Recovery of Empty Nodes in Parse Structures

Denis Filimonov[1]
[1]University of Maryland
College Park, MD 20742
den@cs.umd.edu

Mary P. Harper[1,2]
[2]Purdue University
West Lafayette, IN 47907
mharper@casl.umd.edu

Abstract

In this paper, we describe a new algorithm for recovering WH-trace empty nodes. Our approach combines a set of hand-written patterns together with a probabilistic model. Because the patterns heavily utilize regular expressions, the pertinent tree structures are covered using a limited number of patterns. The probabilistic model is essentially a probabilistic context-free grammar (PCFG) approach with the patterns acting as the terminals in production rules. We evaluate the algorithm's performance on gold trees and parser output using three different metrics. Our method compares favorably with state-of-the-art algorithms that recover WH-traces.

1 Introduction

In this paper, we describe a new algorithm for recovering WH-trace empty nodes in gold parse trees in the Penn Treebank and, more importantly, in automatically generated parses. This problem has only been investigated by a handful of researchers and yet it is important for a variety of applications, e.g., mapping parse trees to logical representations and structured representations for language modeling. For example, SuperARV language models (LMs) (Wang and Harper, 2002; Wang et al., 2003), which tightly integrate lexical features and syntactic constraints, have been found to significantly reduce word error in English speech recognition tasks. In order to generate SuperARV LM training, a state-of-the-art parser is used to parse training material and then a rule-based transformer converts the parses to the SuperARV representation. The transformer is quite accurate when operating on treebank parses; however, trees produced by the parser lack one important type of information – gaps, particularly WH-traces, which are important for more accurate extraction of the SuperARVs.

Approaches applied to the problem of empty node recovery fall into three categories. Dienes and Dubey (2003) recover empty nodes as a pre-processing step and pass strings with gaps to their parser. Their performance was comparable to (Johnson, 2002); however, they did not evaluate the impact of the gaps on parser performance. Collins (1999) directly incorporated wh-traces into his Model 3 parser, but he did not evaluate gap insertion accuracy directly. Most of the research belongs to the third category, i.e., post-processing of parser output. Johnson (2002) used corpus-induced patterns to insert gaps into both gold standard trees and parser output. Campbell (2004) developed a set of linguistically motivated hand-written rules for gap insertion. Machine learning methods were employed by (Higgins, 2003; Levy and Manning, 2004; Gabbard et al., 2006).

In this paper, we develop a probabilistic model that uses a set of patterns and tree matching to guide the insertion of WH-traces. We only insert traces of non-null WH-phrases, as they are most relevant for our goals. Our effort differs from the previous approaches in that we have developed an algorithm for the insertion of gaps that combines a small set of expressive patterns with a probabilistic grammar-based model.

Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pp. 620–629, Prague, June 2007.

2 The Model

We have developed a set of tree-matching patterns that are applied to propagate a gap down a path in a parse tree. Pattern examples appear in Figure 1. Each pattern is designed to match a subtree (a root and one or more levels below that root) and used to guide the propagation of the trace into one or more nodes at the terminal level of the pattern (indicated using directed edges). Since tree-matching patterns are applied in a top-down fashion, multiple patterns can match the same subtree and allow alternative ways to propagate a gap. Hence, we have developed a probabilistic model to select among the alternative paths. We have created 24 patterns for WHNP traces, 16 for WHADVP, 18 for WHPP, and 11 for WHADJP.

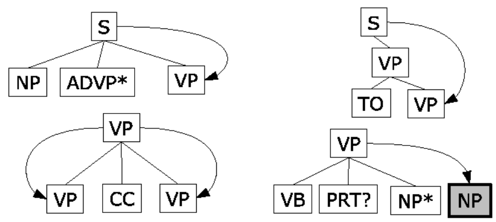

Figure 1: Examples of tree-matching patterns

Before describing our model, we first introduce some notation.

- T_{ij}^N is a tree dominating the string of words between positions i and j with N being the label of the root. We assume there are no unary chains like $N-X-...-Y-N$ (which could be collapsed to a single node N) in the tree, so that T_{ij}^N uniquely describes the subtree.
- A gap location $g_{cd}^{ab,N}$ is represented as a tuple $(gaptype, ancstr(a,b,N), c, d)$, where $gaptype$ is the type of the gap, (e.g., $whnp$ for a WHNP trace), $ancstr(a,b,N)$ is the gap's nearest ancestor, with a and b being its span and N being its label, and c and d indicating where the gap can be inserted. Note that a gap's location is specified precisely when $c = d$. If the gap is yet to be inserted into its final location but will be inserted somewhere inside $ancstr(a,b,N)$, then we set $c = a$ and $d = b$.
- $ancstr(a,b,N)$ in the tuple for $g_{xy}^{ab,N}$ is the tree T_{ab}^N.
- $p(g_{xy}^{ab,N}|gaptype, T_{ij}^N)$ is the probability that a gap of $gaptype$ is located between x and y, with a and b being the span of its ancestor, and $i \le a \le x \le y \le b \le j$.

Given this notation, our model is tasked to identify the best location for the gap in a parse tree among the alternatives, i.e.,

$$\underset{x,a,b,N}{argmax}\ Pr(g_{xx}^{ab,N}|T, gaptype)$$

where $g_{xx}^{ab,N}$ represents a gap location in a tree, and $T = T_{ij}^N$ is the subtree of the parse tree whose root node is the nearest ancestor node dominating the WH-phrase, excluding the WH-node itself, and $gaptype$ is the type of the gap. In order to simplify the notation, we will omit the root labels N in T_{ij}^N and $g_{xy}^{ab,N}$, implying that they match where appropriate.

To guide this model, we utilize tree-matching patterns (see Figure 1), which are formally defined as functions:

$$ptrn : \mathcal{T} \times \mathcal{G} \rightarrow \mathbf{\Gamma} \cup \{none\}$$

where $\mathcal{T}$ is the space of parse trees, $\mathcal{G}$ is the space of gap types, and $\mathbf{\Gamma}$ is the space of gaps g_{cd}^{ab}, and $none$ is a special value representing failure to match[1]. The application of a pattern is defined as: $app(ptrn, \tau, gaptype) = ptrn(\tau, gaptype)$, where $\tau \in \mathcal{T}$ and $gaptype \in \mathcal{G}$. We define application of patterns as follows:

$$app(ptrn, T_{ij}, gaptype) \rightarrow g_{xy}^{ab} : i \le a \le x < y \le b \le j$$
$$app(ptrn, T_{ij}, gaptype) \rightarrow g_{xx}^{ab} : i \le a \le x \le b \le j$$
$$app(ptrn, T_{ij}, gaptype) \rightarrow none$$

Because patterns are uniquely associated with specific gap types, we will omit $gaptype$ to simplify the notation. Application is a function defined for every pair $(ptrn, T_{ij})$ with fixed $gaptype$. Patterns are applied to the root of T_{ij}, not to an arbitrary subtree.

Consider an example of pattern application shown in Figure 2. The tree contains a relative clause such that the WHNP-phrase *that* was moved from some location inside the subtree of its sister node *S*.

$$_2 viewers\ _3 will\ _4 tune\ _5 in\ _6 to\ _7 see\ _8$$

[1]Modeling conjunction requires an alternative definition for patterns: $ptrn : \mathcal{T} \times \mathcal{G} \rightarrow Powerset(\mathbf{\Gamma}) \cup \{none\}$. For the sake of simplicity, we ignore conjunctions in the following discussion, except for in the few places where it matters, since this has little impact on the development of our model.

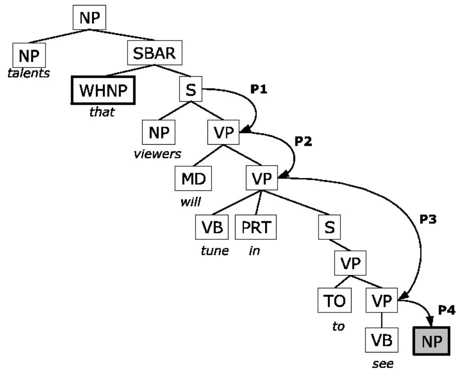

Figure 2: A pattern application example

Now suppose there is a pattern P_1 that matches the tree T_{28} indicating that the gap is somewhere in its subtree T_{38} (*will tune in to see*), i.e., $app(P_1, T_{28}) \rightarrow g_{38}^{38}$. The process of applying patterns continues until the pattern P_4 proposes an exact location for the gap: $app(P_4, T_{78}) = g_{88}^{78}$.

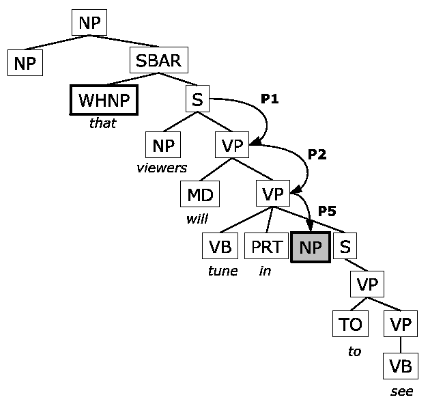

Figure 3: Another pattern application example

Suppose that, in addition to the pattern applications shown in Figure 2, there is one more, namely: $app(P_5, T_{48}) \rightarrow g_{66}^{48}$. The sequence of patterns P_1, P_2, P_5 proposes an alternative grammatically plausible location for the gap, as shown in Figure 3. Notice that the combination of the two sequences produces a tree of patterns, as shown in Figure 4, and this pattern tree covers much of the structure of the T_{28} subtree.

2.1 Tree Classes

The number of unique subtrees that contain WH-phrases is essentially infinite; hence, modeling them directly is infeasible. However, trees with varying details, e.g., optional adverbials, often can be characterized by the same tree of patterns. Hence, we can represent the space of trees by utilizing a relatively small set of classes of trees that are determined by their tree of pattern applications.

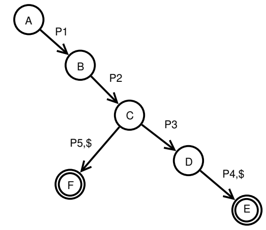

Figure 4: Pattern tree

Let Π be the set of all patterns. We define the set of patterns matching tree T_{ij} as follows:

$$M(T_{ij}) = \{P \mid P \in \Pi \wedge app(P, T_{ij}) \neq none\}$$

To enable recursive application:

$$app(ptrn, g_{xy}^{ab}) = \begin{cases} app(ptrn, T_{ab}) & \text{if } x < y \\ none & \text{if } x = y \end{cases}$$

A *Pattern Chain* PC is a sequence of pairs of patterns and sets of pattern sets, terminated by \$, i.e., $(\frac{p_1}{M_1}, \frac{p_2}{M_2}, ... \frac{p_n}{M_n}, \$)$, where $\forall_i \; p_i \in M_i \subset \Pi$. $M_i = M(T_{ab})$, where T_{ab} is the result of consequent application of the first $i-1$ patterns: $app(p_{i-1}, app(p_{i-2}, ..., app(p_1, T_{\alpha\beta}))) = g_{xy}^{ab}$, and where $T_{\alpha\beta}$ is the subtree we started with, (T_{28} in the example above). We define *the application of a pattern chain* $PC = (\frac{p_1}{M_1}, \frac{p_2}{M_2}, ... \frac{p_n}{M_n}, \$)$ to a tree T_{ij} as:

$$app(PC, T_{ij}) = app(p_n, ...app(p_2, app(p_1, T_{ij})))$$

It is important to also define a function to map a tree to the set of pattern chains applicable to a particular tree. The pseudocode for this function called FindPCs appears in Figure 5[2]. When applied to T_{ij}, this function returns the set of all pattern chains, applications of which would result in concrete gap locations. The algorithm is guaranteed to terminate as long as trees are of finite depth and each pattern moves the gap location down at least one level in the tree at each iteration. Using this function, we define *Tree Class* (TC) of a tree T_{ij} as $TC(T_{ij}) = \text{FindPCs}(T_{ij})$.

[2] $list \circ element$ means "append $element$ to $list$".

```
function FindPCs'(T_ij, PC, allPCs) {
    M_ij ← {P | P ∈ Π ∧ app(P, T_ij) ≠ none}
    forall P ∈ M_ij
        g^ab_xy ← app(P, T_ij)
        PC ← PC ∘ P/M_ij
        if x = y then // g^ab_xy is a concrete location
            allPCs ← allPCs ∪ {PC ∘ $}
        else
            allPCs ← FindPCs'(T_ab, PC, allPCs)
    return allPCs }
function FindPCs(T_ij) { return FindPCs'(T_ij, [ ], ∅) }
```

Figure 5: Pseudocode for FindPCs

In the case of a conjunction, the function FindPCs is slightly more complex. Recall that in this case $app(P, T_{ij})$ produces a set of gaps or *none*. The pseudocode for this case appears in Figure 6.

```
function CrossProd(PC_1, PC_2) {
    prod ← ∅
    forall pc_i ∈ PC_1
        forall pc_j ∈ PC_2 : prod ← prod ∪ {pc_i ∘ pc_j}
    return prod }
function FindPCs(T_ij) {
    M_ij ← {P | P ∈ Π ∧ app(P, T_ij) ≠ none}
    newPCs ← ∅
    forall P ∈ M_ij
        PCs ← {[ ]}
        forall g^ab_xy ∈ app(P, T_ij)
            if x = y then
                forall pc ∈ PCs : pc ← pc ∘ $
            else
                PCs ← CrossProd(PCs, FindPCs(T_ab))
        forall pc ∈ PCs : pc ← P/M_ij ∘ pc
        newPCs ← newPCs ∪ PCs
    return newPCs }
```

The set $app(P, T_{ij})$ must be ordered, so that branches of conjunction are concatenated in a well defined order.

Figure 6: Pseudocode for FindPCs in the case of conjunction

2.2 A Gap Automaton

The set of pattern chains constructed by the function FindPCs can be represented as a *pattern tree* with patterns being the edges. For example, the pattern tree in Figure 4 corresponds to the tree displayed in Figures 2 and 3.

This pattern tree captures the history of gap propagations beginning at A. Assuming at that point only pattern P_1 is applicable, subtree B is produced. If P_2 yields subtree C, and at that point patterns P_3 and P_5 can be applied, this yields subtree D and exact location F (which is expressed by the termination symbol \$), respectively. Finally, pattern P_4 matches subtree D and proposes exact gap location E. It is important to note that this pattern tree can be thought of as an automaton, with $A, B, C, D, E,$ and F being the states and the pattern applications being the transitions.

Now, let us assign meaning of the states $A, B, C,$ and D to be the set of matching patterns, i.e., $A = \{P_1\}, B = \{P2\}, C = \{P_3, P_5\}, D = \{P_4\}$, and $E = F = \emptyset$. Given this representation, the pattern chains for the insertion of the gaps in our example would be as follows:

$$(\{P_1\}) \xrightarrow{P_1} (\{P_2\}) \xrightarrow{P_2} (\{P_3, P_5\}) \xrightarrow{P_3} (\{P_4\}) \xrightarrow{P_4,\$} (\emptyset)$$

$$(\{P_1\}) \xrightarrow{P_1} (\{P_2\}) \xrightarrow{P_2} (\{P_3, P_5\}) \xrightarrow{P_5,\$} (\emptyset)$$

With this representation, we can create a regular grammar using patterns as the terminals and their powerset as the non-terminals (adding a few more details like the start symbol) and production rules such as $\{P_2\} \rightarrow P_2\ \{P_3, P_5\}$. However, for our example the chain of patterns applied $P_1, P_2, P_3, P_4, \$$ could generate a pattern tree that is incompatible with the original tree. For example:

$$(\{P_1\}) \xrightarrow{P_1} (\{P_2\}) \xrightarrow{P_2} (\{P_3, P_5\}) \xrightarrow{P_3} (\{P_3, P_4\}) \xrightarrow{P_4,\$} (\emptyset)$$

which might correspond to something like *"that viewers will tune in to expect to see."* Note that this pattern chain belongs to a different *tree class*, which incidentally would have inserted the gap at a different location (VP see gap).

To overcome this problem we add additional constraints to the grammar to ensure that all parses the grammar generates belong to the same tree class. One way to do this is to include the start state of a transition as an element of the terminal, e.g., $\frac{P_2}{\{P_2\}}$, $\frac{P_3}{\{P_3,P_5\}}$. That is, we extend the terminals to include the left-hand side of the productions they are emitted from, e.g.,

$$\{P_2\} \quad \rightarrow \quad \frac{P_2}{\{P_2\}}\ \{P_3, P_5\}$$

$$\{P_3, P_5\} \quad \rightarrow \quad \frac{P_3}{\{P_3, P_5\}} \ \{P_4\}$$

and the sequence of terminals becomes: $\frac{P_1}{\{P_1\}} \frac{P_2}{\{P_2\}} \frac{P_3}{\{P_3,P_5\}} \frac{P_4}{\{P_4\}}$ \$.

Note that the grammar is unambiguous. For such a grammar, the question "what is the probability of a parse tree given a string and grammar" doesn't make sense; however, the question "what is the probability of a string given the grammar" is still valid, and this is essentially what we require to develop a generative model for gap insertion.

2.3 The Pattern Grammar

Let us define the pattern grammar more rigorously. Let Π be the set of patterns, and $\tilde{\Pi} \subset \Pi$ be the set of *terminal* patterns[3]. Let $pset(P)$ be the set of all subsets of patterns which include the pattern P, i.e., $pset(P) = \{\nu \cup \{P\} \mid \nu \in powerset(\Pi)\}$

- Let $T = \{\frac{P}{pset(P)} \mid P \in \Pi\} \bigcup \{\$\}$ be the set of terminals, where \$ is a special symbol[4].
- Let $N = \{S\} \bigcup powerset(\Pi)$ be the set of nonterminals with S being the start symbol.
- Let P be the set of productions, defined as the union of the following sets:
  1. $\{S \rightarrow \nu \mid \nu \in powerset(\Pi)\}$.
  2. $\{\nu \rightarrow \frac{P}{\nu}\ \mu \mid P \in \Pi - \tilde{\Pi}\ ,\ \nu \in pset(P)$ and $\mu \in powerset(\Pi)\}$. These are nonterminal transitions, note that they emit only non-terminal patterns.
  3. $\{\nu \rightarrow \frac{P}{\nu}\$ \mid P \in \tilde{\Pi}$ and $\nu \in pset(P)\}$. These are the terminal transitions, they emit a terminal pattern and the symbol \$.
  4. $\{\nu \rightarrow \frac{P}{\nu}\ \mu_1 \ldots \mu_n \mid P \in \Pi - \tilde{\Pi}\ ,\ \nu \in pset(P)$ and $\forall_{i \in [1..n]}\ \mu_i \in powerset(\Pi)\}$. This rule models conjunction with n branches.

2.4 Our Gap Model

Given the grammar defined in the previous subsection, we will define a probabilistic model for gap insertion. Recall that our goal is to find:

$$\underset{x,a,b}{argmax}\ Pr(g_{xx}^{ab}|T)$$

Just like the probability of a sentence is obtained by summing up the probabilities of its parses, the probability of the gap being at g_{xx}^{ab} is the sum of probabilities of all pattern chains that yield g_{xx}^{ab}.

[3]Patterns that generate exact position for a gap.

[4]Symbol \$ helps to separate branches in strings with conjunction.

$$Pr(g_{xx}^{ab}|T) = \sum_{pc_i \in \Upsilon} Pr(pc_i|T)$$

where $\Upsilon = \{pc \mid app(pc, T) = g_{xx}^{ab}\}$. Note that $pc_i \in TC(T)$ by definition.

For our model, we use two approximations. First, we collapse a tree T into its Tree Class $TC(T)$, effectively ignoring details irrelevant to gap insertion:

$$Pr(pc_i|T) \approx Pr(pc_i|TC(T))$$

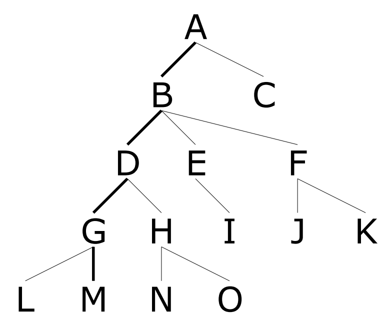

Figure 7: A pattern tree with the pattern chain $ABDGM$ marked using bold lines

Consider the pattern tree shown in Figure 7. The probability of the pattern chain $ABDGM$ given the pattern tree can be computed as:

$$\begin{aligned} Pr(ABDGM|TC(T)) &= \frac{Pr(ABDGM, TC(T))}{Pr(TC(T))} \\ &= \frac{\text{NR}(ABDGM, TC(T))}{\text{NR}(TC(T))} \end{aligned}$$

where $\text{NR}(TC(T))$ is the number of occurrences of the tree class $TC(T)$ in the training corpus and $\text{NR}(ABDGM, TC(T))$ is the number cases when the pattern chain $ABDGM$ leads to a correct gap in trees corresponding to the tree class $TC(T)$. For many tree classes, $\text{NR}(TC(T))$ may be a small number or even zero, thus this direct approach cannot be applied to the estimation of $Pr(pc_i|TC(T))$. Further approximation is required to tackle the sparsity issue.

In the following discussion, XY will denote an edge (pattern) between vertices X and Y in the pattern tree shown in Figure 7. Note that $Pr(ABDGM|TC(T))$ can be represented as:

$$Pr(AB|TC(T), A) \times Pr(BD|TC(T), AB) \times$$
$$\times Pr(DG|TC(T), ABD) \times Pr(GM|TC(T), ABDG)$$

We make an independence assumption, specifically, that $Pr(BD|TC(T), AB)$ depends only on states B, D, and the edge between them, not on the whole pattern tree or the edges above B, i.e., $Pr(BD|TC(T), AB) \approx Pr(BD, D|B)$. Note that this probability is equivalent to the probability of a production $Pr(B \overset{BD}{\rightarrow} D)$ of a PCFG.

Recall that the meaning assigned to a state in pattern grammar in Section 2.2 is the set of patterns matching at that state. Thus, according to that semantics, only the edges displayed bold in Figure 8 are involved in computation of $Pr(B \overset{BD}{\rightarrow} D)$. Written in the style we used for our grammar, the production is $\{BD, BE, BF\} \rightarrow \frac{BD}{\{BD,BE,BF\}}\{DG, DH\}$.

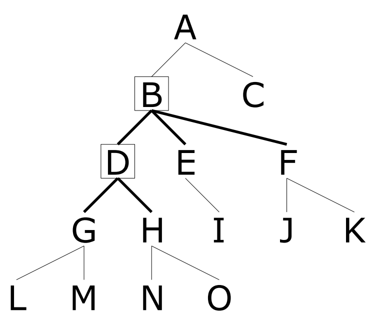

Figure 8: The context considered for estimation of the probability of transition from B to D

Pattern trees are fairly shallow (partly because many patterns cover several layers in a parse tree as can be seen in Figures 1 and 2); therefore, the context associated with a production covers a good part of a pattern tree. Another important observation is that the local configuration of a node, which is described by the set of matching patterns, is the most relevant to the decision of where the gap is to be propagated[5]. This is the reason why the states are represented this way.

Formally, the second approximation we make is as follows:

$$Pr(pc_i|TC(T)) \approx Pr(pc_i|G)$$

where G is a PCFG model based on the grammar described above.

$$Pr(pc_i|G) = \prod_{prod_j \in \mathcal{P}(pc_i)} Pr(prod_j|G)$$

where $\mathcal{P}(pc_i)$ is *the* parse of the pattern chain pc_i which is a string of terminals of G. Combining the formulae:

$$Pr(g_{xx}^{ab}|T) \approx \sum_{pc_i \in \Upsilon} Pr(pc_i|G)$$

Finally, since $Pr(TC(T)|G)$ is a constant for T,

$$\underset{x,a,b}{argmax}\ Pr(g_{xx}^{ab}|T) \approx \underset{x,a,b}{argmax} \sum_{pc_i \in \Upsilon} Pr(pc_i|G)$$

To handle conjunction, we must express the fact that pattern chains yield sets of gaps. Thus, the goal becomes:

$$\underset{(x_1,a_1,b_1),\ldots,(x_n,a_n,b_n)}{argmax} Pr(\{g_{x_1x_1}^{a_1b_1}, \ldots, g_{x_nx_n}^{a_nb_n}\}|T)$$

$$Pr(\{g_{x_1x_1}^{a_1b_1}, \ldots, g_{x_nx_n}^{a_nb_n}\}|T) = \sum_{pc_i \in \Upsilon} Pr(pc_i|T)$$

where $\Upsilon = \{pc \mid app(pc, T) = \{g_{x_1x_1}^{a_1b_1}, \ldots, g_{x_nx_n}^{a_nb_n}\}\}$. The remaining equations are unaffected.

2.5 Smoothing

Even for the relatively small number of patterns, the number of non-terminals in the grammar can potentially be large ($2^{|\Pi|}$). This does not happen in practice since most patterns are mutually exclusive. Nonetheless, productions, unseen in the training data, do occur and their probabilities have to be estimated. Rewriting the probability of a transition $Pr(A \rightarrow \frac{a}{A} B)$ as $\mathcal{P}(A, a, B)$, we use the following interpolation:

$$\begin{aligned} \tilde{\mathcal{P}}(A, a, B) &= \lambda_1 \mathcal{P}(A, a, B) + \lambda_2 \mathcal{P}(A, a) \\ &+ \lambda_3 \mathcal{P}(A, B) + \lambda_4 \mathcal{P}(a, B) + \lambda_5 \mathcal{P}(a) \end{aligned}$$

We estimate the parameters on the held out data (section 24 of WSJ) using a hill-climbing algorithm.

[5] We have evaluated a model that only uses $Pr(BD|\{BD, BE, BF\})$ for the probability of taking BD and found it performs only slightly worse than the model presented here.

3 Evaluation

3.1 Setup

We compare our algorithm under a variety of conditions to the work of (Johnson, 2002) and (Gabbard et al., 2006). We selected these two approaches because of their availability[6]. In addition, (Gabbard et al., 2006) provides state-of-the-art results. Since we only model the insertion of WH-traces, all metrics include co-indexation with the correct WH phrases identified by their type and word span.

We evaluate on three metrics. The first metric, which was introduced by Johnson (2002), has been widely reported by researchers investigating gap insertion. A gap is scored as correct only when it has the correct type and string position. The metric has the shortcoming that it does not require correct attachment into the tree.

The second metric, which was developed by Campbell (2004), scores a gap as correct only when it has the correct gap type and its mother node has the correct nonterminal label and word span. As Campbell points out, this metric does not restrict the position of the gap among its siblings, which in most cases is desirable; however, in some cases (e.g., double object constructions), it does not correctly detect errors in object order. This metric is also adversely affected by incorrect attachments of optional constituents, such as PPs, due to the span requirement.

To overcome the latter issue with Campbell's metric, we propose to use a third metric that evaluates gaps with respect to correctness of their lexical head, type of the mother node, and the type of the co-indexed wh-phrase. This metric differs from that used by Levy and Manning (2004) in that it counts only the dependencies involving gaps, and so it represents performance of the gap insertion algorithm more directly.

We evaluate gap insertion on gold trees from section 23 of the Wall Street Journal Penn Treebank (WSJ) and parse trees automatically produced using the Charniak (2000) and Bikel (2004) parsers. These parsers were trained using sections 00 through 22 of the WSJ with section 24 as the development set.

Because our algorithm inserts only traces of non-empty WH phrases, to fairly compare to Johnson's and Gabbard's performance on WH-traces alone, we remove the other gap types from both the gold trees and the output of their algorithms. Note that Gabbard et al.'s algorithm requires the use of function tags, which are produced using a modified version of the Bikel parser (Gabbard et al., 2006) and a separate software tool (Blaheta, 2003) for the Charniak parser output.

For our algorithm, we do not utilize function tags, but we automatically replace the tags of auxiliary verbs in tensed constructions with *AUX* prior to inserting gaps using tree surgeon (Levy and Andrew, 2006). We found that Johnson's algorithm more accurately inserts gaps when operating on auxified trees, and so we evaluate his algorithm using these modified trees.

In order to assess robustness of our algorithm, we evaluate it on a corpus of a different genre – Broadcast News Penn Treebank (BN), and compare the result with Johnson's and Gabbard's algorithms. The BN corpus uses a modified version of annotation guidelines, with some of the modifications affecting gap placement.

```
Treebank 2 guidelines (WSJ style):
(SBAR (WHNP-2 (WP whom))
  (S (NP-SBJ (PRP they))
    (VP (VBD called)
      (S (NP-SBJ (-NONE- *T*-2))
         (NP-PRD (NNS exploiters))))))
```

```
Treebank 2a guidelines (BN style):
(SBAR-NOM (WHNP-1 (WP what))
 (S (NP-SBJ (PRP they))
  (VP (VBP call)
    (NP-2 (-NONE- *T*-1))
      (S-CLR (NP-SBJ (-NONE- *PRO*-2))
        (NP-PRD (DT an) (NN epidemic))))))
```

Since our algorithms were trained on WSJ, we apply tree transformations to the BN corpus to convert these trees to WSJ style. We also auxify the trees as described previously.

3.2 Results

Table 1 presents gap insertion F measure for Johnson's (2002) (denoted J), Gabbard's (2006) (denoted G), and our (denoted Pres) algorithms on section 23 gold trees, as well as on parses generated by the Charniak and Bikel parsers. In addition to WHNP and WHADVP results that are reported in the literature, we also present results for WHPP gaps even though there is a small number of them in section 23 (i.e., 22 gaps total). Since there are only 3 non-empty WHADJP phrases in section 23, we omit them in our evaluation.

[6] Johnson's source code is publicly available, and Ryan Gabbard kindly provided us with output trees produced by his system.

	Metric	Gold Trees J	Gold Trees G	Gold Trees Pres	Charniak Parser J	Charniak Parser G	Charniak Parser Pres	Bikel Parser J	Bikel Parser G	Bikel Parser Pres
WHNP	Johnson	94.8	90.7	97.9	89.8	86.3	91.5	90.2	86.8	92.6
	Campbell	94.8	97.0	99.1	81.9	83.8	83.5	80.7	81.5	82.2
	Head dep	94.8	97.0	99.1	88.8	90.6	91.0	89.1	91.4	92.3
WHADVP	Johnson	75.5	91.4	96.5	61.4	78.0	80.0	61.0	77.9	77.2
	Campbell	74.5	89.1	95.0	61.4	71.7	78.4	60.0	71.5	74.8
	Head dep	75.5	89.8	95.8	64.4	78.0	84.7	63.0	77.1	80.3
WHPP	Johnson	58.1	N/R	72.7	35.7	N/R	55.0	42.9	N/R	53.7
	Campbell	51.6	N/R	86.4	28.6	N/R	60.0	35.7	N/R	63.4
	Head dep	51.6	N/R	86.4	35.7	N/R	70.0	35.7	N/R	73.2

Table 1: F1 performance on section 23 of WSJ (N/R indicates not reported)

Compared to Johnson's and Gabbard's algorithm, our algorithm significantly reduces the error on gold trees (table 1). Operating on automatically parsed trees, our system compares favorably on all WH traces, using all metrics, except for two instances: Gabbard's algorithm has better performance on WHNP, using Cambpell's metric and trees generated by the Charniak parser by 0.3% and on WHADVP, using Johnson's metric and trees produces by the Bikel parser by 0.7%. However, we believe that the dependency metric is more appropriate for evaluation on automatically parsed trees because it enforces the most important aspects of tree structure for evaluating gap insertion. The relatively poor performance of Johnson's and our algorithms on WHPP gaps compared that on WHADVP gaps is probably due, at least in part, to the significantly smaller number of WHPP gaps in the training corpus and the relatively wider range of possible attachment sites for the prepositional phrases.

Table 2 displays how well the algorithms trained on WSJ perform on BN. A large number of the errors are due to *FRAG*s which are far more common in the speech corpus than in WSJ. WHPP and WHADJP, although more rare than the other types, are presented for reference.

3.3 Error Analysis

It is clear from the contrast between the results based on gold standard trees and the automatically produced parses in Table 1 that parse error is a major source of error. Parse error impacts all of the metrics, but the patterns of errors are different. For WHNPs, Campbell's metric is lower than the other two across all three algorithms, suggesting that this metric is adversely affected by factors that do not impact the other metrics (most likely the span of the gap's mother node). For WHADVPs, the metrics show a similar degradation due to parse error across the board. We are reluctant to draw conclusions for the metrics on WHPPs; however, it should be noted that the position of the PP should be less critical for evaluating these gaps than their correct attachment, suggesting that the head dependency metric would more accurately reflect the performance of the system for these gaps.

Campbell's metric has an interesting property: in parse trees, we can compute the upper bound on recall by simply checking whether the correct WH-phrase and gap's mother node exist in the parse tree. We present recall results and upper bounds in Table 3. Clearly the algorithms are performing close to the upper bound for WHNPs when we take into account the impact of parse errors on this metric. Clearly there is room for improvement for the WHPPs.

	Metric	J	G	Pres
WHNP	Johnson	88.0	90.3	92.0
	Campbell	88.2	94.0	95.3
	Head dep	88.3	94.0	95.3
WHADVP	Johnson	76.4	92.0	94.3
	Campbell	76.3	88.2	92.4
	Head dep	76.3	88.5	92.5
WHPP	Johnson	56.6	N/R	75.7
	Campbell	60.4	N/R	91.9
	Head dep	60.4	N/R	91.9
WHADJP	Johnson	N/R	N/R	89.8
	Campbell	N/R	N/R	85.7
	Head dep	N/R	N/R	85.7

Table 2: F1 performance on gold trees of BN

In addition to parser errors, which naturally have the most profound impact on the performance, we found the following sources of errors to have impact on our results:

- Annotation errors and inconsistency in PTB, which impact not only the training of our system, but also its evaluation.

Charniak Parser	J	G	Pres	UB
WHNP	81.9	82.8	83.5	84.0
WHADVP	61.4	71.7	78.4	81.1
WHPP	28.6	N/R	60.0	86.4
Bikel Parser	J	G	Pres	UB
WHNP	77.0	80.5	81.5	82.0
WHADVP	47.2	70.1	74.8	78.0
WHPP	22.7	N/R	59.1	81.8

Table 3: Recall on trees produced by the Charniak and Bikel parsers and their upper bounds (UB)

1. There are some POS labeling errors that confuse our patterns, e.g.,
```
(SBAR (WHNP-3 (IN that))
    (S (NP-SBJ (NNP Canada))
        (VP (NNS exports)
            (NP (-NONE- *T*-3))
            (PP ...))))
```
2. Some WHADVPs have gaps attached in the wrong places or do not have gaps at all, e.g.,
```
(SBAR (WHADVP (WRB when))
    (S (NP (PRP he))
        (VP (VBD arrived)
            (PP (IN at)
                (NP ...))
            (ADVP (NP (CD two)
                    (NNS days))
                (JJ later)))))
```
3. PTB annotation guidelines leave it to annotators to decide whether the gap should be attached at the conjunction level or inside its branches (Bies et al., 1995) leading to inconsistency in attachment decisions for adverbial gaps.

- Lack of coverage: Even though the patterns we use are very expressive, due to their small number some rare cases are left uncovered.
- Model errors: Sometimes despite one of the applicable pattern chains proposes the correct gap, the probabilistic model chooses otherwise. We believe that a lexicalized model can eliminate most of these errors.

4 Conclusions and Future Work

The main contribution of this paper is the development of a generative probabilistic model for gap insertion that operates on subtree structures. Our model achieves state-of-the-art performance, demonstrating results very close to the upper bound on WHNP using Campbell's metric. Performance for WHADVPs and especially WHPPs, however, has room for improvement.

We believe that lexicalizing the model by adding information about lexical heads of the gaps may resolve some of the errors. For example:

```
(SBAR (WHADVP-3 (WRB when))
  (S (NP (NNP Congress))
    (VP (VBD wanted)
      (S (VP (TO to)
        (VP (VB know) ...)))
      (ADVP (-NONE- *T*-3)))))

(SBAR (WHADVP-1 (WRB when))
 (S (NP (PRP it))
   (VP (AUX is)
     (VP (VBN expected)
       (S (VP (TO to)
            (VP (VB deliver) ...
              (ADVP (-NONE- *T*-1)))))))))
```

These sentences have very similar structure, with two potential places to insert gaps (ignoring reordering with siblings). The current model inserts the gaps as follows: *when Congress (VP wanted (S to know) gap)* and *when it is (VP expected (S to deliver) gap)*, making an error in the second case (partly due to the bias towards shorter pattern chains, typical for a PCFG). However, *deliver* is more likely to take a temporal modifier than *know*.

In future work, we will investigate methods for adding lexical information to our model in order to improve the performance on WHADVPs and WHPPs. In addition, we will investigate methods for automatically inferring patterns from a treebank corpus to support fast porting of our approach to other languages with treebanks.

5 Acknowledgements

We would like to thank Ryan Gabbard for providing us output from his algorithm for evaluation. We would also like to thank the anonymous reviewers for invaluable comments. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-06-C-0023. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

A. Bies, M. Ferguson, K. Katz, and R. MacIntyre. 1995. Bracketing guidelines for treebank II style Penn Treebank project. Technical report.

D. M. Bikel. 2004. *On the Parameter Space of Gen-*

erative Lexicalized Statistical Parsing Models. Ph.D. thesis, University of Pennsylvania.

D. Blaheta. 2003. *Function Tagging*. Ph.D. thesis, Brown University.

R. Campbell. 2004. Using linguistic principles to recover empty categories. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

E. Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the North American Chapter of the Association for Computational Linguistics*.

M. Collins. 1999. *Head-driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

P. Dienes and A. Dubey. 2003. Antecedent recovery: Experiments with a trace tagger. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*.

R. Gabbard, S. Kulick, and M. Marcus. 2006. Fully parsing the Penn Treebank. In *Proceedings of the North American Chapter of the Association for Computational Linguistics*.

D. Higgins. 2003. A machine-learning approach to the identification of WH gaps. In *Proceedings of the Annual Meeting of the European Chapter of the Association for Computational Linguistics*.

M. Johnson. 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

R. Levy and G Andrew. 2006. Tregex and Tsurgeon: Tools for querying and manipulating tree data structures. In *Proceedings of LREC*.

R. Levy and C. Manning. 2004. Deep dependencies from context-free statistical parsers: Correcting the surface dependency approximation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

W. Wang and M. P. Harper. 2002. The SuperARV language model: Investigating the effectiveness of tightly integrating multiple knowledge sources in language modeling. In *Proceedings of the Empirical Methods in Natural Language Processing*.

W. Wang, M. P. Harper, and A. Stolcke. 2003. The robustness of an almost-parsing language model given errorful training data. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*.