glist >= 0 )
    {
        j = 0;
        while ( ( token[gdex][j] = pverb[glist][j] ) != '\0' )
        {
            j++;
        }
    }

    // Else strip the '-ing' and add '-ed'
    else
    {
        return(count);
    }

    // Convert gerund verb form to past tense
    //**********************************************************************
    void fix_gerund(int gdex) {
        return(count);
    }
glen = d_strlen(token[gdex]);
token[gdex][glen-3] = 'e';
token[gdex][glen-2] = 'd';
token[gdex][glen-1] = '\0';

} // end fix_gerund

// Mark idiomatic expressions and collocations
// ***********************************************

int mark_idioms(void)
{
    char *idiom[21][4] = { "on", "the", "one", "hand" },
                         ["on", "the", "other", "hand" ],
                         ["part", "and", "parcel" ],
                         ["at", "a", "minimum" ],
                         ["as", "a", "result" ],
                         ["more", "than", "ever" ],
                         ["so", "long", "as" ],
                         ["wear", "and", "tear" ],
                         ["by", "the", "time" ],
                         ["for", "so", "long", "as" ],
                         ["all", "but" ],
                         ["compared", "with" ],
                         ["along", "with" ],
                         ["instead", "of" ],
                         ["but", "never" ],
                         ["almost", "all" ],
                         ["or", "so" ],
                         ["from", "whom" ],
                         ["at", "least" ],
                         ["and", "half" ],
                         ["and", "a", "half" ];

    int idex[21] = { 4, 4, 3, 3, 3, 3, 3, 3, 3, 4,
                    2, 2, 2, 2, 2, 2, 2, 2, 2, 3 };  

    int i, j, k, m, msum, mlen, count;
    enum boolean match, kmatch[21];

    // Initialize
    count = 0;

    // Check each token in the sentence to see if it is the
    // start of an idiomatic expression
    i = 0;
    while ( i < Znum )
    {
        // Find the next idiom that fits within the remaining
        // part of the sentence.
match = FALSE;
j = 0;
while ( (j < 21) && (match == FALSE) )
{
    if ( i < (Znum - idex[j]) )
    {

        // Check the token and its followers against the
        // words in the idiomatic expression
        msum = 0;
        for (k = 0; k < idex[j]; k++)
        {

            // Keep scanning the letters of the current
            // idiom word so long as the token matches
            kmatch[k] = TRUE;
            m = 0;
            while ( ( kmatch[k] == TRUE ) &&
                ( ( i + k ) < Znum ) &&
                ( ( token[i+k][m] == idiom[j][k][m] ) ||
                ( token[i+k][m] == ( 'A' +
                ( idiom[j][k][m] - 'a') ) ) ) )
            {
                if ( token[i+k][m] == '\0' ) break;
                m++;
            }

            if ( token[i+k][m] != '\0' )
            {
                kmatch[k] = FALSE;
            }
            else
            {
                msum++;
            }
        }
        // end for k

        // If all words in the expression match, then have
        // a true match
        if ( msum == idex[j] ) match = TRUE;
    }
    // end if i

    if ( match == FALSE ) j++;
}
// end while j

// If matched a complete idiomatic expression, mark all
// matching tokens by adding prefix "i" to their tags
if ( match == TRUE )
{
    for ( k = 0; k < idex[j]; k++)
    {
        mlen = d_strlen( tag[i+k] );
        for ( m = (mlen+1); m > 0; m-- )
            
}
```c
{    tag[i+k][m] = tag[i+k][m-1];
        
tag[i+k][0] = 'i';
    }

    // Record the matching idiom
    count++;

    // Go to end of idiomatic expression
    i += ( idex[j] -1 );
    }
    
    // Continue with tokens past end of idiom, if any
    i++;
    }
    // end while i

    return(count);
    }
    // end mark_idioms

    //******************************************************************************
    // Mark non-sentence splitting coordinating conjunctions and commas
    //******************************************************************************

    int mark_ands(void)
    {    int i, j, prior, next, count = 0;

    // Scan the entire sentence except for the first token
    for(i = 1; i < Znum; i++)
    {
        // Find the next coordinating conjunction
        if ( ( (tag[i][0] == 'C') && (tag[i][1] == 'C') ) ||
            ( token[i][0] == ',' ) )
        {
            // Look at the tag for the next token, if any
            next = i + 1;
            if ( next < Znum )
            {
                // Look at the tag for the prior token or,
                // if the token is preceded by a comma, the
                // word preceding the comma
                prior = i - 1;
                if ( ( prior > 0 ) && ( token[prior][0] == ',' ) )
                {
                    prior--;
                }
            }
        }
    }
```
// Compare the tags for the two tokens
j = 0;
while ( tag[prior][j] == tag[next][j] )
{
    if ( ( tag[prior][j] == '\0' ) ||
         ( tag[next][j] == '\0' ) )
    {
        break;
    }
    j++;
}

// If the tags are identical, then mark
// the CC as a non-sentence splitting CC
if ( ( tag[prior][j] == '\0' ) &&
     ( tag[next][j] == '\0' ) )
{
    tag[i][0] = 'z';
    tag[i][1] = 'C';
    tag[i][2] = 'C';
    tag[i][3] = '\0';

    // And record the marking
    count++;

    // If the marked token is a comma,
    // then also mask it
    if ( token[i][0] == ',' )
    {
        token[i][0] = '[';
        token[i][1] = ',';
        token[i][2] = ']';
        token[i][3] = '\0';
    }

    // If the token after the next token is
    // also a CC, then mark it too
    if ( ( ( next + 1 ) < Znum ) &&
         ( tag[next+1][0] == 'C' ) &&
         ( tag[next+1][1] == 'C' ) )
    {
        tag[next+1][0] = 'z';
        tag[next+1][1] = 'C';
        tag[next+1][2] = 'C';
        tag[next+1][3] = '\0';

        // And record the marking
        count++;
    }
}

} // end if

} // end if next

} // end if tag

} // end for i
return (count);
}

// end mark_ands

// Output the sentence to fit on a page

void output_sentence(FILE *outfile, int num_tokens)
{
    char zbuf[10][100][30], ztag[10][100][7];
    int i, j, k, m, nsize, count, max_len = 70;
    int zdex[10], level, limit;
    int start_skip, end_skip, saw_subj, saw_verb;

    char *adverb[36] = { "Almost",         "almost",
                          "Before",        "before",
                          "During",       "during",
                          "After",        "after",
                          "Only",         "only",
                          "Eventually",   "eventually",
                          "Ultimately",   "ultimately",
                          "None",         "none",
                          "Much",         "much",
                          "Some",         "some",
                          "All",          "all",
                          "Meanwhile",    "meanwhile",
                          "Always",       "always",
                          "Sometime",     "sometime",
                          "Sometimes",    "sometimes",
                          "Never",        "never",
                          "Then",         "then",
                          "Once",         "once"
                      };

    int found_adverb;

    // Initialize
    for ( i = 0; i < 10; i++ ) zdex[i] = 0;
    level = 0;
    limit = 0;

    // Distribute the tokens to separate sentence buffers
    for ( i = 0; i < num_tokens; i++ )
    {
        // Insert next token into token & tag buffers
        // for current output sentence
        j = 0;
        while((zbuf[level][zdex[level]])[j] = token[i][j]) != '\0') j++;
        j = 0;
        while((ztag[level][zdex[level]])[j] = tag[i][j]) != '\0') j++;
        zdex[level]++;
    }
// Go to next level if encounter sentence split, but not if
// it's the first token of the sentence or if it's the
// first token on the current level
if ( ( i > 0 )
    ( zdx[level] > 1 )
    ( token[i][0] != ']' )
    ( tag[i][0] == 'p' )
    ( tag[i][1] == '-' )
) {
    limit++; level = limit;
}

// Go back to previous level if reach end of phrase
else if ( (tag[i][0] == 'x')
    (tag[i][1] == '-')
) {
    level--; if( level < 0 ) level = 0;
}
}

// Perform analysis on the separated sentences
for ( i = 0; i <= limit; i++)
{
    start_skip = 0; end_skip = -1; sawsubj = -1; sawverb = -1;

    // Find the first token in the sentence that is a word
    // (i.e., skip any leading quotation marks, parentheses,
    // or spurious tags)
    while( ( zbuf[i][start_skip][0] == '(' )
        ( zbuf[i][start_skip][0] == ')' )
        ( zbuf[i][start_skip][0] != '[' )
        ( (zbuf[i][start_skip][0] < 'a')
            (zbuf[i][start_skip][0] > 'z') )
        ( (zbuf[i][start_skip][0] < 'A')
            (zbuf[i][start_skip][0] > 'Z') ) )
    {
        start_skip++; } 

    // Check the first word against the list of initial adverbs
    // that are not to be deleted
    found_adverb = -1;
    for ( j = 0; j < 36; j++ )
    {
        k = 0;
        while ( zbuf[i][start_skip][k] == adverb[j][k ] )
        {
            if ( zbuf[i][start_skip][k] == '\0' ) found_adverb = j;
            k++;
        }
    }

159
if ( found_adverb >= 0 ) break;
}

// If the first word is a listed adverb, do not delete it
if( (found_adverb >= 0) && (ztag[i][start_skip][0] == 'R') &&
    (ztag[i][start_skip][1] == 'B')) {
    end_skip = -1;
}

// If it is a leading subordinate conjunction,
// mark it to remove it from the sentence buffer
else if( (ztag[i][start_skip][0] == 'p') &&
    (ztag[i][start_skip][1] == '-') &&
    (ztag[i][start_skip][2] == 'c') &&
    (ztag[i][start_skip][3] == 'j')) {
    end_skip = start_skip;
}

// Skip any immediately following comma, too
if( zbuf[i][end_skip + 1][0] == ',') end_skip++;
}

// Else if it is a leading sequence of adverbs and Wh-adverbs,
// delete them all plus any trailing comma, if they are
// followed by a noun phrase before a verb phrase
else if( (ztag[i][start_skip][0] == 'W') &&
    (ztag[i][start_skip][1] == 'R') &&
    (ztag[i][start_skip][2] == 'B')) ||
    (ztag[i][start_skip][0] == 'R') &&
    (ztag[i][start_skip][1] == 'B')) {
    end_skip = start_skip;

    while( (ztag[i][end_skip + l][0] == 'W') &&
        (ztag[i][end_skip + 1][1] == 'R') &&
        (ztag[i][end_skip + 1][2] == 'B')) ||
        (ztag[i][end_skip + 1][0] == 'R') &&
        (ztag[i][end_skip + 1][1] == 'B')) {
        end_skip++;
    }

// Skip any immediately following comma, too
if( zbuf[i][end_skip + 1][0] == ',') end_skip++;

sawsubj = -1;
sawverb = -1;

for( j = (end_skip + 1); j < zdex[i]; j++)
{
    // Find the first noun phrase preceding a verb phrase,
    // if any
    if((sawsubj < 0) &&
        (((ztag[i][j][0] == 'N') && (ztag[i][j][1] == 'N')) ||
        ((ztag[i][j][0] == 'n') && (ztag[i][j][1] == 'n'))) ||

{ (ztag[i][j][0] == 'E') && (ztag[i][j][1] == 'X')) ||
((ztag[i][j][0] == 'P') && (ztag[i][j][1] == 'R') &&
(ztag[i][j][2] == 'P')) ||
((ztag[i][j][0] == 'P') && (ztag[i][j][1] == 'R') &&
(ztag[i][j][2] == 'P')) ||
((ztag[i][j][0] == 'V') && (ztag[i][j][1] == 'B') &&
(ztag[i][j][2] == 'G')) ||
((ztag[i][j][0] == 'C') && (ztag[i][j][1] == 'D')) )
{
    saw_subj = j;
}

// Find the first verb phrase, if any
else if( (saw_verb < 0) &&
    { (ztag[i][j][0] == 'V') &&
      (ztag[i][j][1] == 'B') &&
      (ztag[i][j][2] != 'G') ||
      { (ztag[i][j][0] == 'M') &&
        (ztag[i][j][1] == 'D') } )
{
    saw_verb = j;
}

} // end for j

// Do not remove any tokens if the verb phrase preceded
// the noun phrase
if( saw_verb < saw_subj ) end_skip = -1;

// Else if it is a leading coordinating conjunction followed
// by a sequence of adverbs and Wh-adverbs, delete them all if
// they are followed by a noun phrase before a verb phrase
else if( ( {ztag[i][start_skip][0] == 'C'} &&
            (ztag[i][start_skip][1] == 'C') ) &&
    { (ztag[i][start_skip + 1][0] == 'W') &&
      (ztag[i][start_skip + 1][1] == 'R') &&
      (ztag[i][start_skip + 1][2] == 'B') } ||
    { (ztag[i][start_skip + 1][0] == 'R') &&
      (ztag[i][start_skip + 1][1] == 'B') } )
{
    end_skip = start_skip + 1;

    while( { (ztag[i][end_skip + 1][0] == 'W') &&
             (ztag[i][end_skip + 1][1] == 'R') &&
             (ztag[i][end_skip + 1][2] == 'B') } ||
           { (ztag[i][end_skip + 1][0] == 'R') &&
             (ztag[i][end_skip + 1][1] == 'B') } )
    {
        end_skip++;
    }
}

// Skip any immediately following comma, too
if( zbuf[i][end_skip + 1][0] == ',') end_skip++;

saw_subj = -1;
saw_verb = -1;

for( j = (end_skip + 1); j < zdex[i]; j++)
{
    // Find the first noun phrase preceding a verb phrase, if any
    if((saw_subj < 0) && (saw_verb < 0) &&
        (ztag[i][j][0] == 'N') && (ztag[i][j][1] == 'N') ||
        (ztag[i][j][0] == 'n') && (ztag[i][j][1] == 'n') ||
        (ztag[i][j][0] == 'E') && (ztag[i][j][1] == 'X') ||
        (ztag[i][j][0] == 'P') && (ztag[i][j][1] == 'R') &&
        (ztag[i][j][2] == 'P') ||
        (ztag[i][j][0] == 'P') && (ztag[i][j][1] == 'P') &&
        (ztag[i][j][2] == 'G') ||
        (ztag[i][j][0] == 'C') && (ztag[i][j][1] == 'D'))
    {
        saw_subj = j;
    }

    // Find the first verb phrase, if any
    else if((ztag[i][j][0] == 'V') &&
        (ztag[i][j][1] == 'B') &&
        (ztag[i][j][2] == 'G') ||
        (ztag[i][j][0] == 'M') &&
        (ztag[i][j][1] == 'D'))
    {
        saw_verb = j;
    }
}

    // Do not remove any tokens if the verb phrase preceded the noun phrase
    if(saw_verb < saw_subj) end_skip = -1;
}

// Else if it is a leading "IN"-word, delete it if it is followed by a noun phrase before a verb phrase
else if( (ztag[i][start_skip][0] == 'I') &&
        (ztag[i][start_skip][1] == 'N'))
{
    end_skip = start_skip;
    saw_subj = -1;
saw_verb = -1;

for( j = (end_skip + 1); j < zdex[i]; j++)
{
    // Find the first noun phrase preceding a verb phrase, if any
    if((saw_subj < 0) && (saw_verb < 0) &&
```c
(((ztag[i][j][0] == 'N') && (ztag[i][j][1] == 'N')) ||
((ztag[i][j][0] == 'n') && (ztag[i][j][1] == 'n')) ||
((ztag[i][j][0] == 'E') && (ztag[i][j][1] == 'E')) ||
((ztag[i][j][0] == 'P') && (ztag[i][j][1] == 'P')) ||
((ztag[i][j][2] == 'P')) ||
((ztag[i][j][0] == 'P') && (ztag[i][j][1] == 'R') &&
(ztag[i][j][2] == 'G') ||
((ztag[i][j][0] == 'V') && (ztag[i][j][1] == 'V')) ||
((ztag[i][j][2] == 'G') ) ||
((ztag[i][j][0] == 'M') &&
(ztag[i][j][1] == 'D') ))
}

{ saw_subj = j;
}

// Find the first verb phrase, if any
else if( (saw_verb < 0) &&
( (ztag[i][j][0] == 'V') &&
(ztag[i][j][1] == 'B') &&
(ztag[i][j][2] != 'G') ) ||
( (ztag[i][j][0] == 'M') &&
(ztag[i][j][1] == 'D') ) )
{
    saw_verb = j;
}

} // end for j

// Do not remove any tokens if the verb phrase preceded
// the noun phrase
if( saw_verb < saw_subj) end_skip = -1;

// Else if it is a leading open or close parenthesis
// simply delete it
else if( (zbuf[i][start_skip][0] == '(') ||
( zbuf[i][start_skip][0] == ')') )
{
    end_skip = start_skip;
}
} // end if

// Remove the marked tokens from the sentence buffer
if( end_skip >= start_skip )
{
    m = end_skip - start_skip + 1;

    for( j = start_skip; j < (zdex[i] - m); j++ )
    {
        k = 0;
        while( (zbuf[i][j][k] = zbuf[i][j + m][k]) != '\0' ) k++;
        k = 0;
        while( (ztag[i][j][k] = ztag[i][j + m][k]) != '\0' ) k++;
    }
    zdex[i] -= m;
}

163
```
Output the sentences one at a time, using wordwrap
fprintf(outfile, "\n\n\n\n"");
for ( i = 0; i <= limit; i++)
{
    // Scan the next token and tag for the sentence
count = max_len;
j = 0;
while ( j < zdex[i] )
{
    // Capitalize the first letter of the first
    // token, if necessary
    if ( ( j == 0) && (zbuf[i][j][0] >= 'a') &&
         (zbuf[i][j][0] <= 'z') )
    {
        zbuf[i][j][0] = 'A' + ( zbuf[i][j][0] - 'a' );
    }

    // Go to next line if token & tag will not fit on the
    // current line, including slash and trailing space
    nsize = d_strlen(zbuf[i][j]) + d_strlen(ztag[i][j]) + 2;
    if ( nsize > count)
    {
        fprintf(outfile, "\n\n\n\n"");
        count = max_len;
    }

    // Print the token, a slash, the tag, and a trailing space
    count -= nsize;
    fprintf(outfile, "%s/%s %s", zbuf[i][j], ztag[i][j]);

    // Get the next token and tag
    j++;
}
// end while

// Separate sentences
fprintf(outfile, "\n\n\n\n"");

// Update output sentence statistics
o_count++;
o_sum += zdex[i];
if ( zdex[i] < o_min ) o_min = zdex[i];
if ( zdex[i] > o_max ) o_max = zdex[i];

} // end for

} // end function

//
******************************************************************************
// Determine if period, exclamation, or question mark
// is the end of a sentence
enum boolean end_of_sentence(int t_index)
{
    char *name[203] = {
        "Inc", "Corp", "Co", "Ltd", "v", "vs",
        "Mr", "Ms", "MRS", "Messrs", "Sr", "Jr", "Dr",
        "Pres", "Secty", "Treas", "Gov",
        "Sgt", "Cpl", "Pvt",
        "No", "So", "etc", "Etc", "viz", "Viz",
        "St", "Rev", "Hon", "Sen", "Rep",
        "D-Ct", "D-Del", "D-Fla", "D-Flor", "D-Ga", "D-Ill", "D-Ind",
        "D-Dak", "D-Oklahoma", "D-Ore", "D-Penn", "D-Pa", "D-Tenn", "D-Tex",
        "D-Ver", "D-Vt", "D-Vir", "D-Va", "D-Wash", "D-Wis", "D-Wyo",
        "R-Ala", "R-Ariz", "R-Ark", "R-Calif", "R-Cal", "R-Colo", "R-Conn",
        "R-Ct", "R-Del", "R-Fla", "R-Flor", "R-Ga", "R-Ill", "R-Ind",
        "R-Minn", "R-Miss", "R-Mo", "R-Mont", "R-Neb", "R-Nev", "R-Car",
        "R-Dak", "R-Oklahoma", "R-Ore", "R-Penn", "R-Pa", "R-Tenn", "R-Tex",
        "R-Ver", "R-Vt", "R-Vir", "R-Va", "R-Wash", "R-Wis", "R-Wyo",
        "Ct", "Del", "Fla", "Flor", "Ga", "Ill", "Ind",
        "Dak", "Okla", "Ore", "Penn", "Pa", "Tenn", "Tex",
        "Ver", "Vt", "Vir", "Va", "Wash", "Wis", "Wyo"
    };

    int i, j, k;

    // Not an end of sentence if index out of range
    if ( t_index < 0 ) return FALSE;

    // Nor if it is a recognized abbreviation
    for(i = 0; i < 203; i++)
    {
    // ...
j = 0;
while (token[t_index][j] == name[i][j])
{
    if (name[i][j] == '\0')
        return FALSE;
    j++;
}
// Otherwise, declare end of sentence
return TRUE;

// ********************************************************************************
// Mark open and close quotation marks
// ********************************************************************************
int mark_quotes(void)
{
    int i, j, count = 0, open_flag = 0;

    for (i = 0; i < Znum; i++)
    {
        // Mark alternating quotation marks starting with open
        if (tag[i][0] == "")
        {
            if (open_flag == 0)
                open_flag = 1;
            tag[i][0] = 'q';
            tag[i][1] = 'o';
            tag[i][2] = '\0';
            count++;
        }
        else
        {
            open_flag = 0;
            tag[i][0] = 'q';
            tag[i][1] = 'c';
            tag[i][2] = '\0';
            count++;
        }
    }

    // And precede all intervening tags by the letter 'q'
    // so no splits will be made within direct quotes
    else if (open_flag == 1)
    {
        for (j = d_strlen(tag[i]) + 1; j >= 1; j--)
            tag[i][j] = tag[i][j - 1];
    }

    return TRUE;
}
end for i

// And insert a close quote at end of sentence if quote remains open there
if(open_flag == 1)
{
  token[i][0] = '"';
  token[i][1] = '\0';
  tag[i][0] = 'q';
  tag[i][1] = 'c';
  tag[i][2] = '\0';
  Znum++;
  count++;
}

return(count);

// Mark proper noun appositions not already set off by commas
*
******************************************************************************
******************************************************************************
******************************************************************************
* // Mark proper noun appositions not already set off by commas
******************************************************************************
******************************************************************************
******************************************************************************
int mark_pn_appos(void)
{

  int i, j, k, m, n, count = 0;
  enum boolean match;

  for(i = 0; i < Znum; i++) {
    if((tag[i][0] == 'N') && (tag[i][1] == 'N') && (tag[i][2] == 'P')) {
      j = 1;
      while(((i + j) < Znum) && (tag[i + j][0] == 'N') && (tag[i + j][1] == 'N') && (tag[i + j][2] == 'P')) {
        j++;
      }
      if(j > 1) { // j += i;
        k = i;
        match = TRUE;
        while((match == TRUE) && (k < j)) {
          match = FALSE;
        }
      }
    }
  }

  return(count);
}
m = 0;
while((match == FALSE) && (m < 38)) {
   n = 0;
   while((match == FALSE) &&(token[k][n] == name[m][n])) {
      if(name[m][n] == '\0'){
         match = TRUE;
      }
      n++;
   }
   m++;
}
if(match == TRUE) k++;
}
if((k > i) && (k < j) && (j < Znum)) {
   shift_right(j, Znum, 1);
   token[j][0] = ',';
   token[j][1] = '\0';
   tag[j][0] = 'c';
   tag[j][1] = 'a';
   tag[j][2] = 'p';
   tag[j][3] = 'c';
   tag[j][4] = '\0';
   shift_right(k, Znum, 1);
   token[k][0] = ',';
   token[k][1] = '\0';
   tag[k][0] = 'c';
   tag[k][1] = 'a';
   tag[k][2] = 'p';
   tag[k][3] = 'o';
   tag[k][4] = '\0';
   for(m = i; m < k; m++) {
      tag[m][0] = 'n';
      tag[m][1] = 'n';
      tag[m][2] = 'p';
      tag[m][3] = 'a';
      tag[m][4] = '\0';
   }
   count++;
}
return(count);

//
// ************************************************************************
// Mark commas used to set off np appositions
//************************************************************************

int mark_pn_appos_commas(void)
{
   int idex, jdex, count = 0;
   return(count);
enum boolean found_close, saw_np, saw_vbg, saw_other_verb;
enum boolean push;

idex = 0;
while(idex < Znum) {
  // Find next comma preceded by a noun or by a noun and period
  if((idex > 0) && (token[idex][0] == ',')) &&
  
  { 
    find_close = FALSE;
    saw_np = FALSE;
    saw_vbg = FALSE;
    saw_other_verb = FALSE;

    // Scan to next comma or clause terminator
    idex = idex + 1;
    while((idex < Znum) && (found_close == FALSE))
    { 
      if( (token[idex][0] == ',' || (token[idex][0] || (token[idex][0] == ';')) ||
          (token[idex][0] == '?')) || (token[idex][0] == '!'))
      
      { 
        found_close = TRUE;
      }
    }

    // Check for noun and verb phrases
    else
    { 
      if( ((tag[idex][0] == 'N') && (tag[idex][1] == 'N')) ||
          ((tag[idex][0] == 'n') && (tag[idex][1] == 'n'))) 
      
      { 
        saw_np = TRUE;
      }
    else if((tag[idex][0] == 'V') && (tag[idex][1] == 'B') && (tag[idex][2] == 'G'))
    
      { 
        saw_vbg = TRUE;
      }
    else if( (tag[idex][0] == 'V') &&
             (tag[idex][1] == 'B') &&
             (tag[idex][2] != 'G') ) ||
             ( (tag[idex][0] == 'M') &&
             (tag[idex][1] == 'D') ) )
    
      { 
        saw_other_verb = TRUE;
      }
    }
if (found_close == FALSE) jdex++;

// Mark the noun phrase apposition if found
if ( ((saw_np == TRUE) || (saw_vbg == TRUE))
    && (saw_otherverb == FALSE))
{
    // Determine if apposition is astride a coordination
    push = FALSE;
    if ( (tag[idex][0] == 'c') && (tag[idex][1] == 'c') &&
        (tag[idex][2] == 'r') && (tag[idex][3] == 'd') )
    {
        push = TRUE;
    }

    // Mark the comma introducing the apposition
    tag[idex][0] = 'c';
    tag[idex][1] = 'a';
    tag[idex][2] = 'p';
    tag[idex][3] = 'o';
    tag[idex][4] = '\0';

    // And mark any closing comma with the apposition
    if ( (jdex < Znum) && (token[jdex][0] == ',') &&
        (push == FALSE) )
    {
        tag[jdex][0] = 'c';
        tag[jdex][1] = 'c';
        tag[jdex][2] = 'r';
        tag[jdex][3] = 'd';
        tag[jdex][4] = '\0';
    }

    // Or, if "pushing" the coordination, mark the
    // closing comma with the coordination
    else if ( (jdex < Znum) && (token[jdex][0] == ',') &&
        (push == TRUE) )
    {
        tag[jdex][0] = 'c';
        tag[jdex][1] = 'c';
        tag[jdex][2] = 'r';
        tag[jdex][3] = 'd';
        tag[jdex][4] = '\0';
    }

    // But if the end of the clause is not a comma and
    // if also "pushing" a coordination, insert a comma
    else if ( (jdex < Znum) && (token[jdex][0] != ',') &&
        (push == TRUE) )
    {
        shift_right (jdex, Znum, 1);
        tag[jdex][0] = 'c';
        tag[jdex][1] = 'c';
        tag[jdex][2] = 'r';
tag[jdex][3] = 'd';
tag[jdex][4] = '\0';
}

// And record the split
count++;

} // end if
idex = jdex;
}

else
{
    idex = idex + 1;
} // end if idex

} // end while idex
return(count);
}
DEMETRIOS G. GLINOS

Master's Thesis: "An Intelligent Editor for Natural Language Processing of Unrestricted Text"

Support file "nlp2.c"

************************************************************************

#include "nlp.h"

/**************************************************************************

// Resolve IN preposition/subordinate conjunction tag from Brill tagger
//**************************************************************************

int fix_prep_subconj(void)
{
    int i, j, k, idx, jdx, count = 0;
    int saw_np, saw_verb, saw_vbg;
    enum boolean match, saw_end;

    char *name[30] = {
        "after", "although", "as", "because", "before", "but", "except",
        "inasmuch", "once", "rather", "save", "seeing", "since", "so",
        "while",
        "After", "Although", "As", "Because", "Before", "But", "Except",
        "Inasmuch", "Once", "Rather", "Save", "Seeing", "Since", "So",
        "While"
    };

    // Scan entire input sentence buffer
    idx = 0;
    while(idx < Znum)
    {
        // Find the next "IN" tag that starts the sentence
        // or follows a comma, semicolon, colon, or comma-quote
        if (( tag[idx][0] == 'I' ) && ( tag[idx][1] == 'N' ) &&
            ( ( idx == 0 ) ||
            ( token[idx-1][0] == ',' ) ||
            ( token[idx-1][0] == ';' ) ||
            ( token[idx-1][0] == ':' ) ||
            ( ( token[idx-1][0] == '"' ) &&
            ( token[idx-2][0] == ',' ) )
        )
        {
            // Check it against the list of subordinating conjunctions
            match = FALSE;
            for(i = 0; i < 30; i++)
            {
                //...
            }
        }
    }

    return count;
}

j = 0;
while (token[index][j] == name[i][j])
{
    if (name[i][j] == '\0') match = TRUE;
    j++;
}

// Found it on the list
if(match == TRUE)
{
    saw_np = -1;
    saw_vbg = -1;
    saw_verb = -1;
    saw_end = FALSE;
    index = index + 1;

    // Scan until next unmarked comma (or end of sentence)
    // for noun & verb phrases
    while( (index < Znum) && (saw_end == FALSE) )
    {
        if( (tag[index][0] == ',') || (tag[index][0] == '.') )
        {
            saw_end = TRUE;
        }
        else
        {
            // Check if see a subject
            if ( (saw_np < 0) &&
            ( (tag[index][0] == 'N') &&
                (tag[index][1] == 'N') ) ||
                (tag[index][0] == 'n') &&
                (tag[index][1] == 'n') ) ||
                (tag[index][0] == 'E') &&
                (tag[index][1] == 'X') ) ||
                (tag[index][0] == 'P') &&
                (tag[index][1] == 'R') &&
                (tag[index][2] == 'P') ) ||
                (tag[index][0] == 'P') &&
                (tag[index][1] == 'R') &&
                (tag[index][2] == 'P') &&
                (tag[index][3] == '$' ) )
            {
                saw_np = index;
            }

            // Check if see a gerund (which also counts // as a subject, here)
            else if ( (saw_vbg < 0) &&
            (tag[index][0] == 'V') &&
            (tag[index][1] == 'B') &&
            (tag[index][2] == 'G') )
            {
                saw_vbg = index;
            }
        }
    }

    // Check if see a verb

else if ((saw_verb < 0) &&
        ((tag[jdex][0] == 'V') &&
         (tag[jdex][1] == 'B') &&
         (tag[jdex][2] != 'G')) ||
        ((tag[jdex][0] == 'M') &&
         (tag[jdex][1] == 'D')))
{
    saw_verb = jdex;
}

// Keep checking if the end of the clause has
// not been found
if (saw_end == FALSE) jdex++;

// If found a clause (i.e. subject and verb in that
// order), then mark the token as a subordinating
// conjunction
if ((saw_verb > 0) &&
    ((saw_np > 0) && (saw_np < saw_verb)) ||
    (saw_vbg > 0) && (saw_vbg < saw_verb)))
{
    tag[idx][0] = 's';
    tag[idx][1] = 'c';
    tag[idx][2] = 'n';
    tag[idx][3] = 'j';
    tag[idx][4] = '\0';
    count++;

    // and mark the previously unmarked comma, if any,
    // at the end of the clause
    if (tag[idx][0] == ',')
    {
        tag[idx][0] = 'c';
        tag[idx][1] = 's';
        tag[idx][2] = 'u';
        tag[idx][3] = 'b';
        tag[idx][4] = '\0';
        csub_count++;
    }
}

// Otherwise, mark the token as a preposition
else
{
    tag[idx][0] = 'p';
    tag[idx][1] = 'r';
    tag[idx][2] = 'e';
    tag[idx][3] = 'p';
    tag[idx][4] = '\0';
    count++;
}

// Token not found on coordinating conjunction list -- deal with
// special cases

// Special case 1: token is "that"
else if ((token[idex][0] == 't') && (token[idex][1] == 'h')
        && (token[idex][2] == 'a') && (token[idex][3] == 't'))
{
    tag[idex][0] = 't';
    tag[idex][1] = 'h';
    tag[idex][2] = 'a';
    tag[idex][3] = 't';
    tag[idex][4] = '\0';
    that_count++;
}

// Otherwise, mark the token as a preposition
else
{
    tag[idex][0] = 'p';
    tag[idex][1] = 'r';
    tag[idex][2] = 'e';
    tag[idex][3] = 'p';
    tag[idex][4] = '\0';
    count++;
}

} // end if

// NOTE: If the "IN" token is not introducing a clause, we
// leave it alone and do not resolve the 'IN' tag.

// Go on to the next token
idex = idex + 1;

} // end while

return(count);

//
**********************************************************************
// Mark commas that open or close direct quotes
//
**********************************************************************

int mark_d_quotes(void)
{
    int i, count = 0;

    for(i = 0; i < Znum; i++)
    {
        if((i > 0) && (tag[i][0] == 'q') && (tag[i][1] == 'o')
                && (tag[i][2] == '\0') && (tag[i - 1][0] == ','))
            {
                tag[i - 1][0] = 'c';
                tag[i - 1][1] = 'q';
            }
int mark_i_quotes(void)
{
    char *name[18] =
    {
    };

    int i, j, k, m, match, count = 0;
    enum boolean saw_np;

    // Start from beginning of sentence
    i = 0;
    while ( i < (Znum - 1) )
    {
        j = i + 1;
        match = -1;

        // Scan until the next punctuation mark
        while((j < Znum) && (match < 0) && (token[j][0] != ',')
              && (token[j][0] != '.') && (token[j][0] != ';')
              && (token[j][0] != ':') && (token[j][0] != '?')
              && (token[j][0] != '!'))
        {

            if ((i > 0) && (tag[i][0] == 'q') && (tag[i][1] == 'c')
                && (tag[i][2] == '\0') && (tag[i - 1][0] == ','))
            {
                tag[i - 1][0] = 'c';
                tag[i - 1][1] = 'q';
                tag[i - 1][2] = 'd';
                tag[i - 1][3] = 'c';
                tag[i - 1][4] = '\0';
                count++;
            }
        }
        return (count);
    }

    // Mark indirect quote verbs and commas that precede indirect quotes
    //**********************************************************************
    // Mark indirect quote verbs and commas that precede indirect quotes
    //**********************************************************************

// Check to see if the token is on the list of indirect quotation verbs
for (k = 0; k < 18; k++)
{
    m = 0;
    while (token[j][m] == name[k][m])
    {
        if (name[k][m] == '\0') match = j;
        m++;
    }
}

// If found a match, mark the verb and determine if there is a noun phrase that follows the verb
if (match >= 0)
{
    // Mark the verb and count it
    for (m = (d_strlen(tag[j]) + 1); m > 0; m-- )
    {
        tag[j][m] = tag[j][m-1];
    }
    tag[j][0] = 'm';
    iqv_count++;

    // Check for noun phrase
    saw_np = FALSE;
    k = j + 1;
    while( (k < Znum) && (saw_np == FALSE) &&
            (token[k][0] != ',' && (token[k][0] != '.') &&
             (token[k][0] != ';') && (token[k][0] != ':')) &&
            (token[k][0] != '?') )
    {
        if ( ( (tag[k][0] == 'N') && (tag[k][1] == 'N') ) ||
             ( (tag[k][0] == 'n') && (tag[k][1] == 'n') ) ||
             ( (tag[k][0] == 'E') && (tag[k][1] == 'X') ) ||
             ( (tag[k][0] == 'P') && (tag[k][1] == 'R') &&
                (tag[k][2] == 'P') ) ||
             ( (tag[k][0] == 'P') && (tag[k][1] == 'R') ) &&
             (tag[k][2] == 'P') ) ||
             ( (tag[k][0] == 'V') && (tag[k][1] == 'B') &&
                (tag[k][2] == 'G') ) ||
             ( (tag[k][0] == 'C') && (tag[k][1] == 'D') ) )
        {
            saw_np = TRUE;
        }
    }
    k++;
}

// Mark the leading comma, if any, if it has not been marked already and no noun phrase has been seen
if ( (i > 0 ) && (tag[i][0] == ',' )
    ( saw_np == FALSE ) )
{
    tag[i][0] = 'c';
    tag[i][1] = 'q';
}
```c
int mark_adverbials(void) {
    int i, j, count = 0;

    for(i = 0; i < (Znum - 1); i++) {
        if((tag[i][0] == ',' && (tag[i + 1][0] == 'R')
            && (tag[i + 1][1] == 'B' && (tag[i + 1][2] == '\0')))
            tag[i][0] = 'c';
        tag[i][1] = 'a';
        tag[i][2] = 'v';
        tag[i][3] = 'i';
        tag[i][4] = '\0';
        count++;
        j = i + 2;
        while((j < Znum) && (tag[j][0] != ',') && (tag[j][0] != '.')
            && (tag[j][0] != ';') && (tag[j][0] != ':')
            && (tag[j][0] != '?') && (tag[j][0] != '!')) {
            j++;
        }
        if(tag[j][0] == ',') {
            tag[j][0] = 'c';
            tag[j][1] = 'a';
            tag[j][2] = 'v';
            tag[j][3] = 'c';
            tag[j][4] = '\0';
            count++;
        }
    }
    return(count);
}
```

int split_subconj(void) {
    int i, j, count = 0;
    enum boolean saw_subj, saw_verb, saw_scnj, saw_that, saw_indirect;
    saw_subj = FALSE; saw_verb = FALSE; saw_scnj = FALSE; saw_that = FALSE; saw_indirect = FALSE;

    // Scan the entire sentence buffer
    for(i = 0; i < (Znum -1); i++) {
        // Check if the main clause has the words "that", "neither" or "nor" along the way
        if( ( (token[i][0] == 't') && (token[i][1] == 'h') && (token[i][2] == 'a') && (token[i][3] == 't') && (token[i][4] == '\0') ) ||
            ( (token[i][0] == 'n') && (token[i][1] == 'o') && (token[i][2] == 'r') && (token[i][3] == '\0') ) ||
            ( (token[i][0] == 'n') && (token[i][1] == 'e') && (token[i][2] == 'i') && (token[i][3] == 't') && (token[i][4] == 'h') && (token[i][5] == 'e') && (token[i][6] == 'r') && (token[i][7] == '\0') ) )
            { saw_that = TRUE; }
        }

        // Check for marked indirect quote verbs along the way
        else if( ( saw_indirect == FALSE ) && ( tag[i][0] == 'm' ) && (tag[i][1] == 'V') )
            { saw_indirect = TRUE; }
        }

        // Check for noun phrase along the way
        else if( ( saw_subj == FALSE ) &&
            ( (tag[i][0] == 'N') && (tag[i][1] == 'N') ) ||
            ( (tag[i][0] == 'n') && (tag[i][1] == 'n') ) ||
            ( (tag[i][0] == 'E') && (tag[i][1] == 'X') ) ||
            ( (tag[i][0] == 'P') && (tag[i][1] == 'R') && (tag[i][2] == 'P') ) ||
            ( (tag[i][0] == 'P') && (tag[i][1] == 'R') && (tag[i][2] == 'P') ) ||
            ( (tag[i][0] == 'V') && (tag[i][1] == 'B') && (tag[i][2] == 'G') ) ||
            ( (tag[i][0] == 'C') && (tag[i][1] == 'D') ) )
            { }
saw_subj = TRUE;
}

// Check for verb phrase, too
else if( ( saw_verb == FALSE ) &&
         ( (tag[i][0] == 'V') &&
            (tag[i][1] == 'B') &&
            (tag[i][2] != 'G') &&
            (tag[i][2] != 'N') ) ||
         ( (tag[i][0] == 'M') &&
            (tag[i][1] == 'D') ) )
{
    saw_verb = TRUE;
}

// Find the next subordinating conjunction, if any
// but not the word "if"
else if( (tag[i][0] == 's') &&
        (tag[i][1] == 'c') &&
        (tag[i][2] == 'n') &&
        (tag[i][3] == 'j') &&
        (tag[i][4] == '0') ) &&
        ( (token[i][0] != 'i') ||
            (token[i][1] != 'f') ||
            (token[i][2] != '0') ) &&
        ( (token[i][0] != 'I') ||
            (token[i][1] != 'f') ||
            (token[i][2] != '0') )
{

    // If a complete clause (noun and verb phrases)
    // has already been seen, and if the token is
    // not the first word of the sentence, mark a
    // subordinate clause split, but not if did not
    // see indirect quote verb but did see "that",
    // "neither" or "nor" before the CC or after the
    // CC and before the closing comma, if any
    if ( (i > 1) &&
        (saw_subj == TRUE) &&
        (saw_verb == TRUE) &&
        ( (saw_that == FALSE) || (saw_indirect == TRUE) )
    )
    {
        // Mark the corresponding comma that ends the clause,
        // if any, unless "that", "neither" or "nor" are seen
        // and indirect quote verb not seen
        saw_end = FALSE;
        saw_that = FALSE;
        saw_indirect = FALSE;
        j = i + 1;
        while ( ( j < (Znum - 1) ) &&
                (saw_end == FALSE) &&
                ( (saw_that == FALSE) ||
                  (saw_indirect == TRUE) ) )
        {
            // Mark the end of the clause, if found
            if ( (tag[j][0] == 'c') &&

(tag[j][1] == 's') &&
(tag[j][2] == 'u') &&
(tag[j][3] == 'b') &&
(tag[j][4] == '\0')
{
    tag[j][0] = 'x';
    if ( i < 2 ) tag[j][0] = 'p';
    tag[j][1] = '-';
    tag[j][2] = 'c';
    tag[j][3] = 'j';
    tag[j][4] = '\0';
    if ( i == 0 ) count++;
    saw_end = TRUE;
}

// Else check for "that", "neither" or "nor"
else if ( ( (token[j][0] == 't') &&
            (token[j][1] == 'h') &&
            (token[j][2] == 'e') &&
            (token[j][3] == 't') &&
            (token[j][4] == '\0') ) ||
          ( (token[j][0] == 'n') &&
            (token[j][1] == 'o') &&
            (token[j][2] == 'r') &&
            (token[j][3] == '\0') ) ||
          ( (token[j][0] == 'n') &&
            (token[j][1] == 'e') &&
            (token[j][2] == 'i') &&
            (token[j][3] == 't') &&
            (token[j][4] == 'h') &&
            (token[j][5] == 'e') &&
            (token[j][6] == 'r') &&
            (token[j][7] == '\0') ) )
{
    saw_that = TRUE;
    j++;
}

// Else check for marked indirect quote verbs
else if( ( saw_indirect == FALSE ) &&
         ( tag[i][0] == 'm' ) && (tag[i][1] == 'V') )
{
    saw_indirect = TRUE;
    j++;
}

// Else continue scanning
else
{
    j++;
}

} // end while j

// And change the CC into a split marker
// unless "that", "neither" or "nor" are seen
// while indirect quote verb not seen
if ( ( saw_that == FALSE ) || ( saw_indirect == TRUE ) )
{
    token[i][0] = '.';
    token[i][1] = '\0';
    tag[i][0] = 'p';
    tag[i][1] = '-';
    tag[i][2] = 'c';
    tag[i][3] = 'j';
    tag[i][4] = '\0';
    if ( (i > 1) count++;
}
}

// Else if it is the first token or follows a double quotation mark, delete it from the buffer, unless "that", "neither" or "nor" were seen while indirect quote verb not seen
else if ( ( saw_that == FALSE ) ||
          ( saw_indirect == TRUE ) ) &&
          ( i == 0 ) ||
          ( (i == 1) && (tag[0][0] == '"') )
{
    // Mark the corresponding comma that ends the clause, if any, unless "that", "neither" or "nor" are seen while indirect quote verb not seen
    saw_end = FALSE;
    saw_that = FALSE;
    saw_indirect = FALSE;
    j = i + 1;
    while ( ( j < ( Znum - 1 ) ) &&
            ( saw_end == FALSE ) &&
            ( ( saw_that == FALSE ) ||
              ( saw_indirect == TRUE ) ) )
    {
        // Mark the end of the clause, if found
        if ( (tag[j][0] == 'c') &&
             (tag[j][1] == 's') &&
             (tag[j][2] == 'u') &&
             (tag[j][3] == 'b') &&
             (tag[j][4] == '\0') )
        {
            tag[j][0] = 'x';
            if ( ( i < 2 ) tag[j][0] = 'p';
                tag[j][1] = '-';
                tag[j][2] = 'c';
                tag[j][3] = 'j';
                tag[j][4] = '\0';
            if ( (i == 0) count++;
                saw_end = TRUE;
            }
        }
    }
    // Else check if see "that", "neither" or "nor" else if ( (token[j][0] == 't') &&

(token[j][1] == 'h') &&
(token[j][2] == 'a') &&
(token[j][3] == 't') &&
(token[j][4] == '\0')

(token[j][0] == 'n') &&
(token[j][1] == 'e') &&
(token[j][2] == 'i') &&
(token[j][3] == 't') &&
(token[j][4] == 'h') &&
(token[j][5] == 'e') &&
(token[j][6] == 'r') &&
(token[j][7] == '\0')

saw_that = TRUE;

if (saw_indirect == FALSE) &&
    (tag[j][0] == 'm') && (tag[j][1] == 'V')

saw_indirect = TRUE;
j++;}

// Else check for marked indirect quote verbs
else if ((saw_indirect == FALSE) &&
          (tag[j][0] == 'm') && (tag[j][1] == 'V'))

saw_indirect = TRUE;
j++;}

// Else continue scanning
else
{
j++;}
}

// End while j

// And delete the leading CC unless
// "that", "neither" or "nor" are seen
// while indirect quote verb not seen
if ((saw_that == FALSE) || (saw_indirect == TRUE))

shift_left(i+1, Znum, 1);

// End if sub-conj starts the sentence

else
{
saw_end = FALSE;
j = i + 1;
}

// Begin searching for the next subordinate

183
// conjunction after the closing comma, if any,  
// else after the the token just examined 
  i = j;

} // end if

}; // end for

return (count);

}

int mark_wh_words(void)
{
  int i, count = 0;

  for (i = 0; i < (Znum -1); i++) {
    if ( (tag[i][0] == 'I') &&
        ( (tag[i + 1][0] == 'W') &&
        (tag[i + 1][1] == 'D') &&
        (tag[i + 1][2] == 'T') ) ||
        ( (tag[i + 1][0] == 'W') &&
        (tag[i + 1][1] == 'P') )
    )
    {
      tag[i][0] = 'c';
      tag[i][1] = 'r';
      tag[i][2] = 'e';
      tag[i][3] = 'l';
      tag[i][4] = '\0';
      count++;
    }
  }
  return (count);
}

int mark_coords(void)
{
  char *name[7] = {"and", "or", "nor", "but", "yet", "plus", "less"};
  enum boolean match, saw_verb;
  int i, j, k, m, count = 0;

  // Find next unmarked comma
  for (i = 0; i < (Znum -1); i++)
```c
if(tag[i][0] == ',')
{
    // Then scan to end of sentence for coordinating conjunction
    match = FALSE;
    j = i + 1;
    while( (j < Znum) && (match == FALSE) &&
        (tag[j][0] != '.') &&
        (token[j][0] != ';') &&
        (token[j][0] != ':') &&
        (token[j][0] != '?') &&
        (token[j][0] != '!') )
    {
        for(k = 0; k < 7; k++)
        {
            m = 0;
            while(token[j][m] == name[k][m])
            {
                if(name[k][m] == '\0')
                {
                    match = TRUE;
                }
                m++;
            }
        }
    }
    // Found a coordinating conjunction
    if(match == TRUE)
    {
        // Now rescan but only until the next comma,
        // coordinating conjunction, semicolon, colon,
        // or end of sentence for verb forms, ignoring
        // apposition commas
        saw_verb = FALSE;
        k = i + 1;
        while( (k < Znum) && (k < j) &&
            (saw_verb == FALSE) &&
            (tag[k][0] != '.') &&
            (token[k][0] != ';') &&
            (token[k][0] != ':') &&
            (token[k][0] != '?') &&
            (token[k][0] != '!') )
        {
            if( (tag[k][0] == 'V') &&
                (tag[k][1] == 'B') &&
                (tag[k][1] != 'G') )
            {
                saw_verb = TRUE;
            }
        }
    }
}
```
// If there is a verb in the clause, mark the preceding comma as a coordination
if (saw_verb == TRUE )
{
    tag[i][0] = 'c';
    tag[i][1] = 'c';
    tag[i][2] = 'r';
    tag[i][3] = 'd';
    tag[i][4] = '\0';
    count++;
}

} // end if match
j++;

} // end while j

} // end if tag

} // end for i

return(count);


//

******************************************************************************
// Mark any remaining untagged commas as "futile"
//
******************************************************************************

int mark_futiles(void)
{
    int i, count = 0;

    for(i = 0; i < Znum; i++)
    {
        if(tag[i][0] == ',')
        {
            tag[i][0] = 'c';
            tag[i][1] = 'f';
            tag[i][2] = '\0';
            count++;
        }
        return(count);
    }

    //
    ********************************************************************************
    // Mark open and close parentheses
    //
    ********************************************************************************

186
int mark_paren(void)
{
    int i, count = 0, open_flag = 0;

    for(i = 0; i < Znum; i++)
    {
        if(token[i][0] == '(')
        {
            tag[i][0] = 'p';
            tag[i][1] = 'r';
            tag[i][2] = 'n';
            tag[i][3] = 'c';
            tag[i][4] = '\0';
            open_flag = 1;
            count++;
        }
        else if (token[i][0] == ')
        {
            tag[i][0] = 'p';
            tag[i][1] = 'r';
            tag[i][2] = 'n';
            tag[i][3] = 'c';
            tag[i][4] = '\0';
            open_flag = 0;
            count++;
        }
    }

    // And insert a close parenthesis before period
    // at end of sentence if parens still open there
    if(open_flag == 1)
    {
        i = 0;
        while(token[Znum - 1][i] != '\0')
        {
            token[Znum][i] = token[Znum - 1][i];
            i++;
        }
        token[Znum][i] = '\0';
        i = 0;
        while(tag[Znum - 1][i] != '\0')
        {
            tag[Znum][i] = tag[Znum - 1][i];
            i++;
        }
        tag[Znum][i] = '\0';
        token[Znum - 1][0] = '});'
        token[Znum - 1][1] = '\0';
        tag[Znum - 1][0] = 'p';
        tag[Znum - 1][1] = 'r';
        tag[Znum - 1][2] = 'n';
        tag[Znum - 1][3] = 'c';
        tag[Znum - 1][4] = '\0';
    }
int mark_dashes(void)
{
    int i, count = 0, open_flag = 0;

    for(i = 0; i < Znum; i++)
    {
        if( (open_flag == 0) && (token[i][0] == '-') &&
            (token[i][1] == '-'))
        {
            tag[i][0] = 'd';
            tag[i][1] = 's';
            tag[i][2] = 'h';
            tag[i][3] = 'o';
            tag[i][4] = '\0';
            open_flag = 1;
            count++;
        }
        else if( (open_flag == 1) && (token[i][0] == '-') &&
            (token[i][1] == '-'))
        {
            tag[i][0] = 'd';
            tag[i][1] = 's';
            tag[i][2] = 'h';
            tag[i][3] = 'c';
            tag[i][4] = '\0';
            open_flag = 0;
            count++;
        }
    }

    // And insert a close dash before period
    // at end of sentence if dash still open there
    if(open_flag == 1)
    {
        i = 0;
        while(token[Znum - 1][i] != '\0')
        {
            token[Znum][i] = token[Znum - 1][i];
            i++;
        }
        token[Znum][i] = '\0';

        i = 0;
        while(tag[Znum - 1][i] != '\0')
        {
tag[Znum][i] = tag[Znum - 1][i];
    i++;
}
tag[Znum][i] = '\0';
token[Znum - 1][0] = '-';
token[Znum - 1][1] = '-';
token[Znum - 1][2] = '\0';
tag[Znum - 1][0] = 'd';
tag[Znum - 1][1] = 's';
tag[Znum - 1][2] = 'h';
tag[Znum - 1][3] = 'c';
tag[Znum - 1][4] = '\0';
Znum++;
count++;
}
return(count);
/* Split off independent clauses within parentheses */

int split_paren(void)
{
    int i, j, count = 0;
    int saw_subj, saw_verb, saw_end;
    enum boolean saw_to;

    // Locate next open parentheses, if any
    for ( i = 0; i < Znum; i++ )
    {
        if ( token[i][0] == '(' )
    {

            // Search until find corresponding close parenthesis
            j = i + 1;
            saw_subj = 0;
            saw_verb = 0;
            saw_end = 0;
            saw_to = FALSE;
            while ( (j < Znum) && (saw_end == 0) )
            {
                if ( token[j][0] != ')' )
    {

                    // Find the first noun phrase preceding a verb
                    // phrase, if any
                    if ( (saw_subj == 0) && (saw_verb == 0) &&
                        (((tag[j][0] == 'N') && (tag[j][1] == 'N')) ||
                        ((tag[j][0] == 'n') && (tag[j][1] == 'n')) ||
                        ((tag[j][0] == 'E') && (tag[j][1] == 'E')) ||
                        ((tag[j][0] == 'P') && (tag[j][1] == 'P')) ||
                        (tag[j][2] == 'P') ) ||
                        ((tag[j][0] == 'P') && (tag[j][1] == 'P')) )
    {
                    saw_subj = j;
    }
// Find the first verb phrase, if any, which must precede the word "to"
else if ( (saw_verb == 0) &&
          (saw_to == FALSE) &&
          (tag[j][0] == 'V') &&
          (tag[j][1] == 'B') &&
          (tag[j][2] != 'G') &&
          (tag[j][2] != 'N') ) ||
          (tag[j][0] == 'M') &&
          (tag[j][1] == 'D') ) )
{
    saw_verb = j;
}

// Else check if token is the word "to"
else if ( !saw_to == FALSE) &&
          (token[j][0] == 't') &&
          (token[j][1] == 'T') ) &&
          (token[j][2] == '0') &&
          (token[j][2] == '\0') )
{
    saw_to = TRUE;
}

// Continue with the next token
j++;

// Found close parenthesis
else
{
    saw_end = j;
}

} // end while

// Count and mark the independent clause if found
if( (saw_subj > 0) &&
    (saw_verb > 0) &&
    (saw_subj < saw_verb) )
{
    count++;

    tag[i][0] = 'p';
    tag[i][1] = '-';
    tag[i][2] = 'p';
    tag[i][3] = 'r';
    tag[i][4] = '\0';

    tag[saw_end][0] = 'x';
    tag[saw_end][1] = '-';
    tag[saw_end][2] = 'p';
    tag[saw_end][3] = 'r';
    tag[saw_end][4] = '\0';
}

i = j;
```c
int split_dash(void)
{
    int i, j, count = 0;
    int saw_subj, saw_verb, saw_end;
    enum boolean saw_to;

    // Locate next open dash, if any
    for (i = 0; i < Znum; i++)
    {
        if ((tag[i][0] == 'd') && (tag[i][1] == 's') &&
            (tag[i][2] == 'h') && (tag[i][3] == 'o'))
        {
            // Search until find corresponding close dash
            j = i + 1;
            saw_subj = 0;
            saw_verb = 0;
            saw_end = 0;
            saw_to = FALSE;
            while ((j < Znum) && (saw_end == 0))
            {
                if ((tag[j][0] != 'd') || (tag[j][1] != 's') ||
                    (tag[j][2] != 'h') || (tag[j][3] != 'o'))
                {
                    // Find the first noun phrase preceding a verb phrase,
                    // if any
                    if ((saw_subj == 0) && (saw_verb == 0) &&
                        (tag[j][0] == 'N') && (tag[j][1] == 'N') ||
                        (tag[j][0] == 'n') && (tag[j][1] == 'n') ||
                        (tag[j][0] == 'E') && (tag[j][1] == 'X') ||
                        (tag[j][0] == 'P') && (tag[j][1] == 'R') &&
                        (tag[j][2] == 'P') ||
                        (tag[j][0] == 'P') && (tag[j][1] == 'R') &&
                        (tag[j][2] == 'P') ||
                        (tag[j][0] == 'V') && (tag[j][1] == 'B') &&
                        (tag[j][2] == 'G'))) { saw_subj = j; }
                }
            }
        }
    }
    return (count);
}
```

else if (saw_verb == 0) && (saw_to == FALSE) &&
  ( (tag[j][0] == 'v') &&
    (tag[j][1] == 'b') &&
    (tag[j][2] != 'g') &&
    (tag[j][2] != 'n') ) ||
  ( (tag[j][0] == 'm') &&
    (tag[j][1] == 'd') ) )
{
  saw_verb = j;
}

// Else check if token is the word "to"
else if (saw_to == FALSE) &&
  ( (token[j][0] == 't') ||
    (token[j][0] == 'T') ) &&
  (token[j][1] == 'o') &&
  (token[j][2] == '\0') )
{
  saw_to = TRUE;
}

// Continue with the next token
j++;

// Found closing dash
else
{
  saw_end = j;
}

} // end while

// Count and mark the independent clause if found
if ( (saw_subj > 0) &&
    (saw_verb > 0) &&
    (saw_subj < saw_verb) )
{
  count++;

tag[i][0] = 'p';
tag[i][1] = '-';
tag[i][2] = 'd';
tag[i][3] = 's';
tag[i][4] = '\0';
tag[saw_end][0] = 'x';
tag[saw_end][1] = '-';
tag[saw_end][2] = 'd';
tag[saw_end][3] = 's';
tag[saw_end][4] = '\0';
i = j;
} // end if

} // end for
return (count);
} // end split_dash

//
******************************************************************************
// Split off clauses of form "comma-prep-which"
//******************************************************************************

int split_prep_which(void)
{
    int i, j, k, count = 0;
    int saw_subj, saw_verb, saw_end;
    int saw_first_np, saw_first_vp, saw_first_pw;

    saw_first_np = -1;
    saw_first_vp = -1;
    saw_first_pw = -1;

    for(i = 0; i < (Znum - 2); i++)
    {
        // Find the next 'comma-prep-which' occurrence, if any,
        // which is not within a direct quote (for which the
        // first letter of the tag would be 'q')
        if ( (i > 0) &&
            (token[i-1][0] == ',') &&
            (tag[i][0] == 'p') && (tag[i][1] == 'r') &&
            (tag[i][2] == 'e') && (tag[i][3] == 'p') &&
            (tag[i][4] == '\0') &&
            (token[i + 1][0] == 'w') && (token[i + 1][1] == 'h') &&
            (token[i + 1][2] == 'i') && (token[i + 1][3] == 'c') &&
            (token[i + 1][4] == 'h') && (token[i + 1][5] == '\0') )
        {
            // Record the first comma-prep-which
            if ( saw_first_pw < 0 ) saw_first_pw = i;

            // Scan until end of clause, ignoring appositions
            j = i + 2;
            saw_subj = 0;
            saw_verb = 0;
            saw_end = 0;
            while((j < Znum) && (saw_end == 0))
            {
                if( ( (token[j][0] != ',') ||
                    (tag[j][0] == 'c') &&
                    (tag[j][1] == 'a') &&
                    (tag[j][2] == 'p') ) ) &&
                    (tag[j][0] != '.') &&
                    (token[j][0] != ';') &&
                    (token[j][0] != '?') &&
                    (token[j][0] != '!')
                {
                    // Find the first noun phrase (including gerunds)
                }
            }
        }
    }
} // split_prep_which
// that precedes a verb phrase, if any
if ( (saw_subj == 0) &&
     (saw_verb == 0) &&
     ( (tag[j][0] == 'N') && (tag[j][1] == 'N') ) ||
     ( (tag[j][0] == 'n') && (tag[j][1] == 'n') ) ||
     ( (tag[j][0] == 'E') && (tag[j][1] == 'x') ) ||
     ( (tag[j][0] == 'P') && (tag[j][1] == 'P') ) ||
     ( (tag[j][0] == 'p') && (tag[j][1] == 'p') ) ||
     ( (tag[j][0] == 'V') && (tag[j][1] == 'V') ) &&
     (tag[j][2] == 'G') )
)
{
    saw_subj = j;
}

// Find the first verb phrase, if any
if ( (saw_verb == 0) &&
     ( (tag[j][0] == 'V') &&
       (tag[j][1] == 'B') &&
       (tag[j][2] == 'G') ) ||
     ( (tag[j][0] == 'M') &&
       (tag[j][1] == 'D') )
)
{
    saw_verb = j;
    j++;
}
else {
    saw_end = j;
}

// Split if found an independent clause, but not if it
// ends with the end of the sentence and there was not a
// complete clause (NP & VP) preceding the comma-prep-which
if ( (saw_subj > 0) &&
     (saw_verb > 0) &&
     (saw_subj < saw_verb) &&
     ( (tag[j][0] != ',') ) ||
     ( (saw_first_np >= 0) &&
       (saw_first_vp >= 0) )
)
{
    // Mark the end of the clause --
    // If ending comma is unmarked, mark it
    if ( (tag[j][0] == ',')
        {
        tag[j][0] = 'x';
        tag[j][1] = '-';
        tag[j][2] = 'p';
        tag[j][3] = 'w';
        tag[j][4] = '\0';
        }
    // Else insert a marker comma
    else
    {
        //...
shift_right(j, Znum, 1);
token[j][0] = ',';
token[j][1] = '\0';
tag[j][0] = 'x';
tag[j][1] = '_';
tag[j][2] = 'p';
tag[j][3] = 'w';
tag[j][4] = '\0';
}

// Add the preposition and prior noun tag
shift_right(j, Znum, 2);
k = 0;
while ( (token[j][k] = token[i][k] ) != '\0' ) k++;
tag[j][0] = 'p';
tag[j][1] = 'r';
tag[j][2] = 'e';
tag[j][3] = 'p';
tag[j][4] = '\0';

token[j+1][0] = '[';
token[j+1][1] = 'p';
token[j+1][2] = 'r';
token[j+1][3] = 'i';
token[j+1][4] = 'o';
token[j+1][5] = 'r';
token[j+1][6] = 't';
token[j+1][7] = '\0';
tag[j+1][0] = 'n';
tag[j+1][1] = 'n';
tag[j+1][2] = 'p';
tag[j+1][3] = 'w';
tag[j+1][4] = '\0';
count++;

// Mark it for splitting off and eliminate
// the leading 'prep'
token[i][0] = ',';
token[i][1] = '\0';
tag[i][0] = 'p';
tag[i][1] = '_';
tag[i][2] = 'p';
tag[i][3] = 'w';
tag[i][4] = '\0';

// Also eliminate the 'which'
shift_left(i+2, Znum, 1);
}
// end if found clause

// Continue search from end of clause
i = j;

// Else look for a preceding noun phrase
else if ( (saw_first_np < 0) &&
(saw_first_pw < 0) &&
( (tag[i][0] == 'N') && (tag[i][1] == 'N') ) ) ||
int split_which(void)
{
    int i, j, count = 0;
    int saw_subj, saw_verb, saw_end;
    int saw_first_np, saw_first_vp, saw_first_which;

    saw_first_np = -1;
    saw_first_vp = -1;
    saw_first_which = -1;

    for(i = 0; i < (Znum - 1); i++)
Find the next 'comma-which' occurrence, if any, which is not within a direct quote
if (i > 0) &&
  (token[i-1][0] == ',' ) &&
  (token[i][0] == 'q') &&
  (token[i][0] == 'w') && (token[i][1] == 'h') &&
  (token[i][2] == 'i') && (token[i][3] == 'c') &&
  (token[i][4] == 'h') && (token[i][5] == '\0')
{
  // Record the first comma-which
  if (saw_first_which < 0) saw_first_which = i;

  // Scan until end of clause, ignoring appositions
  j = i + 1;
  saw_subj = 0;
  saw_verb = 0;
  saw_end = 0;
  while((j < Znum) && (saw_end == 0))
  { if( (token[j][0] != ',') ||
        (tag[j][0] == 'c') &&
        (tag[j][1] == 'a') &&
        (tag[j][2] == 'p') ) ) &&
        (tag[j][0] != '.') &&
        (token[j][0] != ',') &&
        (token[j][0] != ':') &&
        (token[j][0] != '?') &&
        (token[j][0] != '!') )
  { // Find the first noun phrase (including gerunds)
    // that precedes a verb phrase, if any
    if ( (saw_subj == 0) &&
         (saw_verb == 0) &&
         ( (tag[j][0] == 'N') && (tag[j][1] == 'N') ) ||
         (tag[j][0] == 'n') && (tag[j][1] == 'n') ) ||
         (tag[j][0] == 'E') && (tag[j][1] == 'X') ) ||
         (tag[j][0] == 'P') && (tag[j][1] == 'R') &&
         (tag[j][2] == 'P') ) ||
         (tag[j][0] == 'P') && (tag[j][1] == 'R') &&
         (tag[j][2] == 'P') && (tag[j][3] == 'S') ) ||
         (tag[j][0] == 'V') && (tag[j][1] == 'B') &&
         (tag[j][2] == 'G')
    )
    saw_subj = j;
  }

  // Find the first verb phrase, if any, including
  // indirect quote verbs
  else if((saw_verb == 0) &&
            ( (tag[j][0] == 'V') &&
              (tag[j][1] == 'B') &&
              (tag[j][2] != 'G') ) ||
            (tag[j][0] == 'M') &&
              (tag[j][1] == 'D') ) ||
  { 198}
( (tag[j][0] == 'm') && 
  (tag[j][1] == 'v') )
}

{  
  saw_verb = j;
}  
  j++;
} else {  
  saw_end = j;
}
}

// Split if found an independent clause, but not if it
// ends with the end of the sentence and there was not
// a complete clause (NP & VP) preceding the comma-which
if ( ( saw_subj > 0 ) && 
  ( saw_verb > 0 ) &&
  ( saw_subj < saw_verb ) && 
  ( ( tag[j][0] !='.' ) || 
    ( saw_first_np >= 0 ) &&
    ( saw_first_vp >= 0 ) )
)
{
  // Mark the end of the clause --
  // If ending comma is unmarked, mark it
  if ( tag[j][0] == ',' )
  {
    tag[j][0] = 'x';
    tag[j][1] = '-';
    tag[j][2] = 'w';
    tag[j][3] = 'h';
    tag[j][4] = '\0';
  }

  // Else insert a marker comma
  else
  {
    shift_right(j, Znum, 1);
    token[j][0] = ',',';
    token[j][1] = ',\0';
    tag[j][0] = 'x';
    tag[j][1] = '-';
    tag[j][2] = 'w';
    tag[j][3] = 'h';
    tag[j][4] = '\0';
  }

  // Add the prior noun tag
  shift_right(j, Znum, 1);
  token[j][0] = '[';
  token[j][1] = 'p';
  token[j][2] = 'r';
  token[j][3] = 'i';
  token[j][4] = 'o';
}
Mark it for splitting off and eliminate the leading 'which'
token[i][0] = 'n';
tag[i][0] = 'n';
tag[i][1] = 'w';
tag[i][2] = 'h';
tag[i][3] = 'h';
tag[i][4] = '\0';

// And count the split count++;
}

// Else if found a verb phrase, split it off unless it end at the end of the sentence and a complete clause (NP & VP) was not seen before the comma-which
else if ( ( saw_verb > 0 ) &&
    ( ( tag[j][0] != '.' ) ||
    ( saw_first_np >= 0 ) &&
    ( saw_first_vp >= 0 ) ) )
{
    // Mark the end of the clause --
    // If ending comma is unmarked, mark it
    if ( tag[j][0] == ',' )
    {
        tag[j][0] = 'x';
tag[j][1] = '-';
tag[j][2] = 'w';
tag[j][3] = 'h';
tag[j][4] = '\0';
    }

    // Else insert a marker comma
    else
    {
        shift_right(j, Znum, 1);
token[j][0] = ',',';
token[j][1] = '\0';
tag[j][0] = 'x';
tag[j][1] = '-';
tag[j][2] = 'w';
tag[j][3] = 'h';
tag[j][4] = '\0';
    }

    // Add the prior noun tag and eliminate
// the leading 'which'
token[i][0] = '[';
token[i][1] = 'p';
token[i][2] = 'r';
token[i][3] = 'i';
token[i][4] = 'o';
token[i][5] = 'r';
token[i][6] = ']';
token[i][7] = '\0';
tag[i][0] = 'n';
tag[i][1] = 'n';
tag[i][2] = 'w';
tag[i][3] = 'h';
tag[i][4] = '\0';

// Mark it for splitting off
shift_right(i, Znum, 1);
token[i][0] = ',';
tag[i][0] = 'p';
tag[i][1] = '-';
tag[i][2] = 'w';
tag[i][3] = 'h';
tag[i][4] = '\0';

// And count the split
count++;

} // end if found clause

// Continue search from end of clause
i = j;

} // Else look for a preceding noun phrase
else if ( ( saw_first_np < 0 ) &&
( saw_first_which < 0 ) &&
( ( (tag[i][0] == 'N') && (tag[i][1] == 'N') ) ) ||
( (tag[i][0] == 'n') && (tag[i][1] == 'n') ) ||
( (tag[i][0] == 'E') && (tag[i][1] == 'X') ) ||
( (tag[i][0] == 'P') && (tag[i][1] == 'R') &&
( tag[i][2] == 'P' ) ) ||
( (tag[i][0] == 'P') && (tag[i][1] == 'R') &&
( tag[i][2] == 'P' ) && (tag[i][3] == 'S') ) ||
( (tag[i][0] == 'V') && (tag[i][1] == 'B') &&
( tag[i][2] == 'G' ) ) )
{saw_first_np = i;}

// Else look for a preceding verb phrase
else if( ( saw_first_vp < 0 ) &&
( saw_first_which < 0 ) &&
( (tag[i][0] == 'V') &&
( tag[i][1] == 'B' ) ) )

201
(tag[i][2] != 'G') ||
( (tag[i][0] == 'M') &&
  (tag[i][1] == 'D') ||
( (tag[i][0] == 'm') &&
  (tag[i][1] == 'V') )
)
{
  saw_first_vp = i;
}
  // end if found comma-which
}
  // end for i
return(count);
} // end split_which

//
******************************************************************************
// Split off clauses introduced by "comma-but"
//
******************************************************************************

int split_but(void)
{
  int i, j, count = 0;
  int saw_subj, saw_verb, saw_end;
  int saw_first_np, saw_first_vp, saw_first_but;

  saw_first_np = -1;
  saw_first_vp = -1;
  saw_first_but = -1;

  for(i = 0; i < Znum; i++)
  {
    // Consider only 'but' that follows a comma and is
    // not within a direct quote nor an idiomatic expression
    if( (i > 0) &&
        (token[i-1][0] == ',' ) &&
        (tag[i][0] != 'q') && (tag[i][0] != 'i') &&
        (token[i][0] == 'b') && (token[i][1] == 'u') &&
        (token[i][2] == 't') && (token[i][3] == '\0') )
    {
      // Record the first comma-but
      if ( saw_first_but < 0 ) saw_first_but = i;
      j = i + 1;
      saw_subj = 0;
      saw_verb = 0;
      saw_end = 0;
      while( (j < Znum) && (saw_end == 0) )
      {
      
202
// Scan until find next exclamation point, comma, // semicolon, token within direct quotes, or end // of sentence, ignoring all apposition commas
if( ( tag[j][0] != 'q' ) &&
    ( tag[j][0] != ',' ) &&
    ( ( token[j][0] != ',' ) ||
      ( tag[j][0] == 'c' ) &&
      ( tag[j][1] == 'a' ) &&
      ( tag[j][2] == 'p' ) ) ) &&
  ( token[j][0] != ';' ) &&
  ( token[j][0] != ':' ) &&
  ( token[j][0] != '!' ) &&
  ( token[j][0] != '?' )
)

if( ( saw_subj == 0 ) && ( saw_verb == 0 ) &&
    ( ( tag[j][0] == 'N' ) && ( tag[j][1] == 'N' ) ) ||
    ( tag[j][0] == 'n' ) && ( tag[j][1] == 'n' ) ) ||
  ( tag[j][0] == 'E' ) && ( tag[j][1] == 'X' ) ) ||
  ( tag[j][0] == 'P' ) && ( tag[j][1] == 'R' ) &&
  ( tag[j][2] == 'p' ) ) ||
  ( tag[j][0] == 'P' ) && ( tag[j][1] == 'R' ) &&
  ( tag[j][2] == 'P' ) && ( tag[j][3] == 'S' ) ) ||
  ( tag[j][0] == 'V' ) && ( tag[j][1] == 'B' ) &&
  ( tag[j][2] == 'G' ) )
)

saw_subj = j;

// Find the first verb phrase, if any
else if( ( saw_verb == 0 ) &&
    ( tag[j][0] == 'V' ) &&
    ( tag[j][1] == 'B' ) &&
    ( tag[j][2] != 'G' ) ) ||
  ( tag[j][0] == 'M' ) &&
  ( tag[j][1] == 'D' ) )

saw_verb = j;

j++;
}
else {
  saw_end = j;
}

// Split off an independent clause, if found, but not if it // end with the end of the sentence and there was not a // complete clause (NP & VP) preceding the comma—but
if ( ( saw_subj > 0 ) &&
    ( saw_verb > 0 ) &&
    ( saw_subj < saw_verb ) &&
    ( tag[j][0] != '.' ) ||
    ( saw_first_np >= 0 ) &&
    ( saw_first_vp >= 0 ) )
)
if((tag[i][0] != 'C') || (tag[i][1] != 'C'))
{
    shift_right(i, Znum, 1);
}

token[i][0] = '.';
token[i][1] = '\0';
tag[i][0] = 'p';
tag[i][1] = '-';
tag[i][2] = 'b';
tag[i][3] = 't';
tag[i][4] = '\0';
count++;

} // Else split off a verb phrase clause, if found
else if ((saw_dist > 0) &&
    ((tag[j][0] != '.') ||
    ((saw_first_np >= 0) &&
     (saw_first_vp >= 0)))
    )
{
    if((tag[i][0] != 'C') || (tag[i][1] != 'C'))
    {
        shift_right(i, Znum, 1);
    }
    token[i][0] = '.';
token[i][1] = '\0';
tag[i][0] = 'p';
tag[i][1] = '-';
tag[i][2] = 'b';
tag[i][3] = 't';
tag[i][4] = '\0';
    j = i + 1;
    shift_right(j, Znum, 1);

token[j][0] = '[';
token[j][1] = 'S';
token[j][2] = 'u';
token[j][3] = 'b';
token[j][4] = 'j';
token[j][5] = ']';
token[j][6] = '\0';
tag[j][0] = 'p';
tag[j][1] = '-';
tag[j][2] = 'b';
tag[j][3] = 't';
tag[j][4] = '\0';
count++;
}
} // end if saw
// Continue search from end of clause
i = j;
}

// Else look for a preceding noun phrase
else if ( ( saw_first_np < 0 ) &&
    ( saw_first_but < 0 ) &&
    ( (tag[i][0] == 'N') && (tag[i][1] == 'N') ) ||
    ( (tag[i][0] == 'n') && (tag[i][1] == 'n') ) ||
    ( (tag[i][0] == 'E') && (tag[i][1] == 'X') ) ||
    ( (tag[i][0] == 'P') && (tag[i][1] == 'R') &&
      (tag[i][2] == 'P') ) ||
    ( (tag[i][0] == 'P') && (tag[i][1] == 'R') &&
      (tag[i][2] == 'P') ) ||
    ( (tag[i][0] == 'P') && (tag[i][1] == 'R') &&
      (tag[i][2] == 'P') && (tag[i][3] == 'S') ) ||
    ( (tag[i][0] == 'V') && (tag[i][1] == 'B') &&
      (tag[i][2] == 'G') )
  ) )
{
    saw_first_np = i;
}

// Else look for a preceding verb phrase
else if( ( saw_first_vp < 0 ) &&
    ( saw_first_but < 0 ) &&
    ( (tag[i][0] == 'V') &&
      (tag[i][1] == 'B') &&
      (tag[i][2] != 'G') ) ||
    ( (tag[i][0] == 'M') &&
      (tag[i][1] == 'D') ) ||
    ( (tag[i][0] == 'm') &&
      (tag[i][1] == 'V') )
  )
{
    saw_first_vp = i;
}

} // end if found comma-but

} // end for i

return (count);

} // end split_but

// Split off S and VP coordinations but
// not within subordinate clauses

int split_svp_coord(void)
{
  int i, j, k, m, mm, start, count = 0;

  // Continue search from end of clause
  i = j;

  // Else look for a preceding noun phrase
  else if ( ( saw_first_np < 0 ) &&
    ( saw_first_but < 0 ) &&
    ( (tag[i][0] == 'N') && (tag[i][1] == 'N') ) ||
    ( (tag[i][0] == 'n') && (tag[i][1] == 'n') ) ||
    ( (tag[i][0] == 'E') && (tag[i][1] == 'X') ) ||
    ( (tag[i][0] == 'P') && (tag[i][1] == 'R') &&
      (tag[i][2] == 'P') ) ||
    ( (tag[i][0] == 'P') && (tag[i][1] == 'R') &&
      (tag[i][2] == 'P') ) ||
    ( (tag[i][0] == 'P') && (tag[i][1] == 'R') &&
      (tag[i][2] == 'P') && (tag[i][3] == 'S') ) ||
    ( (tag[i][0] == 'V') && (tag[i][1] == 'B') &&
      (tag[i][2] == 'G') )
  ))
  {
    saw_first_np = i;
  }

  // Else look for a preceding verb phrase
  else if( ( saw_first_vp < 0 ) &&
    ( saw_first_but < 0 ) &&
    ( (tag[i][0] == 'V') &&
      (tag[i][1] == 'B') &&
      (tag[i][2] != 'G') ) ||
    ( (tag[i][0] == 'M') &&
      (tag[i][1] == 'D') ) ||
    ( (tag[i][0] == 'm') &&
      (tag[i][1] == 'V') )
  )
  {
    saw_first_vp = i;
  }

} // end if found comma-but

} // end for i

return (count);

} // end split_svp_coord

// Split off S and VP coordinations but
// not within subordinate clauses

int split_svp_coord(void)
{
  int i, j, k, m, mm, start, count = 0;

  // Continue search from end of clause
  i = j;

  // Else look for a preceding noun phrase
  else if ( ( saw_first_np < 0 ) &&
    ( saw_first_but < 0 ) &&
    ( (tag[i][0] == 'N') && (tag[i][1] == 'N') ) ||
    ( (tag[i][0] == 'n') && (tag[i][1] == 'n') ) ||
    ( (tag[i][0] == 'E') && (tag[i][1] == 'X') ) ||
    ( (tag[i][0] == 'P') && (tag[i][1] == 'R') &&
      (tag[i][2] == 'P') ) ||
    ( (tag[i][0] == 'P') && (tag[i][1] == 'R') &&
      (tag[i][2] == 'P') ) ||
    ( (tag[i][0] == 'P') && (tag[i][1] == 'R') &&
      (tag[i][2] == 'P') && (tag[i][3] == 'S') ) ||
    ( (tag[i][0] == 'V') && (tag[i][1] == 'B') &&
      (tag[i][2] == 'G') )
  )
  {
    saw_first_np = i;
  }

  // Else look for a preceding verb phrase
  else if( ( saw_first_vp < 0 ) &&
    ( saw_first_but < 0 ) &&
    ( (tag[i][0] == 'V') &&
      (tag[i][1] == 'B') &&
      (tag[i][2] != 'G') ) ||
    ( (tag[i][0] == 'M') &&
      (tag[i][1] == 'D') ) ||
    ( (tag[i][0] == 'm') &&
      (tag[i][1] == 'V') )
  )
  {
    saw_first_vp = i;
  }

} // end if found comma-but

} // end for i

return (count);

} // end split_svp_coord
int verbs[5], priors[5], num_verbs, num_priors;
int saw_subj, sawVerb, saw_end, saw_modal, saw_that;
inr first_subj, last_verb, prior_verb, to_infinite;
inr saw_modal2, first_to;

// First look for an independent clause
saw_subj = -1;
sawVerb = -1;
saw end = -1;
saw that = -1;
saw_modal = -1;
lase verb = -1;
prior_verb = -1;
to_infinite = -1;
first_to = -1;

j = 0;

while( (j < Znum) & & (saw_end < 0) && (saw that < 0) )
{
    // If the main clause has the words "that", "neither" or
    // "nor", do not consider splitting the sentence at all
    if( ( (token[j][0] == 't') & & (token[j][1] == 'h') & &
        (token[j][2] == 'a') & & (token[j][3] == 't') ) & &
        (token[j][4] == '\0' ) ) ||
        ( (token[j][0] == 'n') & & (token[j][1] == 'o') & &
        (token[j][2] == 'r') & & (token[j][3] == '\0' ) ) ||
        ( (token[j][0] == 'n') & & (token[j][1] == 'e') & &
        (token[j][2] == 't') & & (token[j][3] == 't') ) & &
        (token[j][4] == 'h') & & (token[j][5] == 'e') & &
        (token[j][6] == 'r') & & (token[j][7] == '\0' ) ) )
    {
        saw that = j;
    }

    // Scan until find next coordinating conjunction, colon
    // exclamation point, coordination comma, semicolon,
    // split marker, or end of sentence
    if( (j != 0) & &
        ( ( tag[j][0] == '.' ) ) ||
        ( token[j][0] == ';' ) ) ||
        ( token[j][0] == ':' ) ) ||
        ( token[j][0] == '!' ) ) ||
        ( ( tag[j][0] == 'c' ) & & ( tag[j][1] == 'c' ) & &
        ( tag[j][2] == 'r' ) & & ( tag[j][3] == 'd' ) ) ) ||
        ( ( tag[j][0] == 'c' ) & & ( tag[j][1] == 'c' ) ) ) ||
        ( ( tag[j][0] == 'p' ) & & ( tag[j][1] == '-' ) ) ||
        ( ( tag[j][0] == 'x' ) & & ( tag[j][1] == '-' ) ) )
    {
        saw_end = j;
    }
    else
    {
        // Find the first noun phrase preceding a verb phrase,
        // if any
    }
}

206
if ( (saw_subj < 0) && (saw_verb < 0) &&
    (((tag[j][0] == 'N') && (tag[j][1] == 'N')) ||
     ((tag[j][0] == 'n') && (tag[j][1] == 'n')) ||
     ((tag[j][0] == 'E') && (tag[j][1] == 'E')) ||
     ((tag[j][0] == 'P') && (tag[j][1] == 'P') &&
      (tag[j][2] == 'P')) ||
     ((tag[j][0] == 'P') && (tag[j][1] == 'R') &&
      (tag[j][2] == 'P') && (tag[j][3] == 'S')) ||
     ((tag[j][0] == 'V') && (tag[j][1] == 'B') &&
      (tag[j][2] == 'G')) ||
     (((tag[j][0] == 'C') && (tag[j][1] == 'D')))
  ) )

saw_subj = j;
first_subj = j;

// Find the first and last verb phrases, if any
else if ( (tag[j][0] == 'V') &&
          (tag[j][1] == 'B') &&
          (tag[j][2] == 'G') )
{
    // Keep track of previous verb, if any
    if ( saw_verb >= 0 )
    {
        prior_verb = last_verb;
    }

    // Past participle is not a valid first verb seen
    else if ( (saw_verb < 0) && (tag[j][2] != 'N') )
    {
        saw_verb = j;
    }

    // The token is the most recently seen verb
    last_verb = j;

    // Update the list of verbs in sequence
    num_priors = 0;
k = 0;
    while ( (j - k) > 0 ) && ( num_priors < 5 ) &&
        (((tag[j-k][0] == 'V') && (tag[j-k][1] == 'B') &&
          (tag[j-k][2] == 'G')) ||
         ((tag[j-k][0] == 'R') && (tag[j-k][1] == 'B')) ||
         ((tag[j-k][0] == 'M') && (tag[j-k][1] == 'D'))) )
    {
        if ( ((tag[j-k][0] == 'V') && (tag[j-k][1] == 'B') &&
              (tag[j-k][2] == 'G')) ||
             ((tag[j-k][0] == 'M') && (tag[j-k][1] == 'D')) )
        {
            priors[num_priors] = j - k;
            num_priors++;
        }
    }

    // NOTE: k can be >= 5 if there are intervening
    // adverbs
    k++;
}
/**
 * Keep track of modal verb forms, if any
 * else if ((tag[j][0] == 'M') && (tag[j][1] == 'D'))
 * { saw_modal = j; }
 *
 * // Keep track of infinitive verb forms, if any
 * else if ((saw_verb >= 0) && (tag[j][0] == 'T') && (tag[j][1] == 'O'))
 * { to_infinitive = j;
 *   if (first_to < 0) first_to = j;
 * }
 *
 * // Watch for infinitive preceding any other verb form
 * else if ((first_to < 0) && (tag[j][0] == 'T') && (tag[j][1] == 'O'))
 * { first_to = j;
 * }
 *
 * // Go on to examine the next word
 * j++;
 * }
 *
 * } // end if
 * }
 * // end while
 *
 * // Do not process the sentence if did not find a complete clause
 * // or if found "that" in main clause
 * // or if saw "to" before first verb
 * if((saw_that < 0) && ((first_to < 0) || (saw_verb < first_to)) &&
 *   ((saw_subj >= 0) && (saw_verb > 0) && (saw_subj < saw_verb)))
 * { start = saw_end;
 *   if (start <= 0) start = Znum;
 * } else
 * {
 *   start = Znum;
 * }
 *
 * // Start after first complete clause and search for coordinations
 * i = start;
 * while (i < (Znum - 1))
 * {
 *   num_verbs = 0;
 *
 *   // Skip over leading coordination commas, CCs, split codes,
 *   // all tokens within direct quotes, and end of sentence
 *   while (i < Znum) &&
 *     ( (tag[i][0] == 'q') || (tag[i][0] == '.') ||
 *   *
 * 208
if ( (i > 0) && (i < (Znum - 1)) )
{
  j = i;
  saw_subj = -1;
  saw_verb = -1;
  saw_end = -1;
  saw_that = -1;
  saw_modal2 = -1;
  first_to = -1;
  num_verbs = 0;
  while( (j < Znum) && (saw_end < 0) && (saw_that < 0) )
  {
    // If the clause has the word "that", do not consider
    // splitting this clause
    if( (token[j][0] == 't') && (token[j][1] == 'h') &&
        (token[j][2] == 'a') && (token[j][3] == 't') &&
        (token[j][4] == '\0') ) ||
      (token[j][0] == 'n') && (token[j][1] == 'o') &&
      (token[j][2] == 'r') && (token[j][3] == '\0') ) ||
    {
      saw_that = j;
      j++;
    }

    // Scan until find next coordinating conjunction, colon,
    // exclamation point, coordination comma, semicolon,
    // end of sentence or split marker
    if( (j > i) &&
        (tag[j][0] == '.') ) ||
      (token[j][0] == '!!') ) ||
      (token[j][0] == ';') ) ||
      (token[j][0] == ':') ) ||
      (tag[j][0] == 'c') && (tag[j][1] == 'c') &&
      (tag[j][2] == 'r') && (tag[j][3] == 'd') ) ||
    {
      (tag[j][0] == 'C') && (tag[j][1] == 'C') ) ||
      (tag[j][0] == 'P') && (tag[j][1] == '\-') ) ||
      (tag[j][0] == 'x') && (tag[j][1] == '\-') )
  }
{  
saw_end = j;
}

else
{
    // Find the first noun phrase preceding a verb phrase, if any
    if ( (saw_subj < 0) && (saw_verb < 0) &&
         (((tag[j][0] == 'N') && (tag[j][1] == 'N')) ||
          ((tag[j][0] == 'n') && (tag[j][1] == 'n')) ||
          ((tag[j][0] == 'E') && (tag[j][1] == 'X')) ||
          ((tag[j][0] == 'P') && (tag[j][1] == 'R') &&
           (tag[j][2] == 'P')) ||
          ((tag[j][0] == 'P') && (tag[j][1] == 'R') &&
           (tag[j][2] == 'P') && (tag[j][3] == 'S')) ||
          ((tag[j][0] == 'V') && (tag[j][1] == 'B') &&
           (tag[j][2] == 'G')) ||
          ((tag[j][0] == 'C') && (tag[j][1] == 'D')) )
    
    {  
        saw_subj = j;
    }

    // Find the first verb phrase, if any
    else if( (tag[j][0] == 'V') &&
             (tag[j][1] == 'B') ) ||
            (tag[j][0] == 'M') &&
            (tag[j][1] == 'D') )
    
    {  
        // Present participle and past participle
        // are not valid forms for first verb
        if ( (saw_verb < 0) &&
             (tag[j][2] != 'G') &&
             (tag[j][2] != 'N') )
        {
            saw_verb = j;
        }

        // Advance the verb pointer for verbs in sequence
        // if current verb immediately follows prior verb
        // (present and past participles are valid here)
        if ( j == (saw_verb + 1) )
        {
            saw_verb++;
        }
    }

    // Keep track of modal verb forms, if any
    if( (tag[j][0] == 'M') && (tag[j][1] == 'D') )
    {  
        saw_modal2 = j;
    }

    // Update the list of verbs in sequence
    num_verbs = 0;
    k = 0;
}
while((( j - k ) >= 0 ) && ( num_verbs < 5 ) &&
((tag[j-k][0] == 'V') && (tag[j-k][1] == 
'B') &&
(tag[j-k][2] != 'G')) ||
((tag[j-k][0] == 'R') && (tag[j-k][1] == 
'B')) ||
((tag[j-k][0] == 'M') && (tag[j-k][1] == 
'D')) ))
{
    if( ( (tag[j-k][0] == 'V') && (tag[j-k][1] == 
'B') ) &&
        (tag[j-k][2] != 'G') ) ||
    ( (tag[j-k][0] == 'M') && (tag[j-k][1] == 
'D') ) )
    {
        verbs[num_verbs] = j - k;
        num_verbs++;
    }
// NOTE: k can be >= 5 if there are intervening
adverbs
    k++;
}

// Keep track of infinitive verb forms, if any
else if ( (saw_verb >= 0) &&
        (tag[j][0] == 'T') && (tag[j][1] == 'O') )
{
    to_infinitive = j;
    if( first_to < 0 ) first_to = j;
}

// Watch for infinitive preceding any other verb form
else if ( ( first_to < 0 ) &&
        (tag[j][0] == 'T') && (tag[j][1] == 'O') )
{
    first_to = j;
}

// Go on to examine the next token
j++;
} // end if
}
// end while

if ( saw_end <= 0 ) saw_end = Znum - 1;
} // end if i

// If saw both subject and verb in that order, split off a
// sentence unless the clause has already been marked for
// splitting off or unless saw "that", "neither", or "nor"
// or unless saw "to" before any verb form
if( ( i > 0 ) &&
( (tag[i-1][0] != 'p') || (tag[i-1][1] != '-') ) &&
( (tag[i-1][0] != 'x') || (tag[i-1][1] != '-') ) &&
( i < (Znum - 1) ) &&
( saw_that < 0 ) &&
( saw_subj >= 0 ) &&
( saw_verb > 0 ) &&
( saw_subj < saw_verb ) &&
( first_to < 0 ) || ( saw_verb < first_to ) )
}

if( (tag[i-1][0] == 'C') && (tag[i-1][1] == 'C') )
{
    i--;  
}
else
{
    shift_right(i, Znum, l);

    saw_subj++;
    saw_verb++;
    saw_end++;
    for( mm = 0; mm < num_verbs; mm++ ) verbs[mm]++;
}

token[i][0] = '.';
token[i][1] = '\0';

tag[i][0] = 'p';
tag[i][1] = '-';
tag[i][2] = 's';
tag[i][3] = 'c';
tag[i][4] = '\0';

count++;

// Else if saw a verb phrase, split off a VP-coordination
// unless the clause has already been marked for splitting
// off or unless saw "to" before any other verb form
else if( ( i > 0 ) &&

    ( (tag[i-1][0] != 'p') || (tag[i-1][1] != '-') ) &&
    ( (tag[i-1][0] != 'x') || (tag[i-1][1] != '-') ) &&
    ( i < (Znum - 1) ) &&
    ( saw_that < 0 ) &&
    ( saw_verb > 0 ) &&
    ( first_to < 0 ) || ( saw_verb < first_to ) )
)

if( (tag[i-1][0] == 'C') || (tag[i-1][1] == 'C') )
{
    i--;  
}
else
{
    shift_right(i, Znum, l);

    saw_verb++;
    saw_end++;
for ( mm = 0; mm < num_verbs; mm++) verbs[mm]++;
}

token[i][0] = '.';
token[i][1] = '\0';
tag[i][0] = 'p';
tag[i][1] = '-';
tag[i][2] = 'v';
tag[i][3] = 'p';
tag[i][4] = '\0';
j = i + 1;
shift_right(j, Znum, 1);
saw_verb++;saw_end++;for ( mm = 0; mm < num_verbs; mm++) verbs[mm]++;

token[j][0] = '[';
token[j][1] = 'S';
token[j][2] = 'u';
token[j][3] = 'b';
token[j][4] = 'j';
token[j][5] = ']';
token[j][6] = '\0';
tag[j][0] = 'p';
tag[j][1] = '-';
tag[j][2] = 'v';
tag[j][3] = 'p';
tag[j][4] = '\0';
vp_splits++;

// If infinitive form was used in main clause, repeat // main clause's main verb and use infinitive form // in split off clause
if( (prior_verb > 0) &&
    (to_infinitive > 0) &&
    (to_infinitive > prior_verb) &&
    (to_infinitive < last_verb) )
{
    shift_right(j+1, Znum, 2);
saw_verb += 2;saw_end += 2;
for ( mm = 0; mm < num_verbs; mm++) verbs[mm] += 2;

    k = 0;
    while( (token[j+1][k] = token[prior_verb][k]) != '\0' )
        k++;

    k = 0;
    while( (tag[j+1][k] = tag[prior_verb][k]) != '\0' )
        k++;
k = 0;
while( (tag[j+1][k] >= 'A') && (tag[j+1][k] <= 'Z') &&
       (tag[j+1][k] != 'O'))
{
    tag[j+1][k] = 'a' + (tag[j+1][k] - 'A');
    k++;
}

for( (token[j+2][0] = 't';
       token[j+2][1] = 'o';
       token[j+2][2] = '\0';

inf_count++;
}

// Else if modal verb form was used in main clause
// and first verb in split off clause is not modal and
// is in base form, then repeat modal verb form in
// split off clause
else if( (saw_modal > 0) &&
        (saw_modal > first_subj) &&
        (saw_modal < last_verb) &&
        (tag[saw_verb][2] == '\0') &&
        (saw_modal2 < 0)
)
{
    shift_right(j+1, Znum, 1);
    saw_verb++;
    saw_end++;
    for( (mm = 0; mm < num_verb; mm++) verbs[mm]++;

k = 0;
while( (token[j+1][k] = token[saw_modal][k]) != '\0')
    k++;

    tag[j+1][0] = 'm';
    tag[j+1][1] = 'd';
    tag[j+1][2] = '\0';

    mod_count++;
}

// Else if complex verb construction, carry over
// auxiliary verbs, if any
else if( (num_priors > 1) && (num_verb < num_priors) )
{
    shift_right( j+1, Znum, (num_priors - num_verb) );
}

saw_verb += (num_priors - num_verb);

saw_end += (num_priors - num_verb);

for( (mm = 0; mm < num_verb; mm++)
    verbs[mm] += (num_priors - num_verb);
for(m = 1; m <= (num_priors - num_verbs); m++)
{
    k = 0;
    while( (token[j+m][k] =
        token[priors[num_priors - m]][k] ) != '\0')
        k++;
    k = 0;
    while( (tag[j+m][k] =
        tag[priors[num_priors - m]][k]) != '\0')
        k++;
    k = 0;
    while((tag[j+m][k] >= 'A') && (tag[j+m][k] <= 'Z') &&
        (tag[j+m][k] != '\0'))
        { tag[j+m][k] = 'a' + ( tag[j+m][k] - 'A' );
        k++;
    }
} // end for

    aux_count++;

} // end if saw special verb constructions

} // end if saw

// Do not process the sentence further if found "that"
// in the current clause
if( saw_that >=0 )
{
    i = Znum;
}

//Else set up to examine the next clause, if any
else
{
    // This was the last clause
    if ( ( saw_end <= i ) || ( i < 0 ) || ( i >= (Znum - 1) ) )
    {
        i = Znum;
    }

    // Else there is another clause
    else
    {
        i = saw_end;
    }

    // Reset auxiliary verb array to prepare for next clause
    if ( num_verbs > 0 )
    {
        num_priors = num_verbs;
        for(k = 0; k < num_priors; k++) priors[k] = verbs[k];
    }
int split_with(void)
{
    int i, j, count = 0;
    int saw gerund, saw end;
    int saw_first_np, saw_first_vp, saw_first_with;

    // Initialize
    saw_first_np = -1;
    saw_first_vp = -1;
    saw_first_with = -1;

    for(i = 0; i < (Znum - 1); i++)
    {
        // Find the next "comma-with", if any, which is not
        // within a direct quote
        if ( (i > 0) &&
             (token[i-1][0] == ',') &&
             (tag[i][0] != 'q') &&
             (token[i][0] == 'w') &&
             (token[i][1] == 'i') &&
             (token[i][2] == 't') &&
             (token[i][3] == 'h') &&
             (token[i][4] == '\0'))
        {
            // Record the first comma-with
            if ( saw_first_with < 0 ) saw_first_with = i;

            // Search for a gerund, if any, before a terminator,
            // ignoring intervening appositions
            saw gerund = 0;
            saw_end = 0;
            j = i + 1;
            while ( (j < Znum) && (saw_end == 0) )
            {
                if ( (token[j][0] != ',') ||
                     (tag[j][0] != 'c') &&
                     (tag[j][1] == 'a') &&
                     (tag[j][2] == 'p' ) ) ) ) &&
                    (tag[j][0] != '.') ) &&
                    (token[j][0] != ';') &&
                    (token[j][0] != ':') &&
                    (token[j][0] != '?') )
    }
}
(token[j][0] != '!')
{
    if ((saw_gerund == 0) &&
        (tag[j][0] == 'V') && (tag[j][1] == 'B') &&
        (tag[j][2] == 'G') && (tag[j][3] == '\0'))
    {
        saw_gerund = j;
    }
    j++;
}
else
{
    saw_end = j;
}
} // end while

// Found the gerund, so mark the clause for splitting,
// but not if it end with the end of the sentence and
// there was not a complete clause (NP + VP) preceding
// the comma-with
if ((saw_gerund > 0) &&
     (saw_end > 0) &&
     (saw_gerund < saw_end) &&
     ((tag[j][0] != '.') ||
      (saw_first_np >= 0) &&
      (saw_first_vp >= 0)))
{
    token[i][0] = '.';
    token[i][1] = '\0';
    tag[i][0] = 'p';
    tag[i][1] = '-';
    tag[i][2] = 'w';
    tag[i][3] = 'i';
    tag[i][4] = '\0';
    count++;
}

// Convert the gerund verb form to the past tense
fix_gerund(saw_gerund);

// If ending comma is unmarked, mark it
if (tag[j][0] == ',')
{
    tag[j][0] = 'x';
    tag[j][1] = '-';
    tag[j][2] = 'w';
    tag[j][3] = 'i';
    tag[j][4] = '\0';
}

// Else insert a marker comma
else
{
    shift_right(j, Znum, 1);
    token[j][0] = ',';
    token[j][1] = '\0';
    tag[j][0] = 'x';

217
tag[j][1] = '\-';
tag[j][2] = 'w';
tag[j][3] = 'l';
tag[j][4] = '\"';
}
} // end if saw gerund

// Continue search from end of clause
i = j;
}

// Else look for a preceding noun phrase
else if ( (saw_first_np < 0) &&
( saw_first_with < 0 ) &&
( (tag[i][0] == 'N') && (tag[i][1] == 'N') ) ||
( (tag[i][0] == 'n') && (tag[i][1] == 'n') ) ||
( (tag[i][0] == 'E') && (tag[i][1] == 'X') ) ||
( (tag[i][0] == 'B') && (tag[i][1] == 'R') &&
( tag[i][2] == 'P' ) ) ||
( (tag[i][0] == 'P') && (tag[i][1] == 'R') &&
( tag[i][2] == 'P' ) && (tag[i][3] == 'S') ) ||
( (tag[i][0] == 'V') && (tag[i][1] == 'B') &&
( tag[i][2] == 'G' ) )
}
{saw_first_np = i;
}

// Else look for a preceding verb phrase
else if( (saw_first_vp < 0) &&
( saw_first_with < 0 ) &&
( (tag[i][0] == 'V') &&
  (tag[i][1] == 'B') &&
  (tag[i][2] != 'G') ) ||
( (tag[i][0] == 'M') &&
  (tag[i][1] == 'D') ) ||
( (tag[i][0] == 'm') &&
  (tag[i][1] == 'V') )
)
{saw_first_vp = i;
}
} // end if found comma-with

} // end for

return(count);
} // end split_with

218
LIST OF REFERENCES


