Null Element Restoration

Ryan Gabbard

University of Pennsylvania, gabbard@seas.upenn.edu

Follow this and additional works at: http://repository.upenn.edu/edissertations

Part of the Artificial Intelligence and Robotics Commons, and the Linguistics Commons

Recommended Citation

Null Element Restoration

Abstract
Understanding the syntactic structure of a sentence is a necessary preliminary to understanding its semantics and therefore for many practical applications. The field of natural language processing has achieved a high degree of accuracy in parsing, at least in English. However, the syntactic structures produced by the most commonly used parsers are less detailed than those structures found in the treebanks the parsers were trained on. In particular, these parsers typically lack the null elements used to indicate wh-movement, control, and other phenomena.

This thesis presents a system for inserting these null elements into parse trees in English. It then examines the problem in Arabic, which motivates a second, joint-inference system which has improved performance on English as well. Finally, it examines the application of information derived from the Google Web 1T corpus as a way of reducing certain data sparsity issues related to wh-movement.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Computer and Information Science

First Advisor
Mitch Marcus

Subject Categories
Artificial Intelligence and Robotics | Linguistics

This dissertation is available at ScholarlyCommons: http://repository.upenn.edu/edissertations/264
NULL ELEMENT RESTORATION

Ryan Gabbard

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2010

Mitch Marcus
Professor
Supervisor of Dissertation

Jianbo Shi
Associate Professor
Graduate Group Chairman

Dissertation Committee:
Daniel Bikel (Research Scientist, Google Research)
Aravind Joshi (Professor, Computer and Information Science)
Mark Liberman (Professor, Linguistics)
Ben Taskar (Assistant Professor, Computer and Information Science)
For Granny
Acknowledgements

Mitch Marcus has been an outstanding advisor. His ability to place work in a broad intellectual and historical context was very beneficial, and I always came away from our meetings infected anew by his enthusiasm for the field. I have valued both his academic advice and his friendship.

I would also like to thank the members of my committee, Daniel Bikel, Aravind Joshi, Mark Liberman, and Ben Taskar, as well as my frequent collaborator Seth Kulick. I also benefitted very much from a summer at BBN Technologies, and I would like to thank my past and current colleagues there. Special thanks are also due to Lynn Guindon and Greg Stump of the University of Kentucky, who were responsible for first sparking my interest in human language.

Penn is a wonderful place to be for a student of computational linguistics. I have benefited from conversations with many of my fellow students, most notably Erwin Chan, Mark Dredze, Neville Ryant, and Qiuye Zhao. Emily Pitler deserves special mention for her help in running experiments on the POS-tagged version of the Web 1T corpus. I would also like to thank the students in my classes and Cheryl Hickey, who has been a tremendous help on many occasions.

My thanks are also due to several non-academic sources. I would like to thank Rob Mygatt, Sage Shepperd, and all the Penn Gamers for many happy hours playing board games. For a different but no less valuable formation than my PhD program, I owe very deep thanks to the parishes of St. Mark’s, Locust Street (especially the Rev. Richard Alton, Liza Anderson, the Rev. Sean Mullen, and Grace Treiman) and

Finally, I could not have reached this point without the support of my parents, my brother and sister, and my grandparents. My deepest and continual thanks for support and motivation are due to my wife Dianna, who was certainly my most significant discovery while at Penn.
ABSTRACT
NULL ELEMENT RESTORATION
Ryan Gabbard
Supervisor: Mitch Marcus
Professor

Understanding the syntactic structure of a sentence is a necessary preliminary to understanding its semantics and therefore for many practical applications. The field of natural language processing has achieved a high degree of accuracy in parsing, at least in English. However, the syntactic structures produced by the most commonly used parsers are less detailed than those structures found in the treebanks the parsers were trained on. In particular, these parsers typically lack the null elements used to indicate wh-movement, control, and other phenomena.

This thesis presents a system for inserting these null elements into parse trees in English. It then examines the problem in Arabic, which motivates a second, joint-inference system which has improved performance on English as well. Finally, it examines the application of information derived from the Google Web 1T corpus as a way of reducing certain data sparsity issues related to wh-movement.
# Contents

Acknowledgements iii

1 Introduction 1
   1.1 Null Elements in the Penn Treebank 4
      1.1.1 Units 4
      1.1.2 Null Complementizers 5
      1.1.3 PROs 5
      1.1.4 Wh-movement 6
      1.1.5 Topicalization 8
      1.1.6 Ellipsed Predicates 9
      1.1.7 Template gapping anti-placeholder 9
      1.1.8 Pseudo-attachments 9
   1.2 Elements Under Consideration 11

2 Related Work 12
   2.1 Pattern-Matching 12
      2.1.1 The Training Phase 13
      2.1.2 The Application Phase 14
      2.1.3 The Preprocessor 14
      2.1.4 Evaluation 15
      2.1.5 “Looser” Pattern-Matching 16
2.1.6 Regular Expression Patterns .................................... 16
2.2 Parsing ................................................................. 16
  2.2.1 Partial Integration .................................................. 18
  2.2.2 Full Integration .................................................... 19
  2.2.3 Evaluation .......................................................... 20
  2.2.4 Better Unlexicalized Results .................................... 21
2.3 Rules ................................................................. 22
  2.3.1 Technique ......................................................... 22
  2.3.2 Evaluation ......................................................... 24
2.4 Machine Learning .................................................... 24
  2.4.1 Pipeline ............................................................. 25
  2.4.2 Learning ............................................................ 26
  2.4.3 English evaluation ............................................... 26
  2.4.4 German Evaluation .............................................. 29
2.5 Conclusion ............................................................ 30
  2.5.1 Division of the Problem ......................................... 30
  2.5.2 Annotation Inconsistency ....................................... 30
  2.5.3 Efficacy ............................................................ 31
  2.5.4 Appendix: Lexicalized Tree-Adjoining Grammar ........... 32

3 A Null Element System for English ................................ 33
  3.1 Runtime ............................................................. 33
  3.2 Feature Set .......................................................... 35
    3.2.1 Function tags .................................................. 37
  3.3 Training ............................................................. 38
  3.4 Results ............................................................. 38
    3.4.1 Gold-standard data ............................................ 38
    3.4.2 Automatically Parsed Data .................................. 39
    3.4.3 Comparison to Campbell ..................................... 41
5.7.1 Difficulties in Comparing Results 75
5.7.2 Results 77

6 System Analysis 79
6.1 Error Analysis 79
   6.1.1 Nominal null *wh*-words 79
   6.1.2 Adverbial null *wh*-words 80
   6.1.3 Nominal *wh*-traces 82
   6.1.4 Adverbial Trace (ADVP *(T*)) 87
6.2 Feature Ablation 89

7 Parsing With Google 99
7.1 The Google Web 1T Corpus 99
7.2 Null *wh*-word types 100
   7.2.1 Approach 101
   7.2.2 Results 103
7.3 Infinitival Relatives 104
   7.3.1 Applying Google to the Problem 109
   7.3.2 Results 114
7.4 Conclusions 115

8 Conclusions and Future Work 117
8.1 Contributions 117
8.2 Future Work 118

A Evaluation 119
A.1 Johnson’s metric 119
A.2 Campbell’s metric 120
A.3 Typed–dependency metrics 122
   A.3.1 Towards an Ideal Evaluation 124
List of Tables

1.1 Frequencies of the most common null elements in English . . . . . . . 4

2.1 The performance of Johnson’s system, by his metric . . . . . . . . . . 15
2.2 Dienes and Dubey’s null element performance. . . . . . . . . . . . . 21
2.3 Null element performance of Levy and Manning’s system . . . . . . . 28
2.4 Null element performance of Levy and Manning’s system (typed-
dependency) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28

3.1 List of function tags in the Penn Treebank and their distribution. . . 37
3.2 System performance on gold standard trees. . . . . . . . . . . . . . . 38
3.3 System performance on parser output. . . . . . . . . . . . . . . . . . 39
3.4 System performance compared to Dienes and Dubey . . . . . . . . . . 40
3.5 System performance compared to Campbell’s rules. . . . . . . . . . . 40
3.6 A few of the most important features for various classifiers. . . . . . 41

4.1 The relative distribution of the most common null elements in Arabic
and English. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44
4.2 The distribution of nominal *wh*-traces by type of relative clause in the
Arabic Treebank training section . . . . . . . . . . . . . . . . . . . . 46
4.3 Function tagging confusion matrix for Arabic. . . . . . . . . . . . . . 57
4.4 Overall dependency evaluation for Arabic. . . . . . . . . . . . . . . . 58
4.5 Arabic null element performance by type. . . . . . . . . . . . . . . . 59
4.6 Performance for null element placement in Arabic for the system of Bakr et al. (2009) .................................................. 59

5.1 Gold standard and automatically-parsed test set results (F-measure) for the new and old English models by the typed-dependency metric. 74

5.2 Performance on wh-traces with overt wh-words compared to Shen (2006). ............................................................... 77

5.3 Performance on wh-traces with empty wh-words compared to Shen (2006) ............................................................... 78

6.1 Descriptions of the feature classes ................................................. 97

6.2 This table shows how, beginning from a minimal base system, the performance (F-measure) increases as feature classes are added in an order (roughly) from least complex to most complex. ................. 98

6.3 This table shows how performance (F-measure) changes if each class of feature is removed from the full system. ...................... 98

7.1 Distribution of wh-word type, overt and covert. ......................... 101

7.2 The counts used in determining the type of the null wh-word in “the man 0 I saw” ............................................................. 102

7.3 The counts used in determining the type of the null wh-word in “the time 0 I went” ............................................................. 102

7.4 Accuracy of the Google method on wh-type prediction. ............... 104

7.5 Results on section 23 for the baseline, thresholding, parser, and combined system approaches. ................................................. 115

7.6 Change in the performance of null element placement when the original pipeline (PARSER+SYSTEM) is augmented with Web 1T information (COMBINED+SYSTEM) ............................... 115
List of Figures

1.1 An example parse tree. .................................................. 2
1.2 An example parse tree with null elements. ......................... 2
1.3 An example of a raising construction. .............................. 5
1.4 An example of a subject control construction. ..................... 5
1.5 An example of an object control construction. ..................... 6
1.6 Examples of nominal and adverbial *wh*-traces. .................. 7
1.7 Examples of null *wh*-words ........................................... 7
1.8 Examples of topicalization of NP and VP. ........................... 8
1.9 Examples of “permanent predictable ambiguity.” .................. 9
1.10 Examples of right node raising. ..................................... 10
1.11 Examples of “insert constituent here.” ............................. 10
1.12 Examples of expletive *it*. ............................................ 10

2.1 One of Johnson’s patterns .............................................. 13
2.2 A diagram illustrating the idea of gap propagation as used in Model
    3 of Collins (1999). ...................................................... 17
2.3 An example of gap-threading ......................................... 19
2.4 Campbell’s pipeline of rules ......................................... 23
2.5 Campbell’s rule for inserting (NP *) ............................... 23
2.6 Features used by Levy and Manning ................................. 27

4.1 A simple Arabic relative clause. ................................. 50
4.2 Key for explanatory figures for graph creation. Boxes represent variables and circles represent factors.  .................................................. 51
4.3 Arabic graph example, part 1 .................................................. 51
4.4 Arabic graph example, part 2 .................................................. 52
4.5 An example of a trace with a resumptive pronoun within a –PRD. .... 52

5.1 Gold standard analysis for an example where the original model erroneously assigns NP *T* where NP * should be. ......................... 62
5.2 The erroneous original system analysis with an NP *T* where NP * should be. ................................................................. 62
5.3 A trace in the subject position of an infinitival relative. ................. 63
5.4 Key for explanatory figures for graph creation (English) ............... 66
5.5 Adding slot and wh-type variables (English) ............................... 67
5.6 Adding slot and wh-type variables (English) ............................... 68
5.7 Adding wh-path factors .......................................................... 69
5.8 A case of an (NP *) where the head word of the nearby ADJP–PRD indicates there is no coindexation. ..................................... 74
5.9 A sample output tree from binc (from the dev-test section) ........... 76

6.1 An example of an error classed as “significant parser failure.” ....... 80
6.2 Chart showing distribution of errors for nominal null wh-words ...... 81
6.3 An example where the parser analysis (above) appears superior to the analysis of the gold standard (below). The gold-standard analysis would imply that the calls were instrumental in stripping the stock markets. ....................................................... 82
6.4 Chart showing distribution of errors for adverbial null wh-words ... 83
6.5 A case of the parser erroneously inserting a relative clause due to the presence of time. Note that the tendency to place a relative clause after time is so strong it even outweighs the cost of using a rare $\text{VP} \rightarrow \text{NN \ VP}$ rule.

6.6 A case in which the trace placement is correct, but the metric counts it wrong because the parser was mistaken about the parent symbol. The parser/system output is above and the gold standard is below.

6.7 A case where the parser erroneously analyzes that as IN rather than WHNP.

6.8 A case where the parser fails to properly analyze a complex WHNP.

6.9 A case where a simple WHNP is incorrectly analyzed as if it were a more complex WHPP.

6.10 One of the few “miscellaneous” errors.

6.11 Here the parser erroneously creates a relative clause where none should be. The system output is above and the gold standard analysis is below.

6.12 Here call should have been parsed as having an S complement. The parser/system analysis is above and the gold standard is below.

6.13 Here the parser pulls should and one down into a VP. The system has such a strong inclination against allowing subjectless VPs that it incorrectly places the trace. The parser/system analysis is above and the gold standard is below.

6.14 Here the parser marks these days as an argument when it should be an adjunct with a -TMP function tag. Therefore the system sees do’s subcategorization frame as filled and falls back to placing the trace in subject position.

6.15 This is the only remaining case of (NP *T*)/(NP *) confusion in the development test section.

6.16 Chart showing distribution of errors for nominal wh-traces
6.17 A case where the gold standard analysis (shown) is wrong and the system output (not shown) is correct. 91

6.18 A case where the system output, though differing from the gold standard, is plausible. 92

6.19 In this case the trace dependencies in the system output (below; gold is above) are correct, but since the parser puts clean and repair in separate VPs, our second trace is counted as incorrect. 93

6.20 A case where the gold standard analysis (shown) does not coindex a wh-word, but our system does. 94

6.21 A more complicated case where the system output (below) has a trace for a wh-word which lacks one in the gold-standard (above). Note that the system currently does not handle right-node raising. 95

6.22 Chart showing distribution of errors for adverbial wh-traces. 96

7.1 An example relative clause. 100

7.2 An example relative clause with a null wh-word. 100

7.3 The accuracy of determining null wh-word types on the training data as a function of the threshold $\alpha$. 103

7.4 An infinitival relative. 105

7.5 An infinitive acting as an S modifier of a verb. 105

7.6 An infinitive acting as the complement of a noun. 106

7.7 Plot of all cases of infinitival Ss attaching to verbs. 111

7.8 Plot of all cases of infinitival Ss attaching as noun complements. 112

7.9 Plot of all cases of infinitival Ss attaching to nouns as relative clauses. 113

A.1 Example of an ambiguity in Johnson’s metric, part 1. 121

A.2 Example of an ambiguity in Johnson’s metric, part 2. 121

A.3 An example where Campbell’s metric would count a null element as incorrect due to an attachment error in the antecedent. 123
A.4 Dependency Extraction Example

127
Chapter 1

Introduction

The field of natural language processing has achieved a high degree of accuracy in parsing (assigning syntactic structures to sentences, as in Figure 1.1), at least in English. Understanding the syntactic structure of a sentence is a necessary preliminary to understanding its semantics and therefore for many practical applications. However, the syntactic structures produced by the most commonly used parsers\(^1\) are less detailed than those structures found in the treebanks the parsers were trained on.

In particular, the parsers do not recover two sorts of information present in all the Penn Treebanks (English, Arabic, Chinese, and historical). The first are annotations on constituents indicating their syntactic or semantic function in the sentence (Gabbard et al., 2006). For example, the parser will label a noun phrase in its output as simply NP, but the treebank annotation would distinguish an NP-SBJ acting as the subject of a sentence from an NP-TMP (e.g. “tomorrow,” “next week”) acting as a temporal adjunct.

The second kind of information, which the proposed dissertation will focus on, are tree nodes which do not correspond to overt (written or pronounced) words. Such

\(^1\)In particular, this is true of Collins (1999), Bikel (2004), and Charniak (2000), which are very commonly used. Parsers designed for richer formalisms like LFG, TAG, and CCG do generally provide more detailed output, but they lie outside the scope of this work.
This section describes the pattern-matching algorithm in detail. In broad outline the algorithm can

Figure 1.1: A parse of the noun phrase “the man Sam likes,” without null elements (Figure from Johnson)

Figure 1.2: A parse of the same noun phrase which includes null elements (Figure from Johnson)
nodes are often (though not always) associated with other (overt or covert) nodes in the tree by means of bearing common numerical indices (see figure 1.2). These nodes serve several purposes (discussed in detail below), but the most important is to indicate non-local relationships between words and phrases which cannot be encoded the context-free constituent structure produced by the parser: the null element indicates that the co-indexed constituent, which may be far away, should be interpreted as if it were in the element’s position. As Levy and Manning (2004) point out, since these non-local relationships are important for semantics, it is necessary that either a way be found to enrich CFG parser output with this information or else it will be necessary to move to parsers explicitly designed for deeper syntactic frameworks (as they put it, the question is whether “the context-free parsing model is a safe approximation”). This information is also of more immediate practical value, with potential benefit for anything using predicate–argument structures of some sort, including question answering and textual entailment.

In the remainder of this chapter, we will discuss the types of null elements found in English. In the following two chapters we will address the tricky question of how to evaluate this task and what approaches other researchers have tried. In the next chapter, drawing on the insights of previous work, we will present a system for the task in English. In the next chapter we will examine the problem in Arabic, which motivates the creation of a new model for the problem which, in the next chapter, we apply to English. We conclude with a chapter examining some ways of using a large corpus of unlabeled data to mitigate parser errors which cause problems for null element restoration.
Table 1.1: The frequencies of the most common null elements in sections 2-21 of the Penn Treebank (data from Johnson). Those of the form \( X \rightarrow Y \) mean a null element of type \( X \) co-indexed with an antecedent of type \( Y \).

<table>
<thead>
<tr>
<th>Null element</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>(NP *) ( \rightarrow ) NP</td>
<td>18,334</td>
</tr>
<tr>
<td>(NP *)</td>
<td>9,812</td>
</tr>
<tr>
<td>(NP <em>T</em>) ( \rightarrow ) WHNP</td>
<td>8,620</td>
</tr>
<tr>
<td><em>U</em></td>
<td>7,478</td>
</tr>
<tr>
<td>0</td>
<td>5,635</td>
</tr>
<tr>
<td>(S <em>T</em>) ( \rightarrow ) S</td>
<td>4,063</td>
</tr>
<tr>
<td>(ADVP <em>T</em>) ( \rightarrow ) WHADVP</td>
<td>2,492</td>
</tr>
<tr>
<td>(SBAR <em>T</em>) ( \rightarrow ) S</td>
<td>2,033</td>
</tr>
<tr>
<td>(WHNP 0)</td>
<td>1,759</td>
</tr>
<tr>
<td>(WHADVP 0)</td>
<td>575</td>
</tr>
</tbody>
</table>

1.1 Null Elements in the Penn Treebank

1.1.1 Units

The unit element *U* is used to indicate null units of measure, especially monetary ones (Bies et al., 1995, 4.5.1).\(^3\) Most often, they correspond to where a currency word is placed when a text is read aloud, e.g. “$1,000,000 *U*” is pronounced “one-million dollars.” There are a few more (relatively rare) complex cases for the placement and usage of units (see the guidelines). Although they are the third most common type of null element, some systems ignore them because they can be restored pretty well by simple rules and do not create non-local dependencies.

\(^2\)In the Treebank II format, the index is borne by the terminal symbol of the null element and the non-terminal symbol of what it is coindexed with. In later versions of the annotation guidelines, indices are always placed on non-terminal symbols.

\(^3\)Unless it is stated otherwise, all references in this section are to the Treebank II Guidelines (Bies et al., 1995)
Figure 1.3: An example of (NP *) in a raising construction. Here the (NP *) marks that the proposition which *seems* to be the case is “everyone dislikes Drew Barrymore.” (This and all following examples in figures in this section are from the annotation guidelines)

(S (NP-SBJ-3 Everyone)
  (VP seems
    (S (NP-SBJ *))
    (VP to
      (VP dislike
        (NP Drew Barrymore))))))

Figure 1.4: An example of (NP *) in a subject control construction. Here the (NP *) captures that Zaphod is promising that *Zaphod* (not Ford) will run for president.

1.1.2 Null Complementizers

In English, complementizers (roughly; words that introduce subordinate clauses) can often be omitted; these omitted complementizers are annotated as 0 (4.4). For example, you can say “I hope *that* dinner is ready” or “I hope 0 dinner is ready.” Like units, null complementizers are not especially interesting because they do not mediate non-local dependencies. However, there is an interesting and important subset of null complementizers, the null *wh*-words, which will be discussed below (1.1.4) with *wh*-movement.

1.1.3 PROs

The most frequent null element in the English treebank, *(NP *)* (which we will call PRO), has many uses. The simplest (arbitrary PRO) is as the subject of imperatives
(S (NP-SBJ Ford)
  (VP persuaded
    (NP-1 Zaphod)
  )
(S (NP-SBJ *(1))
  (VP to
    (VP run
      (PP-CLR for
        (NP president))))))))

Figure 1.5: An example of (NP *) in an object control construction. Here the (NP *) captures that Ford persuaded Zaphod that Zaphod (not Ford) should run for president.

("(NP *) Go away!")\(^4\) and in constructions where there is an understood pronoun of arbitrary reference, like “It is tough (NP *) to think carefully about St. Anselm’s ontological argument.” The second and most common use is to mark passivization, as in “(NP-1 Dante) was led (NP *-1) by Virgil.” The third primary use of PRO is in what linguists call control and raising constructions, for which see figures 1.3, 1.4, and 1.5. For the less common uses of PRO, see section 4.3 in the guidelines.

### 1.1.4 Wh-movement

Traces of *wh*-movement (\((NP *T*)\) with antecedents of category \(\text{WHNP, WHADVP, WHADJP,}\)
and \(\text{WHPP}\)) are used in the closely-related instances of questions and relative clauses to indicate in which argument or adjunct position the *wh*-word should be interpreted (4.2). For examples, see figure 1.6.

Closely related to them are those instances of null complementizers that replace *wh*-words in relative clauses (see figure 1.7).\(^5\) Determining that there is a missing *wh*-word is not hard, but determining if it is nominal or adverbial is a challenging problem for null element restoration systems.

\(^4\)School grammar sometimes calls this the “understood you.”

\(^5\)These null *wh*-words also occur in some places overt *wh*-words cannot, such as infinitival relatives (see Figure 1.7)
Figure 1.6: Examples of nominal and adverbial wh-traces.

Figure 1.7: Examples of null wh-words. On the top is an ordinary relative clause and on the bottom is a infinitival relative.
1.1.5 Topicalization

A *T* with other sorts of antecedents (e.g. NP, ADVP, VP, etc.) is used to indicate topicalization (4.2.3). Roughly, this is when an element is displaced from its usual position and put at the front of a sentence (see figure 1.8 for examples).

A particularly important subset of topicalization traces are the sentential traces, (S *T*), used to indicate when an S or SBAR from another part of a sentence occupies an argument slot. They are used frequently for either direct ((S-1 "I saw it yesterday") she said (S *T*-1)) or indirect speech ((S-1 The files were lost), he claimed (SBAR 0 (S *T*-1))). Note that in the case of indirect speech, the structure is complicated by the trace being wrapped in an SBAR together with a null complementizer (this is easy to understand if you “detransform” the sentence to “He claimed (SBAR that (S the files were lost)).”). Following Johnson, the whole SBAR in the indirect speech case is often treated as one big null element.

Identifying when one or the other of these two types of sentential traces should occur is not terribly difficult, but results for their recovery are depressed because they are not distinguished very consistently in the treebank (as Levy and Manning (2004) note).
Figure 1.9: Examples of a “permanent predictable ambiguity.” This is the classic example where “with the telescope” could, without a disambiguating context, modify “the man” or “saw.”

1.1.6 Ellipsed Predicates

*??* is used to indicate when it is not an argument or adjunct that has been moved or deleted, but rather a predicate (4.6) like a VP, PP-PRD, etc. This can happen in comparatives (“Acting would help him better than talking (VP *??*),” which is to say “Acting would help him better than talking would help him.”), conjunction (“Dianna likes tea, and I do (VP *??*) too”), and a variety of other cases (“Dianna likes tea, as do I (VP *??*)”). It is also used in some cases where the annotation guidelines do not otherwise specify how to fill the gap (the guidelines in section 4.6.3 give as an example “The plant cost about 50 million Canadian dollars to build (NP *??*)”).

1.1.7 Template gapping anti-placeholder

The last null element, *NOT*, is related to the interaction of gapping and coordination. It will not be discussed here, since it is complicated, extremely rare, and probably impossible to recover automatically (4.7).

1.1.8 Pseudo-attachments

The annotation guidelines distinguish a certain class of null elements that represent shared or ambiguous attachments, calling them pseudo-attachments instead (5.1). There are four of these. First is *PPA* (permanent predictable ambiguity; figure 1.9)
(S (NP-SBJ His dreams)  
  (VP had  
   (VP revolved  
    (PP-CLR around  
     (NP her))  
    (UCP-ADV (ADVP (ADVP so much)  
      (SBAR *RNR*-1))  
    )  
    (PP-TMP for  
      (NP (NP so long)  
        (SBAR *RNR*-1))  
      (SBAR-1 that...)))))

Figure 1.10: Examples of right node raising. Here the trailing SBAR should be interpreted as modifying both “so much” and “so long.”

(S (NP-SBJ (NP a young woman)  
  (SBAR *ICH*-1)))  
  (VP entered  
   (SBAR-1 (WHNP-2 whom)  
     (S (NP-SBJ she)  
      (PP-TMP at  
        (ADVP once))  
      (VP recognized  
        (NP *T*-2)  
        (PP-CLR as  
          (NP Jemima Broadwood))))))

Figure 1.11: Examples of “insert constituent here.” Here, “a young woman whom she at once...” has been split by the verb “entered.”

(S (NP-SBJ My teacher)  
  (VP said  
   (SBAR 0  
     (S (NP-SBJ (NP it)  
       (SBAR *EXP*-1))  
     )  
     (VP was  
      (ADJP-PRD OK)  
      (SBAR-1 for  
       (S (NP-SBJ me)  
        (VP to  
         (VP use  
          (NP the notes)  
          (PP-LOC on  
            (NP the test)))))))))

Figure 1.12: Examples of an expletive it. The null element indicates that this sentence is (basically) a rearranged version of “My teacher said for me to use the notes on the test was OK.”
which is used to indicate places where, even using the context, the annotator cannot
distinguish the correct attachment of a constituent. It is used only where the different
attachments actually change the meaning of the sentence (as opposed to “benign”
ambiguities). It is rare and very unlikely to be automatically recoverable. Second,
is *RNR* (right node raising; figure 1.10) which is used when a constituent needs to
be interpreted in multiple places in the same sentence. Third and most common is
*ICH* (insert constituent here; figure 1.11), which is used when a constituent is split
by other material being inserted into it. The last, *EXP* (expletive; figure 1.12) is
used when a clause has been displaced with an “it” present where the clause should
be interpreted.

1.2 Elements Under Consideration

Although we have above described many types of null elements in the Penn Treebank,
many of them are quite rare. In this work, will will focus our attention (in English)
on the nine non-unit categories in Table 1.1, since they account for the vast majority
of the empty categories in the treebank.
Chapter 2

Related Work

There has been a considerable amount of previous work on the topic of null element restoration, beginning with Collins (1999) and Johnson (2002) and continuing in several directions. In this chapter, we will survey this previous work by grouping it by the four main approaches researchers have taken: patterns, parsing, rules, and machine learning. We will conclude by framing the approach taken in this thesis with respect to previous attempts.

2.1 Pattern-Matching

The seminal paper on the general null element problem is Johnson (2002). Johnson’s approach, which he notes “may be regarded as an instance of . . . Memory-based Learning,” consists of extracting patterns from the Penn Treebank and then matching them against the trees we wish to restore null elements to. Johnson defines a pattern as a “minimal connected tree fragment containing an empty node and all nodes co-indexed with it.” A pattern $P$ matches a tree $T$ if $T$ is an extension of $P$ ignoring $P$’s empty categories.
Figure 2.1: The pattern resulting from doing pattern extraction on Figure 1.2 (Figure from Johnson)

2.1.1 The Training Phase

During the training phase, the system goes through each tree in the corpus and, for every null element, extracts the minimal connected tree which contains it and every node co-indexed with it (a pattern; see figure 2.1). If there are no nodes co-indexed with it, then its parent and siblings are extracted. At this point we have a list of patterns and how many times they each occurred (indicated by \( c_p \) for a pattern \( p \)). This results in about 11,000 patterns.

Next, the system counts how many times each pattern matches in the treebank, called the match value (\( m_p \)). Note that since matching ignores empty categories in the pattern, a pattern may match places in the treebank which are identical to it except for null elements. There are a number of ways to count the pattern matches for determining the match value. The simplest is just to count how many times each pattern matches when applied as often as possible without regard to other patterns. However, if one pattern is a subtree of another pattern (ignoring null elements), both will match, but it is not the case that both could actually be applied. Therefore the naive approach tends to favor “shallow” trees over “deep” trees. To fix this, the system walks through the nodes in a pre-order traversal, attempting to match
patterns at each. Afterward, it chooses whatever pattern would have been correct to apply, if any, and applies it, inserting the appropriate null elements (see section 2.1.2). The presence of these null elements may then block shallower patterns from being applied within the “domain” of this deeper pattern. Johnson notes that this change has a large effect on performance.

Having calculated the counts and match values, patterns are now pruned. This is necessary because some patterns would insert null elements incorrectly more often than they would correctly (that is, the success probability \( \frac{c_p}{m_p} < \frac{1}{2} \)). For each pattern, a statistical technique is used to throw out those patterns we cannot be confident truly have a success probability greater than a half (this is needed because some rare patterns may have such a success probability observed in our training sample by accident). After pruning, about 9,000 patterns remain.

Finally, if more than one pattern can apply at a node, which should be chosen? The patterns are ranked by depth, and the system at runtime will prefer to apply deeper patterns before shallower ones. Johnson also notes that he tried ranking patterns by success probability with very similar results.

### 2.1.2 The Application Phase

To restore empty categories to a tree, the system does a pre-order traversal. At each node, it checks which patterns, if any, match and applies the highest ranked one. To apply a pattern, it replaces the matching subtree with the contents of the pattern, renumbering null element indices if necessary to prevent accidental collision with coindexation already in the tree.

### 2.1.3 The Preprocessor

Before both training and runtime, the trees are modified slightly. First, the part-of-speech tags for auxiliary verbs are changed to match those produced by Charniak’s parser. This is simply for convenience and seems to have little effect on performance.
Table 2.1: The performance of Johnson’s system, by his metric (Table from Johnson)

More importantly, the part-of-speech tags of transitive verbs have a “t” appended to them. A verb is determined to be transitive if more than half the time it is followed by a noun phrase which does not carry a function tag marking it as a non-argument. Johnson notes than experiments on the development test set showed a small improvement from this annotation.

2.1.4 Evaluation

Results from the system can be see in table 2.1. The relative performance of the different null elements set the basic pattern for future work. Units, non-wh null complementizers, and sentential traces are recovered relatively well; nominal wh-traces moderately well; and adverbial traces and null wh-words poorly. (NP *) proves easy to insert, but very difficult to find the antecedent for. Results from other systems, while having trouble in the same places, have generally been better. In part, this is likely due to Johnson’s patterns being less robust – both against parser errors and in the broader sense of generalizability – than later approaches.
2.1.5 “Looser” Pattern-Matching

Another system which uses some form of pattern-matching is Jijkoun and de Rijke (2004), which used memory-based learning. They will not be discussed in further detail here since they operate only on dependency structures and report scores very similar to Dienes and Dubey (2003b), who will be discussed next.\footnote{It is not clear that their numbers are in fact comparable to those of Dienes and Dubey on parsed data because the metrics used are not quite equivalent, particularly for (NP *)s: among other differences, unlike Jijkoun and de Rijke’s dependency metric, Dienes and Dubey’s is sensitive to the string extent of the antecedent node, penalizing them if the parser makes attachment errors involving the antecedent even if the system recovered the long-distance dependency itself correctly.}

2.1.6 Regular Expression Patterns

Filimonov and Harper (2007) present another pattern-based system which achieves significantly more robustness than Johnson’s by means of handwritten patterns (with automatically assigned probabilities) which are made more flexible in a manner rather analogous to regular expressions. Since this system is both rather complicated and very focused (limiting itself to only wh-traces with overt antecedents), we will not discuss it in further detail here.

2.2 Parsing

It is appealing to attempt to recover null elements within the parser. After all, finding null elements is properly part of the task of syntactic analysis the parser is supposed to perform. Indeed, one of the seminal dissertations in modern parsing (Collins, 1999) treated the recovery of wh-traces in its third, most complex model. We will describe it briefly in this section under the assumption the reader is familiar with Collins’s Model 2; for those who are not, we refer them to Collins’s thesis.

Model 3 begins by annotating the training trees with gap annotations in the manner of Gazdar et al. (1985). For every non-terminal on the path between a
Figure 2.2: A diagram illustrating the idea of gap propagation as used in Model 3 of Collins (1999). Every node between the trace and its antecedent is annotated with +gap. Figure from Collins’s thesis.

trace and its antecedent, a +gap feature annotation is added. The parsing model is then modified to take this into account: in addition to indicating whether the usual constituents like NPs are expected, subcategorization frames can now also contain gaps. These gaps may be discharged either by producing a trace or by producing an ordinary non-terminal which has the +gap feature, and the node and head generation probability models are modified accordingly. The question remains, however, of whether a symbol looking for a gap below it should add it to the left subcategorization frame, the right subcategorization frame, or neither (in which case the gap feature would be passed on to the immediate head of the symbol). This is modeled by the addition of a new probability distribution $P_G$ whose values are the three options above and which is conditioned on the parent symbol, head symbol, and head word.

In more recent work, Model 3 has been extended by Dienes and Dubey (2003b), which we will now consider. These authors propose two possible methods which we might call partial and full parser integration. In partial integration, a finite-state
method is applied to the surface string to insert null elements, and the sentence is then parsed treating the null elements just like they were normal words. In complete integration, there is no preliminary step, and null element insertion is done entirely in the parser.

2.2.1 Partial Integration

The first step in this approach is to use a finite-state “trace tagger” to mark the positions of the null elements in the surface string. Dienes and Dubey (2003a) had previously presented such a tagger which achieves a 79.1% F-score on null element detection. The tagger primarily employs three pieces of information:

- the part-of-speech tags in a five word window
- lexical features in a three word window
- non-local features which look through the string for signs of passives, to-infinitives, gerunds, *wh*-words, and “that.”

They note that the first class of features is their most informative.

The second step of this approach is to use a parser to find the antecedents of those null elements inserted in the first step. In order to do this they modify the training trees for the parser according to a variation of the gap-propagation technique described above: for every null element, every non-terminal node between it and (up to but not including) its antecedent’s parent has $\text{gap+<gap-type>}$ appended to its label (for an example, see Figure 2.3). Note that a label may receive more than one gap annotation.

At runtime, the parser is run on the output of the trace tagger, treating null elements just like ordinary overt words. The results are then interpreted in the opposite manner from the transformation of the training data: from a null element, if you follow the path of nodes dominating it until you reach a node which does
not bear gap annotation for the element, you know that node should dominate its antecedent. Knowing the node which dominates the antecedent is not quite the same as knowing the antecedent, of course, so they use a simple deterministic algorithm to choose the appropriate child. They note that on gold-standard trees stripped of node indices this method had an F-score of 95% at finding the correct antecedent.

### 2.2.2 Full Integration

In this approach there is no trace tagger, and thus the parser is not informed of the location of null elements. The authors try both unlexicalized and lexicalized parsers. In the unlexicalized case, they use a parser of their own, while in the lexicalized case they extend Model 3 from Collins (1999) with the idea of a “gapcat” frame analogous to the subcategorization frames already used by the parser.

The gapcat frames work as follows.\(^2\) If a node should have a gap associated

\(^2\)In this paragraph we again assume knowledge of Collins’ Model 2 (Collins, 1999).
with it, that gap is part of its gapcat frame set. This set is treated analogously to the subcategorization frame, but unlike subcategorization frames, which have elements discharged whenever a complement modifier is generated, gapcat frames have their elements only discharged when null element modifiers are generated (null elements generated as complements will also discharge subcat frame items). Since non-terminals have gaps indicated on them, the threading of gaps from one level to another is effectively accomplished by the inclusion of the gapcat frame in the conditioning of the modifier generation probabilities.

2.2.3 Evaluation

A parser-integrated approach must be evaluated in two respects: first, its performance on the null element task itself, and second, on the overall performance of the parser (both in accuracy and computational resources), since the approach will not be useful if it impairs the overall parsing task. We will consider these two aspects of evaluation in reverse order.

The authors find the fully-integrated approach to be entirely intractable for unlexicalized parsing (it cannot find any parse at all for 35% sentences in section 23), so we will focus on the lexicalized case. The core challenge here, of course, is the explosion (by a factor of 7) in the size of the the non-terminal alphabet due to all the gap annotations. The authors claim that this results in the familiar sparse data problem (that is, probabilities involving non-terminal symbols can no longer be estimated as accurately because training instances which were formerly considered together are “shattered” into different classes) and that it has a significant negative impact on parsing performance in both the fully and partially integrated case. The performance of the fully and partially-integrated cases is almost identical (86.6 and 86.4 F-measure, respectively), but this is a 12-13% increase in error relative to the same parsing model without null elements (88.0).
<table>
<thead>
<tr>
<th>Type</th>
<th>EE detection parser</th>
<th>tagger</th>
<th>Antecedent rec. parser</th>
<th>tagger</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP–NP</td>
<td>80.4% 83.5%</td>
<td>70.3% 70.7%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WH–NP</td>
<td>81.5% 83.2%</td>
<td>80.2% 82.0%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRO–NP</td>
<td>64.5% 69.5%</td>
<td>64.5% 69.5%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WH–S</td>
<td>92.0% 92.8%</td>
<td>82.2% 84.5%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WH–ADVP</td>
<td>57.9% 59.5%</td>
<td>53.0% 53.6%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2: Comparison of null element performance for DD’s partially (tagger) and fully (parser) integrated systems. The format of the node types is antecedent-element. PRO-NP indicates an uncontrolled (NP *), while wh-s (confusingly) indicates a sentential trace. (Table from DD)

In terms of the relative performance of the partially and fully-integrated approaches on the null element task itself, the partial approach is consistently superior (see table 2.2). The authors hypothesize that this is because the tagger’s five-word window gives it access to useful lexical information which crosses phrase boundaries. They provide a comparison only to Johnson (the only system available at the time), whom they generally outperform by a significant margin (see section 2.4.2 for an comparison to Levy and Manning).

### 2.2.4 Better Unlexicalized Results

Dienes and Dubey noted that unlexicalized parsing failed completely for their system, presumably because the search space was insufficiently constrained without lexical information. Schmid (2006) attempts to fix this problem by using BitPar (Schmid, 2004) in order to exhaustively search all possible parses.

Schmid first annotates all trees with the features of Klein and Manning (2003) and then adds several related to null element prediction, most notable the slash features of Dienes and Dubey, certain function tags, subcategorization features, a feature for object control verbs, features noting the words which commonly indicate a relative clause with a null wh-word is adverbial, and a feature for expletive it. He
also corrects some part-of-speech errors in the training data.

Parsing time for the test section\textsuperscript{3} was a bit less than three hours, which is a good deal slower than the Collins parser. There is a considerable payoff for the extra time in that the system achieves a state-of-the-art unlexicalized parsing score of 86.6 and achieves very good null-element results (11\% better than Dienes and Dubey and 3\% better than Campbell). However, as Schmid notes, the system’s parsing performance still lags around three points behind the best lexicalized parsers, which is a significant problem for practical applications. The work is nonetheless interesting for demonstrating that null element restoration can in principle help at least unlexicalized parsing (by roughly half a point).

\section{2.3 Rules}

\subsection{2.3.1 Technique}

The only published handwritten rule-based system for the null element problem is that of Campbell (2004). Campbell’s approach is motivated by his observation that the null element problem should differ from those for which data-driven methods have been so successful since “for the most part, their location and existence is determined, not by observable data, but by explicitly constructed linguistic principles which were consciously used in annotation.”

Campbell’s system is straightforward. The system walks through a tree in pre-order traversal, and at each node it attempts to apply the rules in Figure 2.4. Each of these rules makes a decision based on a logical combination of linguistic predicates; he mentions passivization, finiteness, headedness, function words, and syntactic function\textsuperscript{4} as particularly important pieces of information (for an example, see figure 2.5).

\textsuperscript{3}The authors say their timing was done “on a Dual-Opteron system with 2.2 GHz CPUs.”

\textsuperscript{4}Since function tags are not generally present in parser output, other rules are present to provide them.
for each tree, iterate over nodes from top down
  for each node X
    try to insert NP* in X
    try to insert 0 in X
    try to insert WHNP 0 or WHADVP 0 in X
    try to insert *U* in X
    try to insert a VP ellipsis site in X
    try to insert S*T* or SBAR in X
    try to insert trace of topicalized XP in X
    try to insert trace of extraposition in X
  for each node X
    try to insert WH-trace in X
  for each node X
    try to insert NP-SBJ * in finite clause X
  for each node X
    if X = NP*, try to find antecedent for X

Figure 2.4: Capmbell’s pipeline of rules. (Figure from Campbell)

if X is a passive VP & X has no complement S
  if there is a postmodifying dangling PP Y
    then insert NP* before all postmodifiers of Y
  else insert NP* before all postmodifiers of X
else if X is a non-finite S and X has no subject
  then insert NP-SBJ* after all premodifiers of X

Figure 2.5: The rule for inserting (NP *). (Figure from Campbell)
Notably, content words are hardly used at all.\(^5\)

The only two structurally complicated rules are these for finding \textit{wh}-traces and the antecedents of \((\text{NP } *)\). Both of these start from the known location (the \textit{wh}-word or \((\text{NP } *)\)) and walk through the tree, node by node, until they find an appropriate place to insert a trace or choose an antecedent, respectively.

\subsection*{2.3.2 Evaluation}

Unfortunately, Campbell provides results broken down by type only for gold standard data, which makes comparison more difficult. He does provide an aggregate number of 76.7\% across all null elements; this betters Johnson significantly (68.0\%) and Dienes and Dubey moderately (74.6\%). This seems to support his contention that learning-based methods are not clearly superior for this task. He does note two cases where there seems to be room for a learning-based system to make use of lexical information. First, in distinguishing between the placement of \((\text{NP } *)\) and \((\text{NP } *\text{T*})\) in certain infinitives.\(^6\) Second, in determining the antecedent (or lack thereof) of \((\text{NP } *)\), which he notes is a less rule-governed task, even in the annotation guidelines.

\subsection*{2.4 Machine Learning}

Levy and Manning (2004) present the null element problem as a task of long-distance dependency recovery.\(^7\) In particular, they note that while most “deep” syntactic frameworks (e.g. “GB, CCG, HPSG, LFG, [and] TAG”) have a central context-free component for representing “surface” syntactic structure, those frameworks also recognize that such representations alone are inadequate for complete syntactic analysis. This could be a serious problem for current common CFG-based techniques in

\(^5\)The one exception to this is that there are a small number of content words which, if they precede a null complementizer, will indicate it is adverbal.

\(^6\)These are the cases of raising, control, and exceptional case marking familiar from Intro to Syntax.

\(^7\)This same work is also presented in Levy’s dissertation (Levy, 2005).
NLP unless there’s a way found to bridge the gap between the context-free “surface”
dependencies they can provide us with and the full representations with “hidden”
dependencies which are needed for proper modeling of language. As they put it, is
CFG parsing “a safe approximation” to a full, deep linguistic analysis?

A particularly notable aspect of this paper is that in addition to the usual Penn
Treebank WSJ evaluation, they consider the problem for the German NEGRA cor-
pus (Skut et al., 1997). NEGRA is primarily a dependency corpus, but there is a
version available which transforms it into a phrase-structure representation where
what would be discontinuous constituents are handled by using traces to mark part
of a phrase as dislocated.

2.4.1 Pipeline

This system structures null element recovery as a pipeline where each stage performs
operations based on the decisions of a maximum entropy classifier (see below). The
order of the pipeline is important for much the same reasons as already discussed
with respect to Campbell’s work. For English, the pipeline is as follows (each step
is done on all tree nodes before the next step is begun):

1. For every tree node, determine if a null complementizer should be inserted
   under it (IdentNull). If one should be inserted, decide at what position and
   place it in the tree (InsertNull).

2. Classify every tree node as to whether or not it is dislocated (IdentMoved).
   Then, for each node which is dislocated, choose what node it came from (Re-
   locMoved). Finally, insert the trace into the tree (InsertReloc).

3. For every tree node, determine if an (NP *) should be inserted under it (Ident-
   Locus). If so, insert it in the appropriate position (InsertLocus). Finally,
   determine what its controller, if any, is and co-index it appropriately (Find-
   Controller).
Due to the simpler annotation of German, only the second step is used on the NEGRA corpus.

2.4.2 Learning

Each individual decision above is made by a maximum entropy classifier, but the classification problem may take on slightly different forms. Choices with a yes or no form are simple binary classifications done at each node. Choices among nodes or locations (e.g. what is the controller of this (NP *)?) apply a binary classifier to each probability and choose the option with the highest score for the positive classification. In the particular case of finding controllers (where there may in fact be no controller), a special dummy or null option is added.

The features used by their classifiers may be found in Figure 2.6. Two of them require further information. PATH is the sequence of categories on the path between two nodes, along with the direction the path is “moving” at each node. The “# special” line indicates how many custom feature templates were made for that classification task alone which were not shared with other classifiers.

2.4.3 English evaluation

The authors provide two evaluations and in neither is the technique particularly effective. The first, according to Johnson’s metric, can be found in Table 2.3. On parsed output from Charniak’s parser, their system is a significant improvement over both Johnson and Dienes and Dubey for null complementizers. In all other respects, though, the system is less impressive. For sentential traces, it lags both of the other systems. While it modestly improves performance on (NP *)s over Johnson, it lags significantly behind Dienes and Dubey. Most worrisome is its poor performance.

---

8They used both feature thresholding and L2 regularization.
9For a discussion of metrics for this problem, see appendix A
10The system of Campbell had not yet been published.
Figure 2.6: Feature used by LM. Along the top are the classifiers used and along the sides are features templates. ⊗ means they used all subsets of that feature template. The prefixes M, D, G, and R stand for “mother,” “daughter,” “grandparent,” and “relative” (that is, antecedent), respectively. POS indicates the position of a node among its mother’s children, while TAG indicates the head word’s part-of-speech tag. (Figure from LM)
Table 2.3: Performance of LM’s system (Pres) compared to Johnson (Jn) and DD according to Johnson’s metric. (Table from LM)

<table>
<thead>
<tr>
<th></th>
<th>Gold trees</th>
<th>Parser output</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Jn Pres</td>
<td>Jn DD Pres</td>
<td></td>
</tr>
<tr>
<td>NP-*</td>
<td>62.4</td>
<td>75.3</td>
<td>55.6 (69.5) 61.1</td>
</tr>
<tr>
<td>WH-t</td>
<td>85.1</td>
<td>67.6</td>
<td>80.0 (82.0) 63.3</td>
</tr>
<tr>
<td>0</td>
<td>89.3</td>
<td>99.6</td>
<td>77.1 (48.8) 87.0</td>
</tr>
<tr>
<td>SBAR</td>
<td>74.8</td>
<td>74.7</td>
<td>71.0 73.8 71.0</td>
</tr>
<tr>
<td>S-t</td>
<td>90</td>
<td>93.3</td>
<td>87   84.5 83.6</td>
</tr>
</tbody>
</table>

Table 2.4: Performance of LM’s system (A) compared to Johnson (J) and DD (D). P indicates the Charniak parser and G indicates the gold-standard. J ∘ P indicates running Johnson’s system on the parser output. (Table from LM)

<table>
<thead>
<tr>
<th></th>
<th>PCF</th>
<th>P</th>
<th>A ∘ P</th>
<th>J ∘ P</th>
<th>D</th>
<th>G</th>
<th>A ∘ G</th>
<th>J ∘ G</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td>91.6</td>
<td>87.6</td>
<td>90.5</td>
<td>90.0</td>
<td>88.3</td>
<td>95.7</td>
<td>99.4</td>
<td>98.5</td>
</tr>
<tr>
<td>S</td>
<td>93.3</td>
<td>83.4</td>
<td>91.2</td>
<td>89.9</td>
<td>89.2</td>
<td>89.0</td>
<td>98.0</td>
<td>96.0</td>
</tr>
<tr>
<td>VP</td>
<td>91.2</td>
<td>87.3</td>
<td>90.2</td>
<td>89.6</td>
<td>88.0</td>
<td>95.2</td>
<td>99.0</td>
<td>97.7</td>
</tr>
<tr>
<td>ADIP</td>
<td>73.1</td>
<td>72.8</td>
<td>72.9</td>
<td>72.8</td>
<td>72.5</td>
<td>99.7</td>
<td>99.6</td>
<td>98.8</td>
</tr>
<tr>
<td>SBAR</td>
<td>94.4</td>
<td>66.7</td>
<td>89.3</td>
<td>84.9</td>
<td>85.0</td>
<td>72.6</td>
<td>99.4</td>
<td>94.1</td>
</tr>
<tr>
<td>ADVP</td>
<td>70.1</td>
<td>69.7</td>
<td>69.5</td>
<td>69.7</td>
<td>67.7</td>
<td>99.4</td>
<td>99.4</td>
<td>99.7</td>
</tr>
</tbody>
</table>

performance on *wh*-traces, where it increases error over the other systems by a large margin.11

In the second evaluation, they compare the systems by overall typed dependency accuracy (Table 2.4). Here they show a 5% improvement over Johnson and a large improvement compared to Dienes and Dubey. However, as discussed in section A.3.1, this is may well be due to the Charniak parser strongly outperforming the integrated parser of the latter system, and not due to any superiority of the learning system at the null element task (especially given the results of the first evaluation, although it is true this second evaluation includes certain rarer null element types not addressed in the first).

11However, the differences are not quite as large as they appear in the paper itself, where they compare aggregate statistics for certain elements for their system with non-aggregate statistics for the other systems. This actually makes their system appear a bit worse than it is.
2.4.4 German Evaluation

Since there are no other German system to compare their results against, the authors provide a comparison of their system’s performance on German to its performance on English. This is particularly interesting because they can compare only non-relativization dislocations, which is to say precisely those aspects of the English task generally given less attention in other papers. To do a fair comparison, they used plain PCFG parsers for both languages (since state-of-the-art English Penn Treebank parsing was much better than state-of-the-art German NEGRA parsing) and also present results using for training a subset of the PTB WSJ corpus equal in size to NEGRA’s training corpus (“WSJ(sm”)”). In the results German lags English parsed performance by 65% (increase in relative error) and English gold performance by very large margin. However, as careful as the authors have been, they note that it is still unclear how meaningful this comparison is since node dislocation simply may be serving different purposes in the two languages.

We may still note two interesting things related to English from this data. First, it is interesting that their English performance is so high for null element relations other systems have generally found difficult. This may perhaps be due to the frequency of relatively frequent and relatively easy sentential traces among the dislocations they were considering, but it could also reflect the fact that their system models dislocations more directly than others by classifying nodes according to whether or not they appear to be out-of-place in their current location. Second, the results are nearly identical for the system regardless of whether the large or small training set is used, which lends support to Campbell’s claim that much of this task is more fundamentally rule-based than learning-based.
2.5 Conclusion

2.5.1 Division of the Problem

The general term null element encompasses a wide variety of syntactic phenomena, so it is unsurprising that the systems split up the problem slightly differently. Some methods attempt broad coverage through a “one-size-fits-all” technique, such as Johnson’s patterns or Dienes and Dubey’s gap threading. However, other systems vary their methods by individual types of null elements (Campbell) or classes of null elements (Levy and Manning). We should expect the latter method to lead to better results, and Campbell’s high-performing system provides some evidence for this.

On a related note, the reporting of aggregate results alone, as Campbell does for his system on parsed data, should be discouraged since it makes it difficult to determine where the performance improvement of his system relative to others is coming from (performance differences on the gold standard are not always perfectly reflective of performance differences on parsed data, which also reflects the robustness of approaches). It also makes impossible comparison with specialized systems that cover only a subset of null elements (Gabbard et al., 2006; Filimonov and Harper, 2007).

2.5.2 Annotation Inconsistency

Johnson and Levy and Manning both note inconsistent annotation in the treebank, especially regarding three cases: antecedents of (NP *) are often not marked or are marked incorrectly, the distinction between the two types of sentential trace is not maintained consistently, and adverbial null wh-words are often not marked as such.

Correcting the treebank by simply checking relevant cases would be easy in the last case, moderately difficult in the second case, and very time-consuming in the first. However, a null element system itself could be useful both for finding annotation errors and for preprocessing sentences to be annotated to increase both speed and
2.5.3 Efficacy

What is the best approach to finding null elements? Of the options presented, we can quickly eliminate Johnson’s pattern matching; while it was very valuable for introducing the problem and setting a baseline, it is generally outperformed by all the other approaches. That leaves three viable options: Campbell’s hand-written rules, Levy and Manning’s machine-learning, and Dienes and Dubey’s parser integration. Leaving aside the last for the moment, Campbell’s system is clearly superior in performance to that of Levy and Manning. However, there remain a few reasons to think machine-learning approaches might be the best way forward. First, as Campbell notes, there are remaining cases where the lexical information available from machine-learning approaches could be valuable. Second, any sort of linguistic predicates and rules available to the rule-based approach can easily be integrated into the machine-learning framework. While it might be somewhat inelegant to learn from features what could be stated by rule, it is certainly possible and perhaps preferable to trying to build a hybrid system. Finally, while writing hand-tuned rules is fairly easy for English, it is more challenging in the case of a language the researcher does not know and for which annotation guidelines may not be as detailed. In this case, machine learning can make system development easier.

How do post-processing and parser-integrated approaches compare? While the results of Schmid (2006) suggest that the parser-internal approach has the potential for excellent performance on the null element task and perhaps even for a modest improvement in overall parsing performance, no one has yet succeeded in integrating null element restoration into one of the leading parsing models (e.g. Charniak and Collins) without hurting overall parsing performance, and few downstream users are likely to trade several points of overall parsing performance for null elements.
At this point, the advantage seems to lie with post-processing approaches. These have the additional advantage that they can be applied to the output of any Penn Treebank-style parser, which makes them more convenient for integration into existing pipelines.

2.5.4 Appendix: Lexicalized Tree-Adjoining Grammar

As mentioned above (section 1), several syntactic frameworks more powerful than CFG parsing deal with certain aspects of the null element problem in an integrated way. While this dissertation will in general discuss only work in the CFG parsing stream of research, in section 5.7 we will compare our results on wh-traces to one recent representative of the more powerful frameworks, the Spinal Lexicalized Tree-Adjoining Grammar (LTAG-Spinal) parser of Shen (2006). Describing LTAG-Spinal is beyond our scope, and we refer interested readers to Shen (2006).
Chapter 3

A Null Element System for English

In this chapter,\textsuperscript{1} we present a system for the null element problem in English which seeks to combine the linguistic insight of Campbell (2004) with learning methods similar to those of Levy and Manning (2004). We will begin by describing the behavior of the system at runtime. We will then examine the feature set and discuss how the model is trained. Finally, we will present the performance of the system and discuss some possible ways to improve it.

3.1 Runtime

The algorithm applies a series five linear classifiers. Before presenting the pipeline in detail, we will briefly mention each classifier:

- **NULLCOMP** deals with 0.
- **WHXINSERT** deals with inserting \((\text{WHNP 0})\) and \((\text{WHADVP 0})\).
- **WHXPDICERN** deals with distinguishing between \((\text{WHNP 0})\) and \((\text{WHADVP 0})\).
- **WHTRACE** deals with \((\text{NP *T*})\) and \((\text{ADVP *T*})\).

\textsuperscript{1}An earlier version of this chapter was published as the second half of Gabbard et al. (2006)
• NPTrace deals with placement of (NP *).

• PROAntecedent and Antecedentless deal with coindexation for (NP *).

The details of the application of the classifiers is as follows:

1. For each PP, VP, and S node in the tree, ask the classifier NPTrace to determine whether to insert an (NP *) as the object of a preposition, an argument of a verb, or the subject of a clause, respectively.

2. For each node in the tree, ask NullComp to determine whether or not to insert a 0 to the right.

3. For each S node in the tree, ask WHXPIsinsert to determine whether or not to insert a null wh-word to the left. If one should be inserted, ask WHXPDis- cern to decide if it should be a (WHNP 0) or a (WHADVP 0).

4. For each S which is a sister of WHNP or WHADVP, consider all possible places beneath it (i.e. places c-commanded\(^2\) by the WHNP or WHADVP) where a wh-trace could be placed. Score each of them using WHTRACE, and insert a trace in the highest scoring position.

5. For any S lacking a subject, insert (NP *).

6. For each (NP *) in subject position, look at all NPs which c-command it. Score each of these using PROANTECEDENT, and co-index the (NP *) with the NP with the highest score. For all (NP *)s in non-subject positions, we follow Campbell in assigning the local subject as the antecedent.

7. For each (NP *), ask Antecedentless to determine whether or not to remove the co-indexing between it and its antecedent.

\(^2\)A node a c-commands a node b if a’s parent dominates b, but a does not dominate b.
The sequencing of classifiers and choice of how to frame the classification decisions closely follows Campbell with the exception of finding antecedents of (NP *)s and inserting *wh*-traces, which follow Levy and Manning in using a competition-based approach. Also, rather than introducing an extra zero node for uncontrolled (NP *)s, we always assign an antecedent and then remove co-indexing from uncontrolled (NP *)s using a separate classifier.

3.2 Feature Set

The following is the common feature set used by all the classifiers:

- the local features of the focus node and its daughters, left and right sisters, mother, aunts, and grandmother (the local features of a node are its non-terminal or terminal symbol, whether or not it is a non-terminal, its head word and head part-of-speech, its function tags (see section 3.2.1), whether or not it is an argument, and if an S or SQ, whether or not it has an overt subject)
- whether the focus node is the first or last daughter of its mother
- the number of daughters of the focus node which are arguments\(^3\)
- the conjunction of the focus node’s head word with the number of its daughters which are arguments
- the focus node’s great-grandmother’s non-terminal symbol
- whether the focus node is an S with a subjectless infinitive
- whether the focus node is a VP with a logical subject (i.e. a *by*-phrase)
- the token distance from the focus node to the last closing quotation mark

---

\(^3\)Argument annotation is applied to training trees using the rules from Collins’s parser (Collins, 1999); on automatically parsed data, they are provided by the modified parser discussed in section 3.2.1.
• whether the focus node is inside a parenthetical

• whether and how the quotation marks in the sentence match up

The following features are used for **PROANTECEDENT** only:

• the path between the (NP *) and its proposed antecedent

• the path length

• what non-terminal symbols are contained somewhere along the path.

The following features, each conjoined with the type of *wh*-trace being sought, are used for **WHTRACE** only:

• the sequence of categories found on the path between the trace and its antecedent

• the path length

• which categories are contained anywhere along the path

• the number of bounding categories crossed and whether the trace placement violates syntactic constraints on *wh*-trace extraction

• whether or not the trace insertion site’s parent is the first verb on the path

• whether or not the insertion site’s parent contains another verb beneath it

• if the insertion site’s parent is a verb, whether or not the verb is saturated.\(^4\)

---

\(^4\)To provide the verb saturation feature, we calculated the number of times each verb in the training corpus occurs with each number of NP arguments (both overt and traces). When calculating the feature value, we compare the number of instances seen in the training corpus of the verb with the number of argument NPs it overtly has with the number of times in the corpus the verb occurs with one more argument NP.
Table 3.1: A list of the function tags in the Penn Treebank which are of interest to us. Adapted from a table by Seth Kulick (Gabbard et al., 2006).

3.2.1 Function tags

One of the most notable departures of this feature list from previous ones is in the use of function tags (table 3.1) and argument markings, which were previously ignored for the understandable reason that though they are present in the Penn Treebank, parsers generally do not produce them. Function tags indicate whether constituents (most importantly, noun phrases) are arguments such as subjects, direct objects, and dative objects, or adjuncts of various types such as temporal, location, and manner (Bies et al., 1995).

We gain access to function tags and argument markings through a modified version of the Bikel implementation of the Collins parsing model (Bikel, 2004) provided to us by Seth Kulick (Gabbard et al., 2006). Gaining access to argument markings is very simple: they are used internally by the parser and deleted in a post-processing step, so simply skipping this step is sufficient. Similarly, during the training of the parser, function tags are normally stripped off of non-terminals; Kulick’s approach is to remove this preprocessing step so that, for example, NP-SBJ and NP-TMP are treated as separate atomic non-terminal symbols. While this presumably causes some data sparsity, it appears to give a compensating improvement so that overall

---

Table 3.2: F1 scores for our system (Pres), Johnson’s (J), and Levy and Manning’s (LM) on gold standard trees from section 23 using Johnson’s metric, together with the number of occurrences (O) of each category on sections 2-21.

<table>
<thead>
<tr>
<th>Category</th>
<th>Pres</th>
<th>LM</th>
<th>J</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combined 0</td>
<td>99.3</td>
<td>99.6</td>
<td>89.3</td>
<td>7,969</td>
</tr>
<tr>
<td>NP *</td>
<td>87.5</td>
<td>75.3</td>
<td>62.4</td>
<td>28,146</td>
</tr>
<tr>
<td>WHNP</td>
<td>90.9</td>
<td></td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>WHADVP</td>
<td>71.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Combined wh-trace</td>
<td>89.8</td>
<td>67.6</td>
<td>85.1</td>
<td>11,112</td>
</tr>
<tr>
<td>NP <em>T</em></td>
<td>91.8</td>
<td></td>
<td>90</td>
<td>8,620</td>
</tr>
<tr>
<td>ADVP <em>T</em></td>
<td>84.0</td>
<td></td>
<td>66</td>
<td>2,492</td>
</tr>
</tbody>
</table>

parsing performance is basically unchanged. The accuracy of the function tags is fairly good (95.8 on syntactic tags and 84.6 on semantic (Gabbard et al., 2006)), so it is reasonable to rely on them as input features.

### 3.3 Training

Each of the classifiers was trained with MALLET (McCallum, 2009) using the maximum entropy method on sections 2-21 of the WSJ portion of the Penn Treebank. Section 24 was used for development testing while choosing the feature set and other aspects of the system, and section 23 was used for the final evaluation.

### 3.4 Results

#### 3.4.1 Gold-standard data

For the sake of easy comparison, we first report our results using Johnson’s metric, which is the most widely-used metric for performance on this task. On gold standard

---

6 Two of the classifiers were instead trained using perceptron for historical reasons related to the evolution of the system. The difference does not appear to have any effect on performance. The PRO antecedent model was trained on only sections 10-18 due to memory constraints at the time of the system’s development.

7 For a discussion of metrics for the null element task, see appendix A.
<table>
<thead>
<tr>
<th>Category</th>
<th>Pres</th>
<th>LM</th>
<th>J</th>
<th>DD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comb. 0</td>
<td>87.8</td>
<td>87.0</td>
<td>77.1</td>
<td></td>
</tr>
<tr>
<td>COMP-SBAR</td>
<td>91.9</td>
<td>88.0</td>
<td>85.5</td>
<td></td>
</tr>
<tr>
<td>COMP-WHNP</td>
<td>61.5</td>
<td>47.0</td>
<td>48.8</td>
<td></td>
</tr>
<tr>
<td>COMP-WHADVP</td>
<td>69.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP *</td>
<td>69.1</td>
<td>61.1</td>
<td>55.6</td>
<td>70.3</td>
</tr>
<tr>
<td>Comb. wh-trace</td>
<td>78.2</td>
<td>63.3</td>
<td>75.2</td>
<td>75.3</td>
</tr>
<tr>
<td>NP <em>T</em></td>
<td>80.9</td>
<td>80.0</td>
<td></td>
<td>82.0</td>
</tr>
<tr>
<td>ADVP <em>T</em></td>
<td>69.8</td>
<td>56</td>
<td>53.6</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.3: F1 scores comparing our system to the two PSLB post-processing systems and Dienes and Dubey’s integrated system on automatically parsed trees from section 23 using Johnson’s metric.

Trees from section 23 (table 3.2), our system’s performance compares favorably with other post-processing systems (that of Levy and Manning and that of Johnson). Most notably, it has the best performance of any post-processing system on the two most numerous categories, (NP *)s and wh-traces, which together account for 83% of the instances of the null elements under consideration. Compared to the other approach it is very similar to with respect to learning technique (Levy and Manning), it reduces error on these categories by 49% and 69%, respectively.

### 3.4.2 Automatically Parsed Data

F1 scores on automatically parsed sentences from section 23 are given in table 3.3. Note that our system’s parsed scores were obtained using the modified version of Bikel’s implementation of Collins’s thesis parser mentioned above, while the other post-processing systems use Charniak’s parser (Charniak, 2000), which has higher overall parsing performance, and Dienes and Dubey integrate null element recovery directly into a variant of Collins’s parser. On these automatically parsed trees, our...

---

8For both this table and the next, Levy and Manning report only aggregate results for wh-traces and do not distinguish 0s, (WHNP 0)s and (WHADVP 0)s; Johnson’s aggregate scores are taken from their paper.
<table>
<thead>
<tr>
<th>System</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>D&amp;D</td>
<td>78.50</td>
<td>68.08</td>
<td>72.92</td>
</tr>
<tr>
<td>Pres</td>
<td>74.70</td>
<td>74.62</td>
<td>74.66</td>
</tr>
</tbody>
</table>

Table 3.4: Comparison of our system with that of Dienes and Dubey on automatically parsed data from section 23 over the aggregation of all categories in table 3.3 excepting the infrequent (WHADVP 0)s, which they do not report but which we almost certainly outperform them on.

<table>
<thead>
<tr>
<th>Category</th>
<th>Present</th>
<th>Campbell</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP *</td>
<td>88.8</td>
<td>86.9</td>
</tr>
<tr>
<td>NP <em>T</em></td>
<td>96.3</td>
<td>96.0</td>
</tr>
<tr>
<td>ADVP <em>T</em></td>
<td>82.2</td>
<td>79.9</td>
</tr>
<tr>
<td>0</td>
<td>99.8</td>
<td>98.5</td>
</tr>
</tbody>
</table>

Table 3.5: A comparison of the present system with Campbell’s rule-based system on gold-standard trees from section 23 using Campbell’s metric.

system outperforms other post-processing systems. On the most numerous category by far, (NP *), our system reduces the error of the best learning-based post-processing approach by 21%. Comparing our aggregate wh-trace results to the others, we reduce error by 41% over Levy and Manning and by 12% over Johnson. We also slightly improve over the best result on 0s, reducing error by 6% compared to Levy and Manning.

Performance on automatically parsed data compared to the integrated system of Dienes and Dubey is split. We reduce error by 25% and 44% on plain 0s and (WHNP 0)s, respectively and by 12% on wh-traces. We increase error by 4% on (NP *)s. Aggregating over all the categories under consideration, the more balanced precision and recall of our system puts it ahead of Dienes and Dubey’s, with a 6.4% decrease in error (table 3.4).
### Table 3.6: A few of the most important features for various classifiers.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Features with largest weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPTrace</td>
<td>daughter categories, function tags, argumentness, heads, and POS tags, subjectless S...</td>
</tr>
<tr>
<td>NULLComp</td>
<td>is first daughter?, terminalness, aunt’s label and POS tag, mother’s head, daughters’ heads, great-grandmother’s label...</td>
</tr>
<tr>
<td>WHXPIINSERT</td>
<td>is first daughter?, left sister’s terminalness, labels of mother, aunt, and left sister, aunt’s head...</td>
</tr>
<tr>
<td>WHXPDISCERN</td>
<td>words contained by grandmother, grandmother’s head, aunt’s head, grandmother’s function tags, aunt’s label, aunt’s function tags...</td>
</tr>
<tr>
<td>WHTrace</td>
<td>lack of subject, daughter categories, child argument information, subjacency violation, saturation, whether or not there is a verb below, path information...</td>
</tr>
<tr>
<td>PROANTECEDENT</td>
<td>antecedent’s sisters’ function tags, categories path contains, path length, path shape, antecedent’s function tags, antecedent’s sisters’ heads, linear precedence information...</td>
</tr>
<tr>
<td>ANTECEDENTLESS</td>
<td>mother’s function tags, great-grandmother’s label, aunt’s head (e.g. “It is difficult to...”), grandmother’s function tag, mother’s head...</td>
</tr>
</tbody>
</table>

3.4.3 Comparison to Campbell

On gold-standard trees,\(^9\) our system out-performs Campbell’s rule-based system on all four categories, reducing error by 87% on 0s,\(^10\) by 11% on \((\text{ADVP} \ast \text{T}\ast)\)s, by 7% on \((\text{NP} \ast \text{T}\ast)\)s, and by 8% on the extremely numerous \((\text{NP} \ast)\)s.

3.5 Discussion

We have shown that a post-processing approach can outperform the integrated approach of Dienes and Dubey (2003b). Given that their modifications to Collins’s parser caused a decrease in local phrase structure parsing accuracy, our approach

---

\(^9\)Only aggregate statistics over a different set of null elements were available for Campbell on automatically parsed data, making a comparison impossible.

\(^{10}\)Note that for comparison with Campbell, the 0 numbers here exclude \((\text{WHNP} \ 0)\)s and \((\text{WHADVP} \ 0)\)s.
is therefore particularly appealing. We have further shown that our approach, using only simple, unconjoined features, outperforms Campbell’s (Campbell, 2004) state-of-the-art, complex system on gold-standard data, suggesting that much of the power of his system lies in his richer linguistic representation and his structuring of decisions rather than the hand-designed rules.

We have also compared our system to that of Levy and Manning, which is based on a similar learning technique, and have shown large increases in performance on all of the most common types of null elements; this increase seems to have come almost entirely from an enrichment of the linguistic representation and a slightly different structuring of the problem, rather than any use of more powerful machine-learning techniques.

We speculate that the primary source of our performance increase is the enrichment of the linguistic representation with function tags and argument markings from the parser’s first stage, as table 3.6 attests. We also note that several classifiers make use of the properties of aunt nodes, which have previously been exploited only in a limited form in Johnson’s patterns. For example, ANTECEDENTLESS uses the aunt’s head word to learn an entire class of uncontrolled PRO constructions like “It is difficult (NP *) to imagine living on Mars.”

However, there remain a few areas for improvement. First, since decisions are made in a pipeline of separate, unconnected stages, in some cases there can be problems with “cascading” errors. Second, conjunction is not handled well. Third, parser errors sometimes make the correct insertion of null elements difficult or impossible. The next chapter will discuss some steps for ameliorating the latter problem, while the first two will be addressed in chapter 5.
Chapter 4

A Joint Model for the Task in Arabic

In this chapter we will look at the null element problem in Arabic. It is to some extent inaccurate to call it “the null element problem” since, as we will show, the nature and focus of the task shifts significantly due to the change in language. Some of these differences, together with the shortcomings of our original model noted in section 3.5, motivate us to create a new joint-inference model for the null element task.

4.1 Null Elements in Arabic

The Arabic training section\(^1\) of the Penn Arabic Treebank (Maamouri et al., 2004) contains 51,068 null elements, which is about the same as the number in the English Penn Treebank despite the ATB’s smaller size. These primarily fall into four classes:

- Null complementizers, which are essentially the same as those in English (see section 1.1.2), although denoted by *0* instead of 0.

---

\(^1\)See section 4.3 for a discussion of our training/test split.
Table 4.1: The relative distribution of the most common null elements in Arabic and English.

<table>
<thead>
<tr>
<th>Type</th>
<th>Antecedent</th>
<th>Primary Use in Arabic</th>
<th>Arabic</th>
<th>English</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP</td>
<td><em>T</em></td>
<td>Nom. Rel. Clause</td>
<td>30%</td>
<td>17%</td>
</tr>
<tr>
<td>NP</td>
<td>*</td>
<td>Pro-Drop</td>
<td>24%</td>
<td>19%</td>
</tr>
<tr>
<td>NP</td>
<td><em>T</em></td>
<td>Topicalization</td>
<td>17%</td>
<td></td>
</tr>
<tr>
<td>WHNP</td>
<td><em>O</em></td>
<td>Null wh-word</td>
<td>14%</td>
<td>3.5%</td>
</tr>
<tr>
<td>NP</td>
<td>*</td>
<td>Pro-drop, Passive</td>
<td>12%</td>
<td>36%</td>
</tr>
<tr>
<td>ADVP</td>
<td><em>T</em></td>
<td>Adv. Rel. Clause</td>
<td>1.3%</td>
<td>5%</td>
</tr>
<tr>
<td>NP</td>
<td>*</td>
<td>Free relative antecedents</td>
<td>0.5%</td>
<td></td>
</tr>
</tbody>
</table>

- (NP *)s are used for several purposes in the ATB. Some of these, such as passivization, are common to English (see section 1.1.3). Others are not, such as indicating subject pro-drop.

- *wh*-traces (represented by (NP *T*) with, typically, a WHNP antecedent, which could itself be a *O*) are used to represent traces in questions and relative clauses, just as in English (see section 1.1.4).

- (NP *T*) with other antecedents (typically noun phrases) is used to indicate topicalization (e.g., “(NP-1 The Settlers of Catan), Dianna cannot stand (NP *T*).”). In particular, Arabic can have both VSO and SVO word order, and all instances of SVO are annotated as the subject topicalizing and leaving a trace below the VP.

The distribution of null elements in Arabic differs from English significantly. (NP *) accounts for over half (54%) of the null elements in English, but only a third (36%) in Arabic. Discovering the antecedents of (NP *) is a very difficult problem in English, so it is very convenient that, while the ratio of (NP *) with antecedents to those without is around two to one in English, the opposite is true of Arabic.

*Wh*-traces play a greater role in Arabic, with a bit less than double the relative frequency in Arabic as in English (30% compared to 17%). Many of these are in relative clauses with null nominal *wh*-words, which are vastly more common in Arabic.
Fortunately, however, null adverbial complementizers are extremely rare in Arabic, which prevents the need to worry about determining the type of null \textit{wh}-words, which, as noted in section 7.2, is difficult in English.

Finally, topicalization patterns are very different between the two languages. In English, nominal topicalization is very rare, but topicalization of \texttt{S} and \texttt{SBAR} (almost always in reporting direct or indirect speech) is quite common. In Arabic, the opposite is true.

### 4.1.1 Linguistic Facts Regarding Relative Clauses

Arabic verbal relative clauses can be divided into two classes depending on whether the noun they are modifying is definite or indefinite.\footnote{This paragraph is derived from the grammatical descriptions of Ryding (2005) and Hamdallah and Tushyeh (1998).} In the case of indefinite relative clauses, a trace in verbal or prepositional object position must be indicated by the presence of a marker called a resumptive pronoun, while such a pronoun is forbidden in subject position. In the treebank, this is indicated by adjoining the trace to the resumptive pronoun to form \((NP (NP (PRP h)) (NP *T*))\).

In definite relative clauses, resumptive pronouns may optionally occur in direct object positions and obligatorily in other non-subject positions. Examining the distribution of nominal traces in relative clauses in Arabic in table 4.2, we see that the obligatory resumptive pronoun cases account for about a quarter of all \textit{wh}-traces.

### 4.2 Previous Work

Unsurprisingly, there has not been as much work on null elements in other languages as in English. To our knowledge, the only such instance within the phrase structure paradigm outside Arabic is the application by Levy and Manning (2004) of their
Table 4.2: The distribution of nominal *wh*-traces by type of relative clause in the Arabic Treebank training section (each cell indicates a percentage of all relative clauses, definite and indefinite). The obligatory resumptive pronoun cases are bolded.

As the work described in this chapter was being concluded, another empty category system for Arabic was published (Bakr et al., 2009). This system, like the finite-state tagger of Dienes and Dubey (2003b), inserts null elements into the surface string. This system tags each word as with a tag indicating the null element which follows it or the tag NO, indicating that there is no following null element, and then train support-vector-machine-based tagger which has access to the reduced part-of-speech tag (see section 4.3) and chunk parsing information for features (the exact feature set they generate from this information is not discussed). This system will be discussed further in section 4.9.2.

### 4.3 Data Set

We use divide the contents of the ATB into training, development, and test sections according to the “Johns Hopkins Workshop Split.” As is common in much Arabic parsing work at the moment, we use gold part-of-speech tags and the unvocalized forms of words. We reduce the POS tags using the mapping supplied with the ATB.

---

3. There has been some work with stronger linguistic formalisms like LFG, HPSG, etc. in other languages, e.g. Guo et al. (2007).

release. Our automatically parsed data is obtained from a version of the Bikel parser with a few modifications by Seth Kulick.

4.4 Model: Motivation

The variation in word-order in Arabic motivates the creation of a new model. As previously mentioned, the subject of an Arabic sentence can either precede or follow the verb; the VSO word order is taken to be the basic order and SVO is analyzed as the subject having been moved by topicalization to somewhere earlier in the sentence (typically but by no means always immediately before the verb).

Now consider the case in which the verb is followed by a single noun phrase. How should this noun phrase be interpreted? In English, it would clearly be an object, since it follows the verb. In Arabic, it could be either the subject or an object, since both may follow the verb. If another noun phrase immediately precedes the verb, it is likely that this noun phrase is the topicalized subject and the post-verbal noun phrase is the direct object, solving the ambiguity. However, if there is no noun phrase immediately preceding the verb, the ambiguity remains, since the subject could well have been topicalized further up in the sentence. Given this ambiguity, decisions concerning the subjects and objects of Arabic verbs must, in some cases at least, be made jointly.

Our original system for English (chapter 3) consisted of a pipeline of maximum entropy classifiers; the new model we apply to Arabic essentially ‘wires together’ all of these classifiers to make their decisions jointly, making a conditional random field (Lafferty et al., 2001) (although one with a more general graph structure than the familiar linear–chain CRFs which generalize Hidden Markov Models). This will allow us to efficiently search the entire space of possible null element assignments.
4.5 Model: ‘Declarative’ Description

We will first describe the structure of the model in general terms and then give the graph building algorithm in detail. To construct the CRF we need two things: a set of variables and a set of factors. The variables represent the decisions to be made: in this case, things like “is the object of this verb non-existent, an (NP *), or a wh-trace?” or “where should the wh-trace associated with this wh-word be placed?” The factors are are functions which take the values of these variables (or some relevant subset of these variables) as input and produce features which are used to score possible assignments to the variables.

Our model has two different types of variables:

- **Slot variables** represent unfilled positions in the tree which could potentially be filled by something. We have two types of slot variables:
  
  - Nominal slot variables represent unfilled positions in the tree which could potentially be filled by noun phrases, e.g. arguments of verbs, objects of prepositions, etc. The possible values of a nominal slot variable represent those things (e.g. null elements, displaced noun phrases, nothing) which could fill these positions.
  
  - Adverbial slot variables represent places there could possibly be an adverbial wh-trace.

- Every wh-word has a wh-variable associated with it representing the decision of where to put the corresponding trace. Its possible values are the slots it could fill, as defined by the appropriate kind of slot variable.

The model has the following types of factors:

- Between each wh-variable and each of its values (which are slot variables), we create a path factor. This will be used to enforce that the placement of the wh-trace is legal and to contribute to the score information about the relative
location of a trace and its antecedent. See section 4.5.1 for a more detailed explanation of this.

- For every slot variable, we create a slot factor which produces features based on that slot variable alone. This scores possible ways of filling a slot based on the local context (e.g. how many noun phrases the verb associated with the slot dominates, etc.).

- Between the two nominal slot variables of a verb, we create a subcategorization factor including both of these variables in order to model how ‘happy’ the verb is with the proposed way of filling its subcategorization frame (e.g. a known intransitive verb assigned two noun phrases as arguments will result in features indicating this problem).

An example of a graph for a portion of a sentence (the one in figure 4.1) is given in figures 4.3 and 4.4 (key in figure 4.2).

4.5.1 Wh-trace placement in detail

Since the placement of *wh*-traces is the most potentially confusing part of the model, we will describe this aspect in detail here. The variables relate to *wh*-trace placement as follows:

- There is a slot variable associated with every empty argument position of each verb. Among the other values of these slot variables are all the *wh*-variables which represent *wh*-words which could have been extracted from that argument position.

- Associated with each *wh*-word is a *wh*-variable whose values are the slot variables the *wh*-word this variable represents could have been extracted from.
Several factors relate to $wh$-trace placement, but we will focus on the the $wh$-factors which are created between each $wh$-variable and the slot variables representing possible extraction sites.

These play a particularly important role. Some mechanism is needed to prevent nonsensical variable assignments. In particular, while each $wh$-variable by construction can choose only one slot variable as its value, any number of slot variables may choose that $wh$-variable as their value. This is normally uninterpretable. We could avoid this case explicitly in inference, but it is much simpler to indicate it with a feature which will then receive a large negative weight. In particular, the $wh$-factor between a $wh$-variable $w$ and a slot variable $s$ adds a feature $WHMismatch$ if $w$’s value is $s$ but $s$’s is not $w$, or vice-versa.

Because every $wh$-variable must have a value, there is at least one slot variable associated with each $wh$-variable. Because of the $WHMismatch$ features, there is at most one such association. Therefore, as desired, we have a unique slot for each $wh$-trace.

4.6 Model: ‘Procedural’ Description

This section describes in detail how the graph is constructed from an input tree. Readers interested in only the general structure of the system should feel free to proceed to the next section.

The formal algorithm for creating a graph from an Arabic Treebank tree is as
Figure 4.2: Key for explanatory figures for graph creation. Boxes represent variables and circles represent factors.

Figure 4.3: A slot factor is added for each slot variable, and a subcategorization factor is added joining all slots of the same verb. In each box, the name of the variable is in bold at the top and its possible values are listed in italics below it. A value of \textit{null} indicates that a slot is left empty.
Figure 4.4: Wh-path factors are added between the wh-variable and each slot variable. In each box, the name of the variable is in bold at the top and its possible values are listed in italics below it. A value of null indicates that a slot is left empty.

Figure 4.5: An example of a trace with a resumptive pronoun within a -PRD.
follows:

- At each node $t$ of the tree, insert a null complementizer ($0$) below it if:
  - $t$ is an SBAR.
  - $t$ does not immediately dominate a WHNP, a WHADJP, a WHADVP, a WHPP or an IN.
  - $t$ does immediately dominate an S or a FRAG.

The 0 should be marked as a WHNP if $t$ is not the child the of PP.

- The tree is processed by pre-order traversal depth-first search. At each node $t$:
  - If $t$ is a VP whose immediate head is a verb or a modal which lacks a verbal sister,
    * Create two sets $sbjvals$ and $objvals$ to represent the possible values of the verb’s subject and object slot variables, respectively. Add to each $null$ (indicating that the slot should be empty) and $(NP *)$.
    * Add candidate topicalization traces to $sbjvals$ by adding any c-commanding NP which is marked as an argument and has at most one SBAR intervening between it and the verb.
    * Search recursively from parent to parent up the tree from $t$. Whenever you come across an S, SBAR, or SBARQ which dominates a WHNP, add that $wh$-node to $sbjvals$ and $objvals$. As you go along, also note any WHADVPs you encounter in the set $whadvps$, which will contain the possible values of the adverbial slot variable associated with the verb. Break off the search after you have seen two SBARs.
    * Create a slot variable for the subject slot with possible values $sbjvals$.
    * Create a slot variable for the object slot with possible values $objvals$.
    * if the set $whadvps$ is non-empty, create an adverbial slot variable with possible values $whadvps$. 

53
otherwise, if $t$ bears the function tag \texttt{-PRD}, $^5$ $t$ has no overt, non-resumptive-pronoun sister nodes, and $t$ is not an argument of a VP, create a set $vals$ and add (NP $^*$) and $null$ to it. Search for and add $wh$-word possibilities as done above for verbs. Create a new predicative slot variable with possible values $vals$.

Sometimes the \texttt{-PRD} can itself contain a trace attached to a resumptive pronoun (as in figure 4.5). Handling this is slightly complicated: if $t$ is a \texttt{-PRD} but has an overt, non-resumptive-pronoun sister node, we check if the \texttt{-PRD} constituent itself is a resumptive pronoun or contains one in an immediately dominated PP or ADJP. If this is the case, we create the \texttt{-PRD} variable as above. During inference, if a trace is assigned to this variable, it is inserted as attached to the \texttt{-PRD} internal resumptive pronoun.

- if $t$ is a prepositional phrase with no overt, non-resumptive pronoun object and no \texttt{SBAR} intervenes between the \texttt{PP} and the closest \texttt{WHNP}, create a set $vals$ and add $null$ and (NP $^*$) to it. Gather and add $wh$-word values as above. Create a new prepositional slot variable with possible values $vals$.

- Do a depth-first search by pre-order traversal over the tree. Whenever you encounter a \texttt{WHNP} or \texttt{WHADVP} node, attempt to create a $wh$-variable. The procedure for creating a $wh$-variable from a $wh$-node $t$ is as follows:

  - for each slot variable, if the $wh$-word c-commands it with at most one intervening \texttt{SBAR} node, then

    * If the $wh$-variable is not adverbial, add the slot variable as a possible value for it.

    * If the $wh$-variable is adverbial or of unknown type and the slot variable is adverbial, add the slot variable as a possible value for it.

$^5$That is, a predicative phrase, such as an NP or ADJP which would typically be use together with a form of the copula be in English.
• If no possible values are found, abort creation of the \textit{wh}-variable.

• For each slot variable, create a slot factor of the correct type (verbal, predicative, prepositional).

• For each \textit{wh}-variable, for each of its values, create a \textit{wh}-factor for the pair.

• For every verb with associated slot variables, create a subcategorization factor containing all those variables.

4.7 Features

In this section, we will describe the features generated by each type of factor.

• The features added by slot factors are:

  – if the slot is associated with a verb, we add features indicating how many \textit{NP} arguments it has, whether it has a subject, its part-of-speech tag, the presence of \textit{VP} and \textit{S BAR} complements, and features concerning the presence of resumptive pronouns. For values concerning topicalization both path features and features concerning the displaced \textit{NP} are used: whether it is a resumptive pronoun, a child of a \textit{VP} under an \textit{S BAR}, or a child of a \textit{VP} under another \textit{VP}, as well as features concerning the path between the trace and antecedent. Values concerning \textit{wh}-movement add path features as well.

  – if the slot is associated with a \textit{–PRD}, it simply has a feature noting this.

  – if the slot is associated with a preposition, it has features noting this, and in the case of \textit{wh}-trace assignments, it has features noting whether the object of the preposition is a resumptive pronoun.

• The features added by \textit{wh}-path factors are:
– a set of features constraining the graph to legal assignments regarding *wh*-placement. See section 5.1.3 for a discussion of this in the context of English.

– The following features, conjoined with the type of the *wh*-word (nominal or adverbial):

* the sequence of non-terminal symbols along the path between the trace and its antecedent. Sequences of multiple NPs are collapsed and non-terminals present due to conjunction are ignored.

* the number of of categories on the path (any length greater than eight is treated as eight).

* whether a PRN is present on the path.

* whether a FRAG is present on the path.

• The features added by subcategorization factors are:

– a feature which is the conjunction of the number of arguments the verb is being given with the class of the verb, where the class is obtained by five-way bucketing of all the verbs in the training set according to how many arguments they are observed to take.

4.8 Training and Inference

Once constructed, inference was done using the junction tree algorithm (as implemented by Graphical Models in Mallet (Sutton, 2006)). Learning was done using the maximum entropy technique with L2 regularization as implemented by Mallet (McCallum, 2009). At runtime, translation from an assignment to the variables back trees is largely straightforward. The only subtlety concerns resumptive pronouns. These are annotated in the treebank as being grouped in a noun phrase with their
Table 4.3: Function tagging confusion matrix and dev-test accuracy on the ATB dev-test section. TPC indicates topicalized constituents and PRD indicates predicative constituents. Arg collapses together -SBJ and -OBJ. NotArg consists primarily of adjunct tags like -TMP. Data from Seth Kulick

Several of the features above make reference to the presence of function tags like -PRD and to concepts dependent upon function tags, like argumentness. The Bikel parser for Arabic has generally been used with some of these function tags kept on non-terminals (so that, for example, NP-SBJ is treated as an atomic non-terminal). For these experiments we used output from a version of the parser modified by Seth Kulick (in a manner similar to Gabbard et al. (2006)) to include other additional tags like -TPC as well. The accuracy of the parser on the function tags is given in table 4.3.

### 4.9 Results

#### 4.9.1 System Performance

In our evaluation, we use the typed dependency metric discussed in section A.3.1. The coarsest way to measure this task is to evaluate the parser for all dependencies, including those related to null elements. This results in the parser’s dependency score dropping from 85.7 (not counting null elements) to 81.6 (counting null elements) due to a sharp drop in recall (table 4.4). Evaluating the output again after using this system to restore null elements improves the score to 83.8, trading a small decrease
<table>
<thead>
<tr>
<th>System</th>
<th>P</th>
<th>R</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ignoring NEs</td>
<td>85.7</td>
<td>85.8</td>
<td>85.7</td>
</tr>
<tr>
<td>Parser</td>
<td>85.7</td>
<td>77.9</td>
<td>81.6</td>
</tr>
<tr>
<td>Pres</td>
<td>84.2</td>
<td><strong>83.4</strong></td>
<td><strong>83.8</strong></td>
</tr>
</tbody>
</table>

Table 4.4: Overall dependency evaluation of the parser output (Parser) and the parser output with null elements restored (Pres). The parser output is also provided ignoring null element dependencies for comparison.

in precision for a large increase in recall.

We can get a more detailed look at system performance by breaking dependencies down by type (table 4.5). Performance on *0* and WHNP *0* are very high, which is not surprising since they are largely deterministically derivable from the parse tree itself. Performance on (NP *T*) (an aggregate of topicalization and wh-movement) is fairly good, though lagging somewhat behind English. Most troublesome, however, is (NP *). As in English, there is a significant drop-off (107% increase in error) between the accuracy of simply finding (NP *)s and resolving their antecedents as well. However, scores in Arabic are depressed due to greater difficulty in the initial placement of (NP *). This is in part not terribly surprising, since Campbell (2004) showed that on gold standard parses many cases of (NP *) can be restored in English by a simple rule with extremely high accuracy, in large part because passives are easy to spot by their part-of-speech tag. In the standard tagset mapping we used, at least, this was not the case in Arabic: there is not POS tag which uniformly indicates the presence of an (NP *). It is possible however that using the full POS tags provided by a diacritization system could help here (Habash and Rambow, 2005).

Moving to automatically parsed data, it is unsurprising that we see a considerable drop in performance. The sharpest dropoff comes for *0*s, WHNP *0*s, and (NP *T*)s. This largely reflects difficulties by the parser in determining clause structure. Especially troublesome is the sharp dropoff in (WHNP *0*), which is in turn responsible for most of the drop off on (NP *T*). This occurs in English as well (although for somewhat different reasons), but it has a much greater impact in Arabic since so
Table 4.5: Performance on gold standard and automatic parses by the typed dependency metric. “(na)” indicates ignoring antecedents.

<table>
<thead>
<tr>
<th>Type</th>
<th>Gold F1</th>
<th>Parsed F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHNP <em>0</em></td>
<td>98.3</td>
<td>73.1</td>
</tr>
<tr>
<td>NP *</td>
<td>??</td>
<td>??</td>
</tr>
<tr>
<td>NP * (na)</td>
<td>83.0</td>
<td>74.7</td>
</tr>
<tr>
<td>ADVP <em>T</em></td>
<td>64.4</td>
<td>54.2</td>
</tr>
<tr>
<td>NP <em>T</em></td>
<td>86.8</td>
<td>70.4</td>
</tr>
<tr>
<td>NP <em>T</em> (na)</td>
<td>87.4</td>
<td>76.9</td>
</tr>
<tr>
<td><em>0</em></td>
<td>96.3</td>
<td>80.9</td>
</tr>
</tbody>
</table>

Table 4.6: Arabic null element performance for the system of Bakr et al. (2009) (on a different test set, with gold-standard chunk parsing). NO indicates their accuracy at determining a word is followed by no null element. Table from their paper.

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
<td>0.991</td>
<td>0.994</td>
<td>0.993</td>
<td>15138</td>
</tr>
<tr>
<td><em>T</em></td>
<td>0.881</td>
<td>0.855</td>
<td>0.868</td>
<td>614</td>
</tr>
<tr>
<td><em>0</em></td>
<td>0.830</td>
<td>0.672</td>
<td>0.743</td>
<td>131</td>
</tr>
</tbody>
</table>

many more of its nominal relative clauses have null complementizers (half as opposed to about a fifth). Inspection of errors on the development test set reveals consistent confusion (in both directions) between sentences with topicalized subjects and nouns modified by relative clauses with null complementizers. Close attention to this sort of error seems to be called for, and any technique for fixing it could yield significant improvement for both null element recovery and overall Arabic parsing performance.

4.9.2 Difficulties in Comparison to Bakr et al. (2009)

Comparison of our system to that of Bakr et al. (2009) is difficult (their results are given in table 4.6) for several reasons:

- they report only results assuming gold-standard chunk parsing.
- they do not report results for (NP *).
• their technique cannot account for cases where multiple null elements follow a single word, which occurs for 5.7% of all words followed by null elements. Their evaluation does not count these missing multiple elements.\(^6\)

• They do not attempt to find antecedents for null elements.

• Their training/test split differs from ours.\(^7\)

Although we cannot make a rigorous comparison, we can ‘impressionistically’ observe that our results for \(*0*\) look much higher (upper 90s F-measure for us compared to 74.3 for them) while our \(*T*\) results are at least about the same, without taking into account that our evaluation is much more stringent.

### 4.10 Conclusion

We have presented the first system for null element recovery with antecedents on Arabic (and on any non-Germanic language). It presents reasonably good performance on gold-standard data but is hobbled on automatically parsed data by the poor overall parsing performance in Arabic. Perhaps more interestingly, the particular structure of Arabic, especially its varying word-order, motivated the creation of a new joint inference model for the null element task. In the next chapter, we will revisit English using this model.

For future work, there is a need to investigate ways to improve the construction of relative clauses with null complementizers, either inside the parser or by allowing the post-processor to modify the parse trees it gets as input, rather than simply adding to them. There is also room for improvement by making better use of morphological information.

---

\(^6\)One of the authors, in personal communication, notes that they tried a variant where they had special labels for the double null element cases, but obtained poor results (Bakr, 2009). He notes that in one case, 0 followed by \(*T*\), they do use a special tag which is then converted to simply \(*T*\) in the output.

\(^7\)It would be possible to rerun our experiment using their training/test split, but since the other factors render meaningful comparison impossible, it does not seem worthwhile.
Chapter 5

English Revisited

Having created a joint model for Arabic, we now apply it to English to see if it produces any improvement. First we will discuss three possible areas of improvement over the original model. We will then present the joint model for English and discuss its performance.

5.1 Changes in the Model for English

5.1.1 Slot Competition

It sometimes happens that a given slot could plausibly be filled by more than one type of null element. Most commonly this competition is between an (NP *) and a (NP *T*), as in figures 5.1 and 5.2.

However, the original model for English dealt with such competition poorly. Separate models are trained for each null element and these models are applied in an ordered pipeline. If a mistake is made by a classifier early in the pipeline, the later classifiers will not have the opportunity to correctly insert their own elements. Approximately a third of the false positive NP *T*s ought to be NP *, and about 4% of the false positive NP *s ought to be NP *T*s.
Figure 5.1: The gold standard analysis for a case (from section 24) where the original model erroneously assigns NP \( *T* \) where NP * should be.

Figure 5.2: The erroneous original system analysis which assigns NP \( *T* \) where NP * should be. The system thought this was a case like that in figure 5.3.
Figure 5.3: In this case from the training corpus, the sort of analysis the system did in figure 5.2 is correct.

5.1.2 \textit{Wh}-type variables

A similar situation holds with null \textit{wh}-words. In the original system, when a relative clause has no overt complementizer, the WHXPI\textsc{N}ERT classifier inserts a 0. Then a second classifier, WHXP\textsc{D}ISCERN is consulted to determine whether this 0 should be marked as nominal ((\textsc{WHNP} 0)) or adverbial ((\textsc{WHADVP} 0)).

Later on, \textit{wh}-trace insertion is triggered by the presence of \textsc{WHNP}s and \textsc{WHADVP}s. Both the type of \textit{wh}-trace to be inserted and the model used to insert it are chosen based on the type of the \textit{wh}-word, so errors by WHXP\textsc{D}ISCERN can be another source of cascading errors. While most of the decision about the correct type of a null \textit{wh}-word is determined by the preceding word, in unclear cases it can be useful to use information from, for example, the slot factors (e.g. if none of the verbs in the clause appear to be missing arguments, an adverbial analysis is more likely).

Since null complementizers can be effectively restored in a simple way, incorporating the them into the joint model would be quite complicated for little benefit, so we simply insert them by the following rules:

- for a non-terminal node $t$, if $t$ is an \textsc{SBAR}, $t$ dominates an \textsc{S}, and $t$ dominates neither a \textsc{WHNP}, \textsc{WHADVP}, \textsc{WHADVP}, \textsc{WHPP}, nor a word with part-of-speech tag IN,
insert null complementizer below it.

- If a null complementizer was inserted by the above rule, mark it as a \((\text{WH} \ 0)\) if any of the following hold:
  
  - \(t\)’s parent is an \(\text{NP}\) and it dominates a single \(\text{S}\) child.
  
  - \(t\) has children, then first of which is \textit{for} and the second of which is an \(\text{S}\).

The \(\text{WH}\) category we assign above is of course not a proper Penn Treebank non-terminal label; rather, it serves as a signal to the main stage of the system that a null complementizer of unknown type is present. When constructing the factor graph, a \textit{null wh-type factor} is generated for every \textit{wh}-variable attached to a \((\text{WH} \ 0)\) appearing in the tree. This factor will add the following features:

- the proposed \textit{wh-type}

- the conjunction of the proposed \textit{wh-type} and the stemmed\(^1\) head of the preceding constituent.

### 5.1.3 Conjunction

The original model had no direct knowledge of conjunction. This was most problematic for \textit{wh}-traces because the correct placement was decided by scoring the possible placements and choosing the best without provision for the possibility that it may be best to choose more than one option. The new system models conjunction primarily as it applies to this case.

Recall that in our discussion of \textit{wh}-trace placement in the joint model in section 4.5.1 we showed that each \textit{wh}-word is associated with a single unique placement of its associated \textit{wh}-trace. While this is usually desirable, in the case of conjunction, this is precisely what we do not want.

---

\(^1\)We used the Porter Stemmer. (Porter, 1980; Keyes, 1998)
To deal with this, we revise the WHMismatch rule. First, we must define formally what we mean by being “in conjunction with.” Two slot variables $s_1$ and $s_2$ are considered to be in conjunction with one another if all the following conditions hold:

- the slot of both matches (that is, both are subjects or both are objects).
- the paths between each slot node and the closest common ancestor of the nodes in the potential conjunction do not contain an SBAR.
- among the children of their closest common ancestor is a conjunction, a comma, or a parenthetical which dominates one of the slot nodes.

We now revise the WHMismatch rule for a $wh$-variable $w$ and a slot variable $s$ as follows:

- if neither $w$ nor $s$ has the other as a value, do nothing.
- if $w$ has $s$ for its value and $s$ has $w$ for its value, do nothing.
- if $w$ has $s$ for its value, but $s$ does not have $w$ for its value, add WHMismatch.
- if $s$ has $w$ for its value, but $w$ has some other slot variable $t$ for its value, then
  - if $s$ and $t$ are not in conjunction, add WHMismatch.
  - if $s$ and $t$ are in conjunction, then
    * if $t$ is to the left of $s$, add WHConjToTheLeft.
    * if $t$ is to the right of $s$, add WHConjToTheRight.

The purpose of the WHConjToTheLeft and WHConjToTheRight features is to enforce a canonical form for $wh$-traces in conjunctions, namely, that the $wh$-variable should always point to the leftmost trace. We do this because our training procedure requires a particular single correct answer, so we must choose between the multiple graph representations which represent the same syntactic reality.
Figure 5.4: Key for explanatory figures for graph creation. Boxes represent variables and circles represent factors.

5.2 Model: ‘Declarative’ Description

The model is essentially the same as the one used for Arabic, with the following notable differences:

- The English model lacks the subcategorization factors because English’s SVO word order (nearly) always makes it clear whether a noun phrase occupies a subject or an object argument slot.

- The English model includes the *wh*-type variables discussed in section 5.1.2.

A graphical example for a simple sentence can be found in Figures 5.5, 5.6 and 5.7 (key in 5.4).

5.3 Model: ‘Procedural’ Description

The formal algorithm for creating a graph from a Penn Treebank tree is as follows:

- At each node $t$ of the tree, insert a null complementizer (0) below it if:
That is the book I tried to sell

That is the book (WH 0) I tried to sell

(WH 0) I tried [tried/ADV, tried/OBJ] to sell [sell/OBJ, sell/ADV]

Figure 5.5: First, 0s are added by rule, then all open argument and adjunct slots are found.

- t is an SBAR.
- t does not immediately dominate a WHNP, a WHADJP, a WHADVP, a WHPP or an IN.
- t does immediately dominate an S.

Mark the 0 as a WH if either of the following hold:

- t’s parent is an NP, and t has a single S child.
- t has two children, the first of which is for and the second of which is an S.

- The tree is processed by pre-order traversal depth-first search. At each node t:

  - If t is a VP whose immediate head is a verb or a modal which lacks a verbal sister,
    * examine each child node of t.
  * Create a set vals which will be the set of values for the slot variable when it is created. Add to it null, representing that the slot is filled by nothing, and (NP *).
Figure 5.6: A slot factor is created for each slot variable, and a \textit{wh}-type factor is created including the \textit{wh}-variable and the \textit{wh}-type variable. The variables for the subject slot of \textit{sell} and adverbial slot for \textit{tried} have been omitted to reduce clutter. In each box, the name of the variable is in bold at the top and its possible values are listed in italics below. A value of \textit{null} indicates that a slot is left empty.
Figure 5.7: Between each *wh*-variable and each slot variable a *wh*-path factor is created. The variables for the subject slot of *sell* and adverbial slot for *tried* have been omitted to reduce clutter.

In each box, the name of the variable is in bold at the top and its possible values are listed in italics below. A value of *null* indicates that a slot is left empty.
* Search recursively from parent to parent up the tree from $t$. Whenever an S, SBAR, or SBARQ which dominates a WH\(^2\) or a WHNP is encountered, add that $wh$-node to $vals$. Note any WHADVPs seen in a set $whadvps$, which will be the possible values of the adverbial slot variable. Break off the search after two SBARs have been seen.

* If $t$ has no VP argument, create slot variables as follows:
  - If the verb lacks a subject, create a slot variable for the subject slot with possible values $vals$.
  - If the verb has no object, create a slot variable for the object slot with possible values $vals$.
  - If the verb has one overt object, create a slot variable for the second object slot with possible values $vals$.

* If the set $whadvps$ is non-empty, create an adverbial slot variable with possible values $whadvps$.

  – otherwise, if $t$ bears the function tag -PRD (that is, is is a predicative phrase) and $t$ has no sister nodes, create a variable $vals$ and add (NP *) and $null$ to it. Search for and add $wh$-word possibilities as done above for verbs. Create a new predicative slot variable with possible values $vals$.

  – if $t$ is a prepositional phrase with no object, create a set $vals$ and add $null$ and (NP *) to it. Gather and add $wh$-word values as above. Create a new prepositional slot variable with possible values $vals$.

• Do a depth-first search by pre-order traversal over the tree. Whenever an WHNP or WHADVP node is encountered, attempt to create a $wh$-variable. The procedure for creating a $wh$-variable from a $wh$-node $t$ is as follows:

  – for each slot variable, if the $wh$-word $c$-commands it with at most one intervening SBAR node, then

\(^2\)See section 5.1.2.
* If the *wh*-variable is not adverbial, add the slot variable as a possible value for it.

* If the *wh*-variable is adverbial or of unknown type and the slot variable is adverbial, add the slot variable as a possible value for it.

– If no possible values are found, abort creation of the *wh*-variable.

• For each slot variable, create a slot factor of the correct type (verbal, predicative, prepositional).

• For each *wh*-variable, for each of its values, create a *wh*-factor for the pair.

• For each *wh*-variable of undetermined type (that is, whose *wh*-variable is of type *WH*), add a *wh*-type factor.

### 5.4 (NP *) antecedent model

The very common null element (NP *) has a wide variety of uses. Some of them, such as control constructions and passivization, are entirely syntactic in nature. In other cases, however, it has a more pronomial character.

Concerning the coindexation of (NP *), the treebank guidelines (Bies et al., 1995) state that:

(NP *) bears a reference index whenever it is fairly clear what nominal it is controlled by, corresponding roughly to controlled PRO and the passive trace. However, indexing also reflects pragmatic coreference in addition to syntactic relations...

For the most part, with a few exceptions noted in the guidelines, (NP *) in non-syntactically-controlled cases are coindexed with whatever NP in the sentence the annotator takes to be coreferent on the basis of pragmatic judgement.
One could therefore plausibly ask whether the antecedents of \((\text{NP } *)\) in these cases should properly be resolved in the context of a more general coreference resolution system. While this would be possible, there is no strong motivation for it. First, the coindexed antecedent will always be within the sentence, making the document-level scope of a coreference resolution system unnecessary. Second, the coreference behavior of \((\text{NP } *)\) is not identical to an ordinary pronoun (e.g. local cues can indicate the lack of coindexation), so special handling is necessary for them regardless of what system handles them.

Therefore it seems to us there is no compelling reason not to simply handle their coindexation as part of our system rather than a coreference resolution system. However, given their partially non–syntactic nature, we use an additional separate post-processing step for this. This post-processing system works as follows:

- \((\text{NP } *)\)s in reduced relative clauses are never coindexed.
- Other \((\text{NP } *)\)s which do not bear a \textit{-SBJ} function tag are coindexed by searching up the tree to the first \texttt{NP}, \texttt{S}, or \texttt{SQ} encountered. If it has a \textit{-SBJ} child, coindex with that child. Otherwise, if it has an \texttt{NP} child immediately preceding a \texttt{VP}, coindex with that child. Otherwise, do not coindex.
- If the \((\text{NP } *)\) bears a \textit{-SBJ} function tag, each \texttt{NP} node which c-commands it is a candidate for coindexation, as well as a special virtual ‘null’ node indicating no co-indexation (similar to the approach of Levy and Manning (2004); see section 2.4). These candidates are scored according to features described below and the highest scoring option is used to determine the coindexation of the \((\text{NP } *)\).

The features used for scoring candidate \texttt{NPs} for coindexation in this last case are:

- the sequence of non-terminal symbols on the path between the \((\text{NP } *)\) and the proposed node.
• If at any point along the path there is an S argument of a VP and a constituent intervenes between the verb and the S, its non-terminal or terminal symbol is added as a feature.

• The head word and part-of-speech tag of the proposed antecedent. Head words not seen more than once during training are mapped to a special token \texttt{UNKNOWN}.

• For every non-terminal symbol seen along the path, a feature indicating this is added (e.g. \texttt{SAWNP}, \texttt{SAWVP}, etc.).

• The number of nodes along the path (capped at eight)

The features used to score the null case are:

• A feature \texttt{NULL} which indicates this is a null case

• The head word and part-of-speech tag of what the (NP *) depends on (typically a verb). If the head word is \textit{to} another version of this feature is added which uses the infinitive as the head word.

• Whether the (NP *) lies within an ADJP-PRD (e.g. Figure 5.8). If so, the ADJP-PRD’s headword is added as a feature.

5.5 Training and Inference

Training and inference were performed identically to the Arabic model (see section 4.8). The training set was sections 2-21 of the Penn Treebank WSJ corpus, and the development and test sets were sections 24 and 23, respectively. Function tags were obtained from Seth Kulick’s modified version of the Collins-Bikel parser (see section 3.2.1).
Figure 5.8: A case of an (NP *) where the head word of the nearby ADJP-PRD indicates there is no coindexation.

<table>
<thead>
<tr>
<th>Type</th>
<th>Gold</th>
<th>Parsed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Old</td>
<td>New</td>
</tr>
<tr>
<td>NP <em>T</em></td>
<td>92.8</td>
<td>96.5</td>
</tr>
<tr>
<td>ADVP <em>T</em></td>
<td>79.2</td>
<td>86.7</td>
</tr>
<tr>
<td>NP *</td>
<td>78.6</td>
<td>82.7</td>
</tr>
<tr>
<td>NP *(na)</td>
<td>95.8</td>
<td>96.6</td>
</tr>
<tr>
<td>WHNP 0</td>
<td>92.0</td>
<td>90.2</td>
</tr>
<tr>
<td>WHADVP 0</td>
<td>71.0</td>
<td>66.7</td>
</tr>
</tbody>
</table>

Table 5.1: Gold standard and automatically-parsed test set results (F-measure) for the new and old English models by the typed-dependency metric. (na) indicates ignoring antecedents.

5.6 Results

Results for the new system are evaluated according to the typed-dependency metric (see section A.3.1). Results are summarized in table 5.1.

On gold standard data, error on nominal wh-traces is reduced by a bit over half and error on adverbial traces decreases by over a third. Surprisingly, discernment of the types and locations of relative clauses with null wh-words decreases slightly. 

**precision recall tradeoff.** Error is reduced by 15% on the placement of (NP *) and by 20% on placement combined with coindexation.
On automatically-parsed data, the improvement is more modest. Most significantly, error drops on nominal traces by 14% and on adverbial traces by 9%. On the other hand, performance actually drops slightly for the placement and coindexation of (NP *) (a bit over a 2% increase in error).

5.7 Comparison to Shen (2006)

As promised in section 2.5.4, in this section we will make a comparison, to the degree possible, with the LTAG-Spinal parser of Shen (2006).

5.7.1 Difficulties in Comparing Results

However, making this comparison is not easy. The LTAG-Spinal parser exists in two versions, a left-to-right incremental parser and a bidirectional incremental parser (hereafter ‘binc’). Since the latter has better parsing performance, we will compare to it. We immediately run into our first problem for comparison, however: binc’s output does not include the full information necessary to reconstruct the derived tree, but rather includes only unlabeled dependencies, making it impossible to tell if a null element is being attached in subject, object, or adverbial position. Subject and object confusion is rarely a problem (though it sometimes can be, as in figure 5.9), but properly making the distinction between nominal and adverbial traces for relative clauses with null wh-words is both important and difficult. Since that information is not present, in our evaluation we suppress these distinctions for our system as well.

Apart from this, evaluating relative clauses with overt wh-words is fairly straightforward since both systems will place the wh-words as dependencies on the verb of the relative clause in the same way. For consistent comparison, we modify our system to follow binc’s practice of treating the the first conjunct of a conjunction as the head. We also modify our head-finding rules to make auxiliary verbs depend on their main verbs.
Figure 5.9: A sample output tree from binc (from the dev-test section). For each word, the other words that depend on it are indicated by entries prefixed with att. Notice that since the dependencies are unlabelled, we do not know whether the relative clause indicated by the dependency of do on lot is nominal or adverbial, or if nominal, which argument slot it occupies.
<table>
<thead>
<tr>
<th>System</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pres</td>
<td>83.6</td>
<td>84.8</td>
<td>84.2</td>
</tr>
<tr>
<td>LTAG/Binc</td>
<td>84.0</td>
<td>82.0</td>
<td>83.0</td>
</tr>
</tbody>
</table>

Table 5.2: Performance on wh-traces with overt *wh*-words (nominal and adverbial together) compared to binc.

Evaluating relative clauses without overt *wh*-words is trickier. Relative clauses are not the only thing binc marks with the dependency of a verb on a noun, and since there is no overt *wh*-word present, the relevant cases cannot be picked out for comparison. We therefore evaluate binc’s performance by finding each occurrence of a nominal or adverbial *wh*-trace with no overt *wh*-word in the gold standard and examining binc’s analysis of the sentence by hand. This, of course, only allows us to compare on the basis of recall and not of precision.

5.7.2 Results

Performance of our system compared to binc is presented in tables 5.2 and 5.3 for *wh*-traces with overt and hidden *wh*-words, respectively. On overt cases, we lag binc very slightly on precision but gain considerably on recall, so our overall performance has 7% less error (measured by F-measure).

On covert *wh*-traces the gap is wider, with our error being 20% lower. This appears to be largely due to the Collins-Bikel parser being better at detecting the presence of relative clauses with null *wh*-words, which binc seems more likely to attempt to analyze in other ways.
Table 5.3: Performance on wh-traces with empty *wh*-words (nominal and adverbial combined) compared to binc. “LTAG Multiplaced” refers to cases which are difficult to classify as correct or incorrect because binc places the trace in two places, one of which is right and one of which is wrong.

<table>
<thead>
<tr>
<th>System</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pres</td>
<td>65.2</td>
</tr>
<tr>
<td>LTAG/Binc</td>
<td>56.5</td>
</tr>
<tr>
<td>(LTAG Multiplaced)</td>
<td>1.5</td>
</tr>
</tbody>
</table>
Chapter 6

System Analysis

6.1 Error Analysis

For future work, it is useful to know what the remaining sources of error are. Therefore in this section we will examine all\(^1\) mistakes made by the core system of the previous chapter on automatic parses of the development test section.

6.1.1 Nominal null \textit{wh}-words

The development test section contained 29 sentences involving errors with respect to nominal null complementizers. Excepting the determination of whether an instance is nominal or adverbial, placement of this category is largely deterministic given the parser output. It is therefore unsurprising that most of the system errors are a direct consequence of parser errors.

Eighteen of the twenty-nine errors result from confusion involving infinitival relatives. Either another sort of clause was misanalyzed as a relative clause (ten cases for purpose clauses, one for verbal complements, and four for nominal and adjectival complements), or, in the other direction, a relative clause was misanalyzed as a purpose clause (one case) or nominal/adjectival complement (three cases). Clearly

\(^1\)For the very numerous (NP *)s, we sampled instead.
the mislabeling is not symmetrical; the parser tends to favor the creation of relative clauses. These cases will be discussed in more detail in section 7.3.

For the remaining cases, the largest cause (six cases) was when the parser made larger scale errors which made recovery impossible (for example, figure 6.1; this includes one null parse). There were two more cases of misparsed infinitival relatives that did not fit into the above categories. Once the system mislabeled a WHADVP as a WHNP (“a series of steps to soften big stock drops”), and once the parser inserted a relative clause, but the system inserted no null complementizer (“a much easier standard for a state to satisfy”; the system tends to avoid inserting null complementizers in SBARs headed by for).

6.1.2 Adverbial null wh-words

On the development test set the system makes ten erroneous predictions concerning (WHADVP 0). In two of these cases, the gold standard was wrong and the system was right (see figure 6.1.2). Of the remaining eight true errors, one was the WHNP/WHADVP
Figure 6.2: Chart showing distribution of errors for nominal null *wh*-words
Figure 6.3: An example where the parser analysis (above) appears superior to the analysis of the gold standard (below). The gold-standard analysis would imply that the calls were instrumental in stripping the stock markets.

confusion mentioned in the previous section. In six cases the parser introduced a relative clause where none was present in the gold standard, and in one case the opposite occurred. Almost every time the parser erroneously inserted a relative clause, it was triggered by the presence of a word like time, day, or way which is frequently modified by relative clauses in the corpus (see figure 6.1.2). Many of these cases were in expressions like from time to time and in time for.

6.1.3 Nominal wh-traces

There are sixty-three errors related to nominal wh-traces in the development test set. Twenty-nine of these errors are a direct consequence of the nominal null wh-word errors which have already been discussed. Of the remaining errors, for one the
Figure 6.4: Chart showing distribution of errors for adverbial null *wh*-words

```
(VP (VBP flare)
   (ADVP (RB up)
      (PP (IN from)
         (NP (NP (NN time))
            (SBAR (WHADVP-0 (-NONE- 0)))
            (S (NP-SBJ (-NONE- *))
               (VP (TO to)
                  (VP (NN time) (, ,)
                    (VP-A (VB depress))...)
```

Figure 6.5: A case of the parser erroneously inserting a relative clause due to the presence of *time*. Note that the tendency to place a relative clause after time is so strong it even outweighs the cost of using a rare $VP \rightarrow NN\ VP$ rule.
Figure 6.6: A case in which the trace placement is correct, but the metric counts it wrong because the parser was mistaken about the parent symbol. The parser/system output is above and the gold standard is below.

Of the true errors, by far the leading cause is the parser either erroneously marking SBARs as relative clauses (eight cases) or failing to do so (ten cases). The false negatives fall into two main classes. The largest are those relative clauses headed by that where the parser marked that as IN rather than WHNP (seven cases, e.g. figure 6.7). The second class are those cases where either the WHNP is complex and therefore misparsed (figure 6.8), or the WHNP is parsed as something complex when in fact it is simple (figure 6.9). The false positives show a similar split, with five of them being due to marking that as a WHNP when it should have been IN and the other three being due to miscellaneous causes (e.g. 6.11).

The next most common type of error (four cases) involves when verbs take S complements (especially S complements whose immediate heads are NP-PRDs) in the gold standard which are missed in the parser output, leading to incorrect placement of the trace (see figure 6.12). Rounding out the list are three cases of difficulties
Figure 6.7: A case where the parser erroneously analyzes that as IN rather than WHNP.

Figure 6.8: A case where the parser fails to properly analyze a complex WHNP.

Figure 6.9: A case where a simple WHNP is incorrectly analyzed as if it were a more complex WHPP.
Despite the relatively strong economy, junk bond prices did nothing except go down.

Figure 6.10: One of the few “miscellaneous” errors.

The parser erroneously creates a relative clause where none should be. The system output is above and the gold standard analysis is below.

Figure 6.11: Here the parser erroneously creates a relative clause where none should be. The system output is above and the gold standard analysis is below.
6.1.4 Adverbial Trace (ADVP *T*)

There are forty-three errors related to adverbial traces in the development test set. In nine of these cases, the system is right and the gold standard is wrong (e.g. figure 6.17) and in seven more the system output is at least defensible (e.g. figure 6.18.) In two cases the system output is basically correct but evaluated as incorrect (see figure 6.19 for an explanation). Four more cases are due to large scale parser errors and two are obscured by the presence of topicalization and gapping (which are beyond
( (SBARQ (WHNP-1 (WP What) (NN real-estate) (NN strategy))
  (SQ (NP-SBJ (-NONE- *T*-1))
    (VP (MD should))
    (NP-TMP (NN one))
    (VP-A-A (VB follow))
    (PP-LOC (IN in))...)

( (SBARQ (WHNP-2 (WDT What) (NN real-estate) (NN strategy)))
  (SQ (MD should))
  (NP-SBJ (NN one))
  (VP (VB follow))
  (NP (-NONE- *T*-2))
  (PP-LOC (IN in)))

Figure 6.13: Here the parser pulls *should* and *one* down into a *VP*. The system has such a strong inclination against allowing subjectless VPs that it incorrectly places the trace. The parser/system analysis is above and the gold standard is below.

( (SINV (S-TPC-0 (NP-SBJ ('` '`) (PRP They)))
  (VP (VBP have))
  (NP (NP (DT a) (NN lot)))
  (SBAR (WHNP-2 (-NONE- 0))
    (S (NP-SBJ-1 (-NONE- *T*-2)))
    (VP (TO to))
    (VP (VB do))
    (NP-A (DT these) (NNS days))
    (S-PRP (NP-SBJ (-NONE- *-1)))
    (VP (TO to))
    (VP (VB compete))

Figure 6.14: Here the parser marks *these days* as an argument when it should be an adjunct with a -TMP function tag. Therefore the system sees *do*’s subcategorization frame as filled and falls back to placing the trace in subject position.

88
Figure 6.15: This is the only remaining case of \((\text{NP } *\text{T}*)/(\text{NP } *)\) confusion in the development test section.

the scope of the system for English), respectively.

Of the remaining classes of errors, only two occur more than twice. The most common problem (seven cases) concerns spurious or omitted relative clauses with empty \(wh\)-words; these are very similar to the analogous cases for nominal \(wh\)-traces. The other problem (five cases) is when our system attempts to place a trace for an overt \(wh\)-word which the gold standard leaves uncoindexed (figures 6.20 and 6.21).

The remaining problems are due to not placing a trace at all for a \(wh\)-word (twice; both appear to be due to programming bugs), a conjunction error by the parser (once), trace placement on only one branch of a conjunction (once), and one case with an overt \(wh\)-word where the parser failed to create a relative clause.

### 6.2 Feature Ablation

In this section, we briefly examine the effect of removing feature classes (table 6.1) on the performance of the core null element system. We do this in two ways.

First, we begin with a minimal base system and add feature classes one by one, in order roughly from the least complicated to the most (table 6.2). As expected, the relatively complicated (either in complexity or size of the feature class) feature classes (\textsc{Context} and below on the table) contributed quite a bit, with the exception of \textsc{Paths}. This is especially surprising because other null element systems such as
Figure 6.16: Chart showing distribution of errors for nominal $wh$-traces
Figure 6.17: A case where the gold standard analysis (shown) is wrong and the system output (not shown) is correct.
Filimonov and Harper (2007) are based around such paths. We suspect that the hard linguistic constraints on paths built into the base system combined with the \texttt{PathLength} and \texttt{SeenOnPath} features encode most of the information present in the full paths and also generalize better to unseen data.

A few feature classes seem to provide no value, at least when added in this order. These include \texttt{1stLevelInf}, \texttt{ParCat}, and \texttt{SbjObjFuncTags}. The last may seem surprising given the value of function tags to the original version of the null element system, but function tags were already been used in building the graph and constraining the space of possibilities in the base system.

A more informative way to look at the features is to examine what happens when each feature class is removed from the full model (table 6.3). From this view we see somewhat different results. All three features which appeared to provide no value before now provide some value, while \texttt{PRDType} and \texttt{GrandParCat} are now somewhat harmful. The exact contributions and interactions of the features remain somewhat opaque.
Figure 6.19: In this case the trace dependencies in the system output (below; gold is above) are correct, but since the parser puts clean and repair in separate VPs, our second trace is counted as incorrect.
Figure 6.20: A case where the gold standard analysis (shown) does not coindex a *wh*-word, but our system does.
Figure 6.21: A more complicated case where the system output (below) has a trace for a \textit{wh}-word which lacks one in the gold-standard (above). Note that the system currently does not handle right-node raising.
Figure 6.22: Chart showing distribution of errors for adverbial *wh*-traces
**SeenOnPath** what non-terminals are on the path between a null element and its antecedent or controller.

**PathLength** the number of non-terminals on the path between a null element and its antecedent or controller.

**PRDType** whether the null element is in a predicative phrase (e.g. NP-PRD, etc.)

**1stLevelInf** whether the null element is the subject of an embedded infinitive

**ParCat** the non-terminal immediately dominating the null element

**GrandParCat** the non-terminal two levels above the null element.

**SubjObjFuncTags** the presence of the -SBJ and -OBJ function tags in the vicinity of the null element

**AuxVBN** whether the null element is in the context of an auxiliary verb followed by a VBN.

**SComp** whether the verb the null element is an object of has a clausal complement.

**Context** describes the context of the null element up to the dominating VP above the one the element is attached to.

** Conj** features controlling conjunction

**PrevHd** the head of the noun phrase preceding a relative clause.

**Paths** the sequence of non-terminals between a null element and its antecedent.

**Lex** lexicalization of verbs and their subcategorization information.

Table 6.1: Descriptions of the feature classes
Table 6.2: This table shows how, beginning from a minimal base system, the performance (F-measure) increases as feature classes are added in an order (roughly) from least complex to most complex.

<table>
<thead>
<tr>
<th>Features Used</th>
<th>WHNP 0</th>
<th>WHADVP 0</th>
<th>NP <em>T</em></th>
<th>ADVP <em>T</em></th>
<th>NP *</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE</td>
<td>51.3</td>
<td>13.8</td>
<td>68.5</td>
<td>46.7</td>
<td>68.2</td>
</tr>
<tr>
<td>+ SeenOnPath</td>
<td>70.6</td>
<td>44.3</td>
<td>68.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ PathLength</td>
<td>76.3</td>
<td>54.7</td>
<td>70.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ PRDType</td>
<td>77.3</td>
<td>54.4</td>
<td>70.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ 1stLevelInf</td>
<td>76.4</td>
<td>54.3</td>
<td>70.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ ParCat</td>
<td>75.7</td>
<td>53.7</td>
<td>70.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ GrandParCat</td>
<td>75.9</td>
<td>56.1</td>
<td>70.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ SbjObjFuncTags</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>52.4</td>
</tr>
<tr>
<td>+ AuxVBN</td>
<td>75.6</td>
<td>54.4</td>
<td>70.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ SComp</td>
<td>76.2</td>
<td>54.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ Context</td>
<td>78.4</td>
<td>63.2</td>
<td>85.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ Conj</td>
<td>52.5</td>
<td>19.4</td>
<td>78.9</td>
<td>64.0</td>
<td>85.7</td>
</tr>
<tr>
<td>+ PrevHd</td>
<td>65.2</td>
<td>81.5</td>
<td>82.0</td>
<td>71.3</td>
<td>86.2</td>
</tr>
<tr>
<td>+ Paths</td>
<td>65.9</td>
<td>80.0</td>
<td>82.2</td>
<td>70.2</td>
<td>86.3</td>
</tr>
<tr>
<td>+ Lex</td>
<td>67.4</td>
<td>81.5</td>
<td>83.2</td>
<td>71.9</td>
<td>86.4</td>
</tr>
</tbody>
</table>

Table 6.3: This table shows how performance (F-measure) changes if each class of feature is removed from the full system.

<table>
<thead>
<tr>
<th>Feature Class Removed</th>
<th>WHNP 0</th>
<th>WHADVP 0</th>
<th>NP <em>T</em></th>
<th>ADVP <em>T</em></th>
<th>NP *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full System</td>
<td>67.4</td>
<td>81.5</td>
<td>83.2</td>
<td>71.9</td>
<td>86.4</td>
</tr>
<tr>
<td>- SeenOnPath</td>
<td></td>
<td></td>
<td>-0.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- PathLength</td>
<td>-3.5</td>
<td>-8.0</td>
<td>-1.0</td>
<td>-7.6</td>
<td>+0.1</td>
</tr>
<tr>
<td>- PRDType</td>
<td></td>
<td></td>
<td>+0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- 1stLevelInf</td>
<td>-1.5</td>
<td>-1.5</td>
<td>+0.1</td>
<td>-1.0</td>
<td>+0.1</td>
</tr>
<tr>
<td>- ParCat</td>
<td>-3.7</td>
<td>-0.1</td>
<td></td>
<td>-0.2</td>
<td></td>
</tr>
<tr>
<td>- GrandParCat</td>
<td></td>
<td></td>
<td>+0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- SbjObjFuncTags</td>
<td>-2.2</td>
<td>-3.7</td>
<td>-1.6</td>
<td>+0.1</td>
<td></td>
</tr>
<tr>
<td>- AuxVBN</td>
<td></td>
<td></td>
<td>+0.4</td>
<td>-0.1</td>
<td></td>
</tr>
<tr>
<td>- SComp</td>
<td></td>
<td></td>
<td>-2.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Context</td>
<td>-7.8</td>
<td>-20.0</td>
<td>-3.6</td>
<td>-11.5</td>
<td>-1.8</td>
</tr>
<tr>
<td>- Conj</td>
<td>-2.2</td>
<td>-7.4</td>
<td>-1.0</td>
<td>-1.2</td>
<td>-0.2</td>
</tr>
<tr>
<td>- PrevHd</td>
<td>-16.5</td>
<td>-62.8</td>
<td>-3.7</td>
<td>-7.5</td>
<td>-0.7</td>
</tr>
<tr>
<td>- Paths</td>
<td>+0.7</td>
<td>-1.5</td>
<td>+0.1</td>
<td>-1.2</td>
<td></td>
</tr>
<tr>
<td>- Lex</td>
<td>-1.5</td>
<td>-1.5</td>
<td>-1.0</td>
<td>-0.3</td>
<td>-0.1</td>
</tr>
</tbody>
</table>
Chapter 7

Parsing With Google

In this chapter we will examine whether unsupervised learning from large amounts of data can be useful for null element restoration. The data used for this experiment is the Google Web 1T 5-gram Corpus (henceforth Web 1T) (Google, Inc., 2006), which has been shown to be useful in other applications (Lin et al., 2010; Nulty and Costello, 2009).

7.1 The Google Web 1T Corpus

The Web 1T corpus consists of all one through five-grams extracted from roughly a trillion words of text sampled from the World Wide Web.\footnote{1} Automatic identification was used to remove non-English data as much as possible and filtering was applied to try to get rid of non-useful tokens.

For the most part Google’s tokenization procedures follow those of the Penn Treebank, which is very convenient for using it for parsing experiments. One somewhat troublesome exception is that hyphenated words are always separated, which is not the case for the standard release of the Penn Treebank (Bies et al., 1995). Sentence boundaries are treated as words and marked as <S> and </S>.

\footnote{1}{This section is based on the readme.txt included with the corpus.}
(NP (NP the place)
  (SBAR (WHADVP-2 that/where))
  (S (NP-SBJ I)
   (VP put
    (NP the book)
    (ADVP-PUT *T*-2))))

Figure 7.1: An example relative clause.

(NP (NP the place)
  (SBAR (WHADVP-2 0)
   (S (NP-SBJ I)
    (VP put
     (NP the book)
     (ADVP-PUT *T*-2))))

Figure 7.2: An example relative clause with a null wh-word.

Not all tokens are kept. First, any word not occurring at least 200 times is mapped to an unknown word token <UNK>. Then any n-gram not appearing at least forty times is pruned.

Processing this immense corpus (approximately 25 GB compressed) is quite a challenge. For the following experiments, a combination of custom-written code and the Get IT tools (Hawker et al., 2007) were used for data extraction.

7.2 Null wh-word types

Wh-trace performance suffers substantially on automatically parsed data in our systems (see 3.4.2). As previously discussed in section 1.1.4, every relative clause requires a node of type WHNP, WHADVP, WHADJP, or WHPP as a child of the top SBAR (see Figure 7.1), representing the displaced wh-word. However, in many cases this wh-word may be omitted in the sentence, in which case the treebank fills it by a 0 under a non-terminal of type WHNP, etc (Figure 7.2). The presence of these null complementizers is largely implicit in the parser output, and they are inserted by rule (see appendix 5.3).
Table 7.1: Distribution of wh-word type, overt and covert.

<table>
<thead>
<tr>
<th>Type</th>
<th>% Overt</th>
<th>% Covert</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHNP</td>
<td>74.5</td>
<td>75.3</td>
</tr>
<tr>
<td>WHADVP</td>
<td>20.9</td>
<td>24.6</td>
</tr>
<tr>
<td>WHPP</td>
<td>4.0</td>
<td>0.04</td>
</tr>
<tr>
<td>WHADJP</td>
<td>0.6</td>
<td>0.0</td>
</tr>
</tbody>
</table>

However, error analysis reveals that a large portion of wh-trace errors are concentrated in relative clauses with null wh-words, and these errors in turn seem to come from problems with the null wh-words themselves (our original system, for example, has F-measures of 61.5 and 69.0 for null WHNPs and WHADVPs, respectively). There are two possible problems with the null wh-words: either they may have the wrong type assigned, guaranteeing an incorrect trace placement, or the parser may not have created a relative clause structure at all (recall that the placement of 0s is implicit in the parser output). We will examine the first source of error in this section and the second in the next.

Since WHNP and WHADVP are by far the most common null wh-word types (see table 7.2), we restrict our attention to distinguishing them. We attempt this by means of a simple intuition: if we look at the head word of the phrase preceding the null wh-word, that word’s correlation with overt wh-words should give us a clue to the null element’s type.

### 7.2.1 Approach

To decide the type of a particular null wh-word, we look at the headword $w$ of the phrase immediately preceding it. If this word was seen preceding a null wh-word in the training data, whatever labeling it was given there is used. Otherwise, we assign it adverbial type if

$$\frac{\max(f("w\ where"), f("w\ when"))}{\max(f("w\ who"), f("w\ which"))} > \alpha$$

where $f(x)$ indicates the count in Web 1T of the bigram $x$ and $\alpha$ is a constant to
Table 7.2: The counts used in determining the type of the null \textit{wh}-word in “the man 0 I saw”

<table>
<thead>
<tr>
<th>Bigram</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>man who</td>
<td>6,968,548</td>
</tr>
<tr>
<td>man which</td>
<td>93,342</td>
</tr>
<tr>
<td>man when</td>
<td>129,565</td>
</tr>
<tr>
<td>man where</td>
<td>20,812</td>
</tr>
</tbody>
</table>

Table 7.3: The counts used in determining the type of the null \textit{wh}-word in “the time 0 I went”

<table>
<thead>
<tr>
<th>Bigram</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>time who</td>
<td>74,216</td>
</tr>
<tr>
<td>time which</td>
<td>343,509</td>
</tr>
<tr>
<td>time when</td>
<td>6,066,703</td>
</tr>
<tr>
<td>time where</td>
<td>303,480</td>
</tr>
</tbody>
</table>

be discussed below.\(^2\) If adverbial type is not assigned, we assign nominal type. For some examples, see the counts in tables 7.2 and 7.3 for “the man 0 I saw” and “the time I went,” respectively.

To select the value of \(\alpha\), we search for the one which gives the best performance on the training data (sections 2-21); the training data accuracy for varying values of \(\alpha\) can be seen in Figure 7.3. The final value selected is 9.2, although performance varies little over a wide range from around seven to around sixteen. The performance drops sharply when the threshold goes above that range due to several very common adverbial cases being ruled nominal.

The intuition behind this method is that we wish to compare the best evidence seen have for a nominal labeling with the best evidence seen for an adverbial labeling. The threshold \(\alpha\) for the ratio is also not simply one because we have a strong prior belief that any given instance should be nominal until good evidence convinces us otherwise.

Note above that if the previous word was seen in the training data, we respect the

\(^2\)Replacing \(\max\) by \(\sum\) in this equation is possible, though in testing a some cases by hand early in system development, \(\max\) seemed slightly better.
observed labeling. This is important because there are a few common words which are consistently annotated with WHADVP in the treebank which our approach would have difficulty getting right, e.g. way and means. It is possible that expanding the search beyond bigrams to look for cases like “the means by which” would be effective.

7.2.2 Results

The results on the development test set (section 24) are shown in Table 7.4. We see that the Google approach provides a small improvement, but most of the problem remains. Given that the Google approach performed fairly well on the training data, we can attribute the small performance gain to two factors. First, the previous word was seen in the training data 79% of the time, limiting the number of opportunities for the new approach to do better. Second, the fact that an oracle with access to
<table>
<thead>
<tr>
<th>Approach</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominal baseline</td>
<td>40.3</td>
</tr>
<tr>
<td>Original</td>
<td>68.7</td>
</tr>
<tr>
<td>Google</td>
<td>71.6</td>
</tr>
<tr>
<td>Oracle</td>
<td>74.6</td>
</tr>
</tbody>
</table>

Table 7.4: Accuracy of the Google method on *wh*-type prediction.

the gold annotations performs so poorly indicates that the problem is primarily null *wh*-words which are not being inserted at all rather than those being assigned an incorrect type (that is, our problem #2).

### 7.3 Infinitival Relatives

In this section we investigate *wh*-trace placement errors due to the 0 not being inserted at all or due to an erroneous 0 being inserted. Error analysis on the development test data suggests that many of these errors are due to infinitival relative constructions. These are, as the name suggests, relative clauses consisting of infinitives, as in figure 7.4.

In a situation where the parser sees a verb followed by a noun phrase followed by an infinitival S constituent, it has three plausible analyses:

1. To analyze the S as an infinitival relative clause, as in figure 7.4.

2. To analyze the S as a modifier of the verb, as in figure 7.5.

3. To analyze the S as a complement of the noun, as in figure 7.6.

If either of the first two cases were incorrectly parsed as infinitival relatives, an incorrect relative clause would be created, triggering an incorrect insertion of a *wh*-trace. On the other hand, if an infinitival relative were given one of the other analyses, no attempt would be made to insert a *wh*-trace when in fact there should be one.
Figure 7.4: An infinitival relative.

```plaintext
(S
  (NP-SBJ
    (NP (DT The) (NN league) (POS ’s) )
    (NNS promoters) )
  (VP (VBP hope) )
  (SBAR (-NONE- 0)
    (S
      (NP-SBJ-1 (NNS retirees) )
      (VP (MD will) )
      (VP (VB pack) )
      (NP (DT the) (NNS stands) )
      (S-PRP
        (NP-SBJ (-NONE- */1) )
        (VP (TO to) )
        (VP (VB see) )
        (NP (DT the) (NNS seniors) ))))))))

( . . )
```

Figure 7.5: An infinitive acting as an S modifier of a verb. As is typical, it expresses the purpose of the action of the verb pack. (Adapted from the training data)
Figure 7.6: An infinitive acting as the complement of the noun *attempt*. (Adapted from the training data)
The parser’s performance on this task on the standard development test section of the Penn Treebank (section 24) is an F-measure of 53.9. It is apparent that this is a difficult task and performance is significantly below the general accuracy of the parser.

What information does the parser have in making this decision? We will consider this question in the particular case of the Collins parsing model.\textsuperscript{3} Given a phrase of the form \textit{verb NP[n] S[to]} (where \textit{n} is the head of the noun phrase and \textit{to} is the head of the \textit{S}), the following probability distributions are used in making the attachment decision for the \textit{S}:

- The head-generation probability \( P_h(S|SBAR, to) \), which is the probability that an \textit{SBAR} headed by \textit{to} will generate an \textit{S} as its head child, is used in scoring the infinitival relative option, since in this case a unary projection of \textit{S} to \textit{SBAR} is created. This probability is very nearly 1 and has little value for making attachment decisions.

- The head-generation probability \( P_h(NP|NP, n) \) measures how likely an \textit{NP} headed by the noun \textit{n} will have another \textit{NP} as its head, which largely occurs in cases of postmodification. This can be caused by both the causes we are concerned with and several other things, so it is unclear how valuable the information it supplies is.

- The right subcategorization frame probability \( P_{rc}(SC|S, VP, v) \) measures how likely a verb (or more exactly, an \textit{VP} with parent \textit{S} and head \textit{v}) is to take an \textit{S} complement. Taking an \textit{S} complement is represented by a value of \{\textit{NP, S}\} for \textit{SC} while not taking one is represented by a value of \{\textit{NP}\} for \textit{SC}. If a verb has a high probability of taking an \textit{S} complement then attachment of the \textit{S} to the verb is almost always the correct choice. However, since the verb could take

\textsuperscript{3}For a description of it and further information on the probability structures mentioned below, see chapter 7 of Collins (1999).
an S adjunct, an NP subcategorization frame does not rule out attachment to the verb.

- The right attachment probability $P_r(S[to]|S, \text{VP}, v, distance(1), SC)$ scores how likely an attachment of an infinitival S to the verb is. More exactly, it measures how likely such an attachment is to a VP with parent S and head v ($distance(1)$ indicates the attachment is not immediately adjacent to the head). SC will be either empty or S, depending on whether the S is being analyzed as an adjunct or a complement, respectively. This does not provide a great deal of information in the latter case since the likelihood of a verb taking an S complement has already been modelled by the right subcategorization probability discussed above. However, it does provide useful information about whether a verb tends to take S adjuncts.

- The right attachment probability $P_r(sym[to]|\text{NP}, \text{NP}, n, distance(0), \{}\}$ measures how likely an infinitival complement (for $sym = S$) or relative clause (for $sym = \text{SBAR}$) is to modify an NP headed by n. $distance(0)$ indicates that the attachment is immediately adjacent to the head.

It is interesting to compare how the parser models an S as a complement of a verb and of a noun differently. In the verbal case, information about attachment to the verb is split over two probability distributions: $P_{rc}$ measures the probability of attachment as an argument while $P_r$ measures the probability of attachment as an adjunct.\footnote{While $P_r$ of course still provides a score in the argument case, it is almost entirely determined by the subcategorization frame.} In the nominal cases, $P_r$ provides all the information because the parser does not count any modifier of an NP as a complement, which causes $P_{rc}$ to provide no information at all.

We see from the above that the parser has access to all the necessary information for making the attachment decision: the likelihood of the verb taking an S as a complement or adjunct, the likelihood of the noun taking a complement, and the
likelihood of the noun taking an infinitival relative. So why is performance so poor? In part it may be due to an inherent difficulty in the discrimination task, but it could also be due to having relatively few training examples of a phenomenon which depends heavily on lexical information. In the following section we will perform an experiment to shed some light on this question.

7.3.1 Applying Google to the Problem

In this section we will investigate whether the difficulties with the attachment of infinitives can be alleviated to some degree by the extracting statistics from the Web 1T corpus. However, we immediately run into difficulties, since the corpus is derived from raw text and therefore does not annotate certain distinctions. In particular, we cannot tell directly what the attachments are, and in the case of nominal attachments, even if we knew the attachments we wouldn’t know if they were as complements or as relative clauses.

Nonetheless, we put forward the following hypotheses:

- Nouns taking infinitival complements will show a tighter co-occurrence correlation with the infinitive marker *to* than nouns which do not. For a given noun $x$, we will refer to this measure of correlation as $n_x$, which will be precisely defined below.

- Verbs taking infinitival complements will show a tighter co-occurrence correlation with the infinitive marker *to* than verbs which do not. For a given verb $x$, we will refer to this measure of correlation as $v_x$, which will be precisely defined below.

- For any given instance of the discrimination task with verb $x$ and noun $y$, we should be able to accurately predict the attachment as follows:

  - If $n_y$ is high, the noun probably takes an infinitival complement, so predict an attachment to the noun.
If $v_x$ is high, the verb either takes an infinitival complement or frequently has an infinitival adjunct, so predict an attachment to the verb.

If neither $v_x$ nor $n_y$ is especially high, it is unlikely either the noun or the verb take infinitival complements or frequently occur with infinitival adjuncts, so predict an attachment to the noun as a relative clause.

Calculating a correlation value for $n_x$ for a noun $x$ is appears at first entirely straightforward:

$$n_x = \frac{f(“x \text{ to}”)}{f(“x”)},$$

that is, the number of occurrences of $x$ followed by $to$ over the total number of occurrences of $x$. However, $to$ is of course ambiguous between being the infinitive marker and being a preposition. Therefore we revise our definition to:

$$n_x = \frac{f(“x \text{ to} \text{ verb}”)}{f(“x”)},$$

Calculating a correlation value for $v_x$ is more complicated because a noun phrase of arbitrary length intervenes between the verb and the $to$. This would not be a serious problem in a raw text corpus, but we have a window of at most five words because the Web 1T corpus only goes up to five-grams. However, we can get a fair approximation to the correlation we are interested in by considering only cases where the NP is simply an object pronoun. This gives us:

$$v_x = \frac{f(“x \text{ object-pronoun to} \text{ verb}”)}{f(“x \text{ object-pronoun}”)},$$

In all of these we use a newer version of the Web 1T corpus which includes part-of-speech tags Lin et al. (2010).

Plotting values of $n$ and $v$ for all cases of this task in the training data we (figures 7.7, 7.8, 7.9) we see roughly what we expected. However, there is a fairly large number of non-infinitival relative cases which have small values for both nominal and verbal correlation.
Figure 7.7: Plot of all cases of infinitival Ss attaching to verbs. The horizontal axis indicates the measure $n$ of nominal correlation, while the vertical axis indicates the measure $v$ of verbal correlation. In the case of $S$ attaching to the verb, nominal correlation is very weak and verbal correlation is often relatively strong.
Figure 7.8: Plot of all cases of infinitival Ss attaching as noun complements. The horizontal axis indicates the measure $n$ of nominal correlation, while the vertical axis indicates the measure $v$ of verbal correlation. In the case of noun complements, verbal correlation is relatively weak and nominal correlation is relatively strong.
Figure 7.9: Plot of all cases of infinitival Ss attaching to nouns as relative clauses. The horizontal axis indicates the measure $n$ of nominal correlation, while the vertical axis indicates the measure $v$ of verbal correlation. In the case of infinitival relatives, neither nominal nor verbal correlation is particularly strong.
7.3.2 Results

To evaluate the value of the Web-1T-derived information, we attempt to use it in two ways. The first method, which we will call \textsc{threshold}, applies the algorithm sketched in the previous section using thresholds set to optimize performance on WSJ sections 2-21. The second, called \textsc{combined}, is to use a maximum entropy model trained with the following features designed to capture other information available to the parser when making these attachment decisions:

- the nominal ratio, the verbal ratio, and a “distance” feature which is the greater of the two
- the head POS tag of the preceding NP
- whether there is a verb present or not
- whether or not the preceding NP is modified by anything, conjoined with the bucketed length of the NP
- If the verb occurs more than twice, it is added as a feature. If not, a feature is added indicating whether \textsc{vp} or non-\textsc{vp} attachment is more common for that verb in the training data.
- If the noun occurs more than twice, it is added as a feature. If not, a feature is added indicating whether, when the noun is present, an \textsc{np} complement or non-\textsc{np}-complement attachment is more commonly seen in the training data. A second feature is added indicating whether, when the noun is present, an infinitival relative or non-infinitival-relative attachment is more commonly seen in the training data.

For comparison we also include the parser output (\textsc{parser}) as well as a baseline which always chooses attachment as a nominal complement. The results are given in table 7.5. From this we see that the Web 1T information strongly outperforms
Table 7.5: Results on section 23 for the baseline, thresholding, parser, and combined system approaches.

<table>
<thead>
<tr>
<th>Type</th>
<th>Baseline</th>
<th>Threshold</th>
<th>Parser</th>
<th>MaxEnt</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P  R  F</td>
<td>P  R  F</td>
<td>P  R  F</td>
<td>P  R  F</td>
</tr>
<tr>
<td>VP</td>
<td>0  0</td>
<td>79.1 36.5</td>
<td>50.0 68.5</td>
<td>71.2 69.8</td>
</tr>
<tr>
<td>NP</td>
<td>36.3 94.3</td>
<td>67.1 67.1</td>
<td>67.1 67.7</td>
<td>90.0 90.0</td>
</tr>
<tr>
<td>Rel</td>
<td>0  0</td>
<td>29.5 48.1</td>
<td>36.6 68.6</td>
<td>44.4 53.9</td>
</tr>
</tbody>
</table>

Table 7.6: Change in the performance of null element placement when the original pipeline (PARSER+SYSTEM) is augmented with Web 1T information (COMBINED+SYSTEM)

<table>
<thead>
<tr>
<th></th>
<th>PARSER+SYSTEM</th>
<th>COMBINED+SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P  R  F</td>
<td>P  R  F</td>
</tr>
<tr>
<td>WHNP 0</td>
<td>64.8 55.1 59.6</td>
<td>67.0 62.6 64.7</td>
</tr>
<tr>
<td>WHADVP 0</td>
<td>65.6 58.3 61.8</td>
<td>64.7 61.1 62.9</td>
</tr>
<tr>
<td>NP *</td>
<td>70.5 72.8 71.6</td>
<td>70.7 73.1 71.9</td>
</tr>
<tr>
<td>NP <em>T</em></td>
<td>90.1 85.8 87.9</td>
<td>89.3 86.8 88.0</td>
</tr>
<tr>
<td>ADVP <em>T</em></td>
<td>81.5 78.1 79.8</td>
<td>81.01 78.1 79.5</td>
</tr>
</tbody>
</table>

the baseline, indicating that is is provide a significant amount of information. The thresholding lags behind the parser, most likely due to its failure on some common words which the parser is able to memorize. Combining the threshold with other sources of information available to the parser results in the best performance across all three attachment possibilities. Running the null element system on trees modified by the combined method results in some improvement, although quite small (table 7.6).

### 7.4 Conclusions

In this chapter we showed in two ways how using unsupervised information from Google’s Web 1T corpus can produce modest improvements on two tasks related to parsing relative clauses. Although the effects are small, it does indicate that some amount of syntactic information can be extracted from Web 1T. Future investigation
is warranted to see if there are other constructions which could benefit to a greater degree.
Chapter 8

Conclusions and Future Work

8.1 Contributions

The placement of null elements in parse trees is necessary for the full representation of syntactic structure and therefore for the creation of predicate–argument structure. In this work we made several contributions to this problem:

- we introduced a system for the task which combined machine–learning with linguistically–motivated features to achieve state–of–the–art results among broad–coverage post–processing systems.

- we introduced a second system for the task which had better cross–lingual properties. This system
  
  – allowed the implementation of a state–of–the–art (and the first syntax–based) null element system for Arabic.

  – improved performance on \(wh\)–trace placement in English.

  – is one of the first applications of a graphical model to deep syntactic structure.

- We provided the first detailed error analysis of the problem and highlighted
areas for improvement not previously remarked upon, such as infinitival relatives.

- We investigated the application of the Google Web 1T corpus to the task, showing that some degree of information even about subtle linguistic points can be extracted from raw text.

8.2 Future Work

There are a number of possible directions for future work:

- Parses enhanced with null elements could be incorporated into other downstream systems, such as question answering and textual entailment. They could also be useful for doing pre-translation rearrangement for machine translation of certain language pairs (along the lines of Elming (2008); Elming and Habash (2009)). For example, if one language is \textit{wh}-in-situ and the other has overt \textit{wh}-movement, one could be “normalized” to have a syntactic form closer to the other before translation.

- Adapting the system to other, more typologically diverse languages could present interesting challenges.

- It would be interesting to investigate what degree of prior knowledge would be necessary to learn trace placement in an unsupervised way.

- Finding a way to integrate null element restoration into a state-of-the-art lexicalized parser in a way which does not significantly damage its parsing performance would be very valuable. A possible intermediate step would be applying a post-processing system to parser output in the context of a reranker like Charniak and Johnson (2005) or (even better but less easily) Huang (2008).
Appendix A

Evaluation

Settling on a common evaluation technique has proven particularly difficult for this problem, with at least four metrics in use, none of them entirely satisfactory. Each of the three basic methods are based on the usual practice of finding precision (the fraction of predicted null elements which are correct) and recall (the fraction of true null elements which are found) values for each category of empty categories and computing an F–measure \( \frac{2pr}{p+r} \), where \( p \) is precision and \( r \) is recall) from it. Where they differ is on the question of when an empty category should be judged to be correct.

A.1 Johnson’s metric

The first metric, proposed by Johnson (2002), is that for every null element we should record the following facts:

- its type (e.g. (NP *T*) or (WHNP 0))
- its token position
As he notes, this is essentially a degenerate case\(^1\) of what PARSEVAL (Black et al., 1991) does for constituents of parse trees. If the null element has an antecedent, we add the following information:

- for every node co-indexed with the null element, a triple of the form:
  - the type of the node
  - the left token position of the node
  - the right token position of the node

A null element in system output is counted as correct only if it matches a corresponding null element in the gold standard in all of the above respects.

### A.2 Campbell’s metric

Campbell notes that Johnson’s metric is in some cases too generous and in others too strict. It is too strict in that the exact placement of a null element is unimportant for semantic interpretation so long as it remains within the same constituent.\(^2\) On the other hand, he notes that in some cases like Johnson’s metric cannot distinguish semantically significant differences in attachment level. For example, if someone asks you “when did you decide to go to Walsingham (ADV P *T*)?” it matters whether they want to know when you made up your mind to go to Walsingham or when you are actually going there, but simply knowing the token position of the trace at the end of the sentence will not disambiguate whether the trace modifies *decide* or *go* (see figures A.1 and A.2).

As an alternative he proposes replacing the information recorded for matching in Johnson’s metric by the following:

---

\(^1\)PARSEVAL records for every constituent its left and right token positions. For null elements, these are always the same.

\(^2\)In fact, it is not always even clear linguistically what the notion of the token position of a null element means. For example, in “the book you read quickly,” ought the trace go before or after the adverb?
Figure A.1: Under this interpretation, you are being asked when you made the decision that you would be going to Walsingham. Here the null element depends on the verb *decide* and its parent spans from four to eight.

```
(SINV
  (WHADVP-1 (WRB When))
  (VBD did)
  (NP-SBJ-2 (PRP you))
  (VP
    (VBD decide)
    (S
      (NP-SBJ (*NONE- *-2))
      (VP (TO to)
        (VP (VB go)
          (PP (IN to)
            (NP Walsingham))))))
  (ADVP (*NONE- *T*-1))))
```

Figure A.2: Under this interpretation, you are being asked when you are travelling to Walsingham. Here the null element depends on the verb *go* and its parent spans from six to eight.

```
(SINV
  (WHADVP-1 (WRB When))
  (VBD did)
  (NP-SBJ-2 (PRP you))
  (VP
    (VBD decide)
    (S
      (NP-SBJ (*NONE- *-2))
      (VP (TO to)
        (VP (VB go)
          (PP (IN to)
            (NP Walsingham))
          (ADVP (*NONE- *T*-1)))))))
```
• the type of the null element

• its parent node’s type

• its parent node’s left token position

• its parent node’s right token position

The information added concerning antecedents is the same as for Johnson.

On gold–standard input, results by this metric seem to be (at least for Campbell’s system) slightly higher (about 5-10%) than by Johnson’s metric. However, it is significantly lower for parser output (about 25-30%). Since the gold–standard results increase, it is unlikely the lower results come from the elimination of those cases in which Johnson’s case is too generous. Rather, this metric is now much more sensitive to errors in the parser output, which he views as possibly “an unavoidable consequence of using a tree–based evaluation.” It is not clear that this is the case (to this extent, at least): while certainly in some cases the parser output is so wrong that correct recovery is impossible, sometimes simple attachment errors to either the parent node or the antecedent which are irrelevant to the interpretation of the null element are the cause of the metric’s disapproval. Sometimes this is due to punctuation, which is easily ignored, but sometimes it is not, as in figure A.3.

Campbell also proposes that the task should also properly include the assignment of function tags, and he provides results showing a drop in accuracy of about 9% when this requirement for correctness is added.

A.3 Typed–dependency metrics

Levy and Manning (2004) notice the same difficulties with Johnson’s metric with respect to tree position as Campbell does, but they propose an alternative solution more in line with their way of posing the problem (see section 2.4). They do this by taking a parse tree with null elements added and extracting its interpretation
(S
  (NP-SBJ-1 (NP (DT An) (JJ omnibus) (NN bill)))
  (VP (VP (VBN assembled)
    (PP (IN by)
      (NP-LGS (NP (NNP Sen.) (NNP Edward) (NNP Kennedy))
        (PRN (-LRB- -LRB-)
          (NP (NP (NNP D.))
            (, ,)
            (NP (NNP Mass.))
            (-RRB- -RRB-))))
        (, ,)
        (CC and)
        (VP (VBG including)
          (NP (DT some) (NNP Nunn-McCurdy) (NNS provisions))))))
  (ADVP (IN along)
    (PP (IN with)
      (NP (NP (NNS proposals))
        (PP (IN by)
          (NP-LGS (NP (NNP Sen.) (NNP Pell))
            (CC and)
            (NP (NNP Christopher) (NNP Dodd))))))
    (, ,)
    (VP (VBZ has)
      (VP (VBN been)
        (VP (VBN reported)
          (NP (-NONE- *))
          (PP-CLR (IN out)
            (PP (IN of)
              (NP (DT the) (NN committee))))))) (. .) )
)

Figure A.3: In this sentence from the development set, \textit{along with proposals by Sen.s Pell, Barbara Mikulski, and Christopher Dodd} has here been analyzed by the Collins–Bikel parser as an adverb phrase at the top level of the sentence rather than correctly as a prepositional phrase modifying \textit{including}. Because of this, the token span for the noun phrase headed by \textit{bill} which is the antecedent of the null element has been shortened, causing Campbell’s metric to count it wrong. (Figure adapted from the development test data)
as dependency relations. For non–co–indexed parts of the tree, this can be done by straightforward head percolation. For null elements which are co–indexed, the co–indexed material is interpreted at the location of the null element. The system is then evaluated by comparing typed dependency relations, where every relation includes:

- the head word
- the depending word
- the category of the mother node

This metric seems superior to that of Campbell in that it more directly models the needs of the predicate–argument recovery task null element recovery is intended to serve. Most importantly, it fixes the issues with Johnson’s metric, but unlike Campbell’s metric, it doesn’t make the evaluation sensitive to errors in the parser output which are irrelevant to the role of the null elements in determining predicate–argument structure.

### A.3.1 Towards an Ideal Evaluation

Unfortunately, the metric as employed by Levy and Manning has its own problems. It works very well when comparing post–processing systems applied to the same parser output. However, it is difficult to use to compare systems using different parser output because differences in parser accuracy on dependencies unrelated to null elements will obscure differences in null element performance. This is a minor shortcoming for post–processing systems, since optimally you want to use the same parser for both systems anyway.

When comparing to null element systems which are integrated inside parsers, the problem is more serious. For example, by this metric the post–processing system of

---

3Johnson mentions experimenting with a similar metric and finding its results to be approximately the same. However, this could in part be due to Johnson’s pattern–matching being more likely to fail anyway in the case of parser errors.
Levy and Manning (running on the output of the Charniak parser) outperforms the parser–integrated system of Dienes and Dubey, but it is unclear to what degree this is due to the parser–integrated system (which is based on the lower–performance Collins model) having lower performance than Charniak’s parser on dependencies unrelated to null elements.

To some extent this is justifiable since lowered general parsing performance is a possible argument against using parser–integrated approaches (see section 2.2.3). However, it would still be useful to be able to distinguish this side–effect of such approaches from their performance on the null element task considered in isolation. In a recent paper on recovering a particular type of null element, Filimonov and Harper (2007) employ a variant of the typed–dependency metric in which only those dependencies related to null elements are counted. This reduces the potential for differences in the dependency accuracy of the underlying parsers to overwhelm differences in the performance of null element systems, although it does not eliminate it completely. Using this metric together with the overall typed–dependency score should give the fullest picture of null element system performance, so we adopted it for our final evaluations.

A.4 Typed Dependency Metric Implementation

In this section, we will describe exactly how we derive the typed dependence evaluation scores used in chapters 4 and 5. As mentioned above, this is essentially the method of Filimonov and Harper (2007).

The first task is, for a given sentence, to extract all dependencies relevant to null elements. For each of these dependencies, we must determine the head and the modifier. We choose as the head the head word of the constituent the null element is located in (determined using the same head rules as the parser). To find the modifier, we do as follows:
• if the null element is a *wh*-trace, adverbial trace, or \((\text{NP } *)\) with an antecedent, we choose as the modifier the head word of the antecedent or controller. If the head word is itself an empty category with an antecedent or controller, we keep following the coindexation chain until we encounter either an overt word or a null element without an antecedent or controller.

• if the null element has no antecedent or controller, the null element itself is chosen as the modifier.

We represent a dependency by the zero–based token positions of the head and modifier, where the token position of a null element is assigned to be \(-1\). To this we add the type of the null element involved in the dependency. This, however, is not quite sufficient, because it cannot, for example, distinguish between placing a trace in the subject versus the object position of the same verb, since in both cases the modifier token position is not determined by the null element location and the head for both is the verb. We therefore also add as our final piece of information the non–terminal symbol immediately dominating the null element, which will be \(\text{S}\) for null elements in subject position and \(\text{VP}\) for those in object position.

More formally, for each dependency involving a null element we have a four–tuple \((t, p, m, h)\) where:

• \(t\) is the type of the null element

• \(p\) is the non–terminal symbol of the head’s parent node

• \(m\) is the token position of the modifier

• \(h\) is the token position of the head

For an example of the null element dependencies extracted from a tree, see figure A.4.
Figure A.4: The above has dependencies \{(NP *, S, 4, 5), (NP T*, VP, 0, 6)\}. 

\[(SINV (WHNP-1 (WDT What))
  (VBD did)
  (NP-SBJ (PRP you))
  (VP (VB persuade)
    (NP-OBJ-2 (PRP him))
    (S (NP-SBJ (-NONE- *-2))
      (VP (TO to)
        (VP (VB do)
          (NP-OBJ (-NONE- *T*-1)))))\)

Figure A.4: The above has dependencies \{(NP *, S, 4, 5), (NP T*, VP, 0, 6)\}. 

\[(SINV (WHNP-1 (WDT What))
  (VBD did)
  (NP-SBJ (PRP you))
  (VP (VB persuade)
    (NP-OBJ-2 (PRP him))
    (S (NP-SBJ (-NONE- *-2))
      (VP (TO to)
        (VP (VB do)
          (NP-OBJ (-NONE- *T*-1)))))\)
Bibliography


