Parsing as Statistical Pattern Recognition

David M. Magerman
IBM T. J. Watson Research Center

January 16, 1995

1 Introduction

Parser development is generally viewed as a primarily linguistic enterprise. A grammarian examines sentences, skillfully extracts the linguistic generalizations evident in the data, and writes grammar rules which cover the language. The grammarian then evaluates the performance of the grammar, and upon analysis of the errors made by the grammar-based parser, carefully refines the rules, repeating this process on and off for about a decade.

This grammar refinement process is extremely time-consuming and difficult, and has not yet resulted in a grammar which can be used by a parser to analyze accurately a large corpus of unrestricted text. As an alternative to writing grammars, one can develop corpora of hand-analyzed sentences (*treebanks*) with significantly less effort. With the availability of treebanks of annotated sentences, one can view NL parsing as simply *treebank recognition.*

The work described in this paper represents a movement away from traditional grammar-based parsing towards parsing as pattern recognition, where the pattern to be recognized is a linguistic analysis of a sentence. This approach divides the parsing problem into two separate tasks: *treebanking,* defining the annotation scheme which will encode the linguistic content of the sentences and applying it to a corpus, and *treebank recognition,* generating these annotations automatically for new sentences.

The treebank can contain whatever information is deemed valuable by the treebanker, as long as it is annotated according to some consistent scheme, probably one which represents the intended meaning of the sentence. If one views the treebank as a black box which, given a sentence, assigns a unique analysis to that sentence, then the goal of treebank recognition is to produce the exact same analysis of a sentence that the treebank would generate.

The solution to the treebank recognition problem described in this paper is called SPATTER (Statistical PATTErn Recognizer). The SPATTER parser uses probabilistic models to predict the part-of-speech labels, parse tree edges, and constituent labels for a given sentence. The parsing process is divided into a sequence of actions which add structure to the sentence in a bottom-up fashion. The probabilistic models used to predict each action condition their distributions on the information made available by the sequence of actions preceding the current action. Thus, the derivational order of the actions leading to an analysis affects the probability assigned to the parse. In [4], this type of model is referred to as a history-based grammar model. A derivational model, which assigns a distribution to the possible derivations of a parse tree, is self-organized in the process of training the other models. The probabilistic models in SPATTER are estimated using statistical decision trees, and these distributions are refined using two different applications of the expectation-maximization (E-M) algorithm.

In this paper, I will define and motivate the treebank recognition problem (TRP), discuss the knowledge representation issues involved in SPATTER parsing, describe in detail the statistical modelling techniques used in this work, and present results of both the training and testing of SPATTER.

2 The Treebank Recognition Problem

The purpose of introducing the treebank recognition problem (TRP) is to remove a significant obstacle impeding progress in the area of natural language (NL) parsing. One of the most difficult aspects of NL parsing is that, to the casual observer, or to a dedicated parsing specialist for that matter, it is not quite clear exactly what the parsing problem is. To wit, parsing has something to with assigning syntactic constituent structure to sentences. But this simplistic definition does little more than replace one unknown term (parsing) with another (syntactic constituent structure). Disregarding the issue of what "syntactic constituent structure" means, many would argue that a parse is much more than syntactic structure alone. To some, parsing involves assigning semantic labels to constituents as well. Still others expect parse trees to indicate predicate-argument relations, along with resolving anaphoric and elliptical references. To make the task even more difficult, linguists have divergent views on the definitions of semantics and predicate-argument relations. In fact, it is difficult to pin down precise definitions to many of the features of a sentence that parsers are expected to identify. In summary, the natural language parsing problem is an ill-defined task.

In contrast, the TRP is *posed* as a well-defined task. The annotation scheme used to analyze the treebank sentences *defines* the expected output of the parser. For instance, if a linguist can consistently annotate sentences with predicate-argument structure, then it is the goal of the treebank recognition problem to identify that structure in new sentences accurately.

The most significant advantage the TRP has over the parsing problem is in evaluation of different approaches and implementations. For a given treebank, divided into training data and unseen test data, the best solution to the TRP is the one which assigns THE correct analysis to the most test sentences. Comparative evaluation of solutions to the parsing problem has been hindered by the claim that there is no one *correct* parse, i.e. any reasonable analysis of a sentence is correct, where a "reasonable" analysis is one supported by any one of the diverse and numerous linguistic theories available on a given day. The TRP explicitly rules out this argument. For better or for worse, the one and only *correct* parse of sentence is the one which the treebank assigns to that sentence. Of course, the treebank may have internal inconsistencies. While statistical training methods can overcome some inconsistencies in the training data, annotation schemes which border on the random will be difficult to reproduce accurately. And errors in the test data are unavoidable. But, assuming the description task, as discussed in the introduction, results in a labeling scheme which can be consistently assigned to text by humans, the inconsistencies in both the training and testing sentences should be negligible.

The remainder of this paper will discuss SPATTER, a proposed solution to the treebank recognition problem for the Lancaster Treebank of IBM Computer Manuals.

3 Representing Parts of Parse Trees

The initial development of SPATTER was guided by the principle that the best representation to start with is the most straightforward. The rationale behind this principle is that, if experiments based on a straightforward representation failed, error analysis might reveal obvious flaws in the representation which could be corrected. But, if solving the TRP is reduced to a search for the best representation of a parse tree, then there is little hope for success.

3.1 Four Elementary Features

The treebank can be viewed as a collection of n-ary branching trees, with each node in a tree labeled by either a non-terminal label, a part-of-speech label, or a word token. For the purposes of pattern recognition, these trees must be described precisely, in terms of elementary components.

Grammarians elevate constituents to the status of elementary units in a parse, especially in the case of rewrite-rule grammars where each rewrite rule defines a legal constituent. If a parse tree is interpreted as a geometric pattern, however, a constituent is no more than a set of edges which meet at the same tree node. For instance, the noun phrase, "a brown cow," consists of an edge extending to the left from "a," an edge

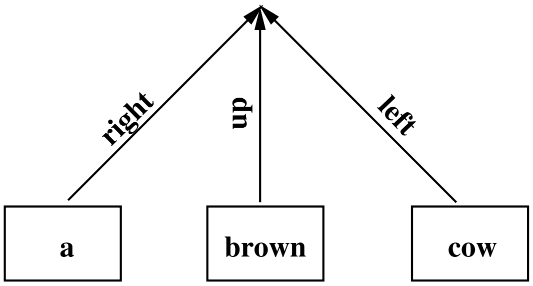

Figure 1: Representation of constituent and labeling of extensions in SPATTER.

extending to the right from "cow," and an edge extending straight up from "brown" (see figure 1). This is precisely how constituents are represented in SPATTER.

The parse tree is encoded in terms of into four elementary components, or *features*: words, tags, labels, and extensions. Each feature has a fixed vocabulary, with each element of a given feature vocabulary having a unique representation.

The input to the treebank recognizer is a sentence, which is a sequence of tokens. The first feature, the *word* feature, defines the mapping from these tokens onto a fixed vocabulary of words, including an "unknown" word for all tokens which do not map onto a word in the vocabulary. The mapping is almost trivial, with most tokens mapping to the vocabulary item with the same spelling. However, because the capitalization in a sentence sometimes contains useful information, e.g. in titles, and sometimes doesn't, e.g. at the beginning of a sentence, the capitalization of the tokens is altered according to a deterministic algorithm before the tokens are mapped onto the word vocabulary.

The next feature assigned, the *tag* feature, represents the part-of-speech information for each word in the sentence. The tag vocabulary is defined to be the set of all part-of-speech labels which are assigned to words in the treebank training data.

Once a leaf node has been assigned a tag feature value, it is extended in some direction towards a parent node. The direction of this extension is represented by the *extension* feature. This feature takes on one of 4 possible values, *left, right, up, or unary.* For example, in the constituent, "a brown cow," the correct extension feature value for the word "a" is *right,* for "cow" *left,* and for "brown" *up,* as is shown in figure 1. The extension feature value *unary* is reserved for the case where a tree node extends directly up to another tree node, as in the case of "I saw the brown cow," where the pronoun "I" might be parsed as a noun phrase constituent.

A constituent is defined in terms of the extension feature values of consecutive nodes. A node with extension value *right* followed by any number of nodes with extension value *up* followed by a node with extension value *left* is a constituent. Any node which is assigned the *unary* extension value is also a constituent. Whenever the parser detects a sequence of nodes which corresponds to a constituent, it generates a new node and begins the process of assigning feature values to that node. The first feature value assigned to an internal node in a parse tree is a *label* feature value, which encodes information about the constituent beneath that node.

3.2 Propogating Lexical Information

The constituent labels in a treebank are not very descriptive. It would be difficult for human treebankers to assign constituent labels to text consistently, accurately, and efficiently if the labels conveyed subtle nuances of meaning. For example, the Lancaster Treebank uses only 17 constituent labels, whereas the grammar described in [2] covering the same domain assigns over 13,000 unique non-terminal categories.

Given the dearth of information in the label set, it would be difficult to model the assignment of feature values by asking questions primarily of the *label* values. Questions about the words and part-of-speech tags

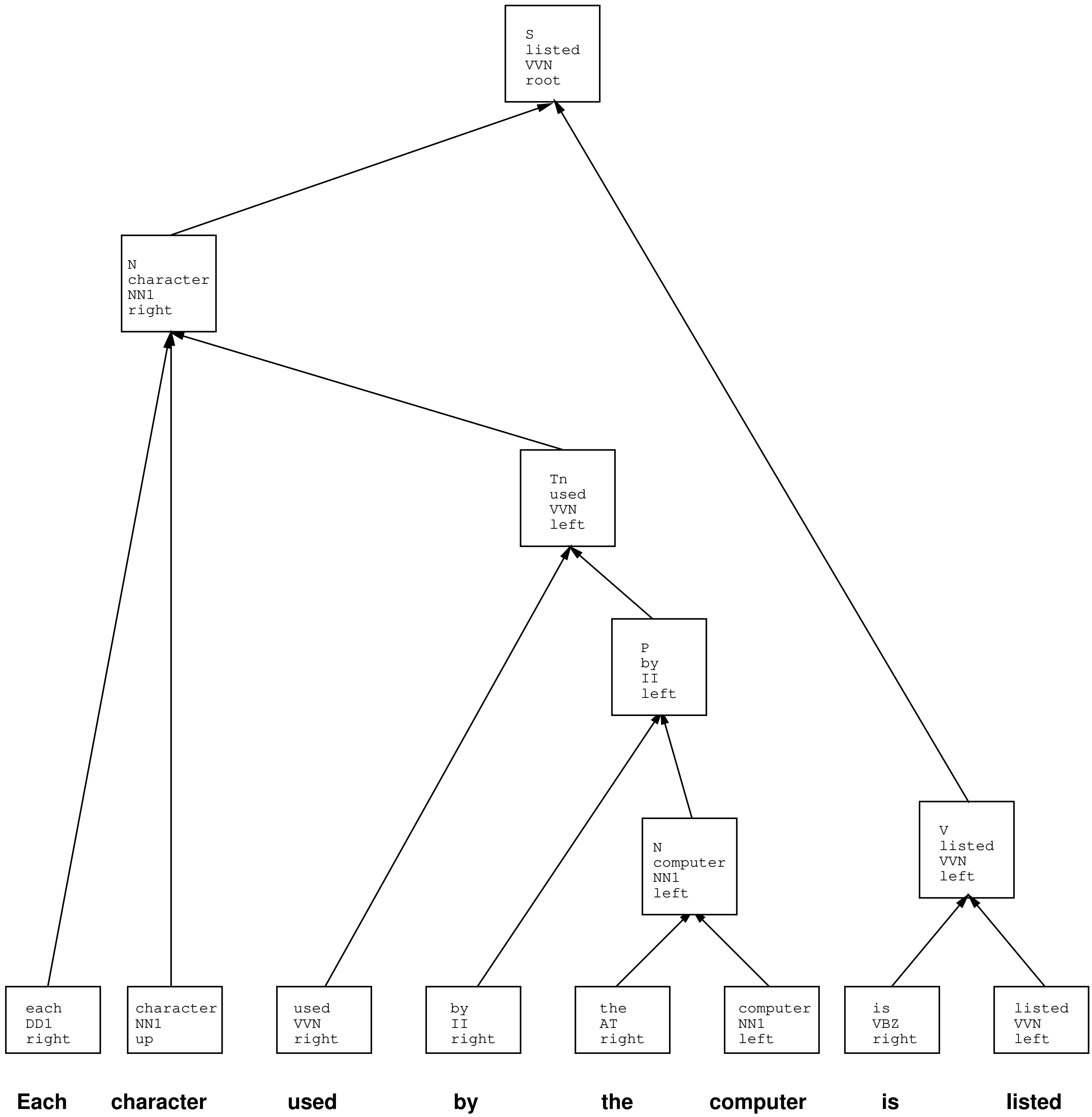

Figure 2: Treebank analysis encoded using feature values. Each internal node contains, from top to bottom, a *label, word, tag,* and *extension* value, and each leaf node contains a *word, tag,* and *extension* value.

are more informative, but it is infeasible to ask questions about arbitrarily selected *word* and *tag* values from a constituent.

To augment the information available at constituent nodes, a word, along with its corresponding part-of-speech tag, is selected from each constituent to act as a lexical representative. Thus, at each internal node in the parse tree, there is a *word* and a *tag* feature value which is deterministically assigned, as well as the *label* and *extension* values which are predicted by models.[1]

The lexical representative from a constituent loosely (*very* loosely) corresponds to the linguistic notion of a head word. For example, the lexical representative of a noun phrase is the rightmost noun, and the lexical representative of a verb phrase is the leftmost non-auxiliary verb. However, the correlation to linguistic theory ends there. The deterministic rules which select the representative word from each constituent were developed in the better part of an hour, in keeping with the philosophy of avoiding excessive dependence on rule-based methods. If the performance of these methods depend on the precise word selection rule set, there is little hope of success.

To illustrate the mapping between a treebank parse and the feature value representation scheme, consider the following sentence from the IBM computer manuals domain, as annotated in the Lancaster Treebank:

```
[N Each_DD1 character_NN1 [Tn used_VVN [P by_II [N the_AT computer_NN1 N] P] Tn] N]
[V is_VBZ listed_VVN V] ._.
```

The node feature value representation of this treebank analysis is shown in figure 2.

3.3 Decision Tree Questions

In SPATTER, a combination of binary statistical decision tree (SDT) models assign a probability distribution on the space of possible parse trees. The exact form of these models will be discussed in a later section, but the use of binary SDTs brings up an interested representation problem. Binary SDTs require that the contexts, or histories, be represented in terms of answers to binary questions. Questions like "Are the words *and* or *or* in the sentence?" have convenient binary answers. But questions like "What is the word at the current node?" or "How many children does the current node's constituent have?" have more than 2 answers. This problem is solved by creating a *binary classification tree* for the answers to each question.[2]

3.3.1 Binary Classification Tree Feature Representations

SDTs ask questions about the contextual history which help them predict the future more accurately, i.e. which reduce the entropy (uncertainty) of the future. In the context of SPATTER, these questions are about the feature values of nearby nodes in the parse tree. To simplify the mathematical algorithms used in training SDTs, it is very useful to enforce a binary structure on the decision trees. In a binary decision tree, only binary questions can be asked of the history. Of course, a question which has four possible answers can be encoded as a sequence of binary questions.

Consider a 4-valued question which asks about the an extension feature value. Without the binary question constraint, one can ask a question like "What is the extension feature value of the parse node to the left of the current node?" With this constraint, one can ask the same question using the sequence of questions shown in figure 3. However, other sequences of questions might be more predictive of the future. For instance, one might want to ask if the feature value is *left* or *right* first, indicating a constituent boundary, as shown in figure 4.

[1]The decision trees still ask questions about the occurrence of individual words in the constituent, but the questions about the lexical representative of a constituent turn out to be the most statistically informative questions.

[2]This paper contains references to three different types of trees: parse trees, decision trees, and classification trees. It is important to keep in mind which type of tree is being discussed. The parse trees are constructed using decision tree models. The answers to the questions asked by the decision tree models are encoded using classification trees. Now, if you were a tree, which kind would you be?

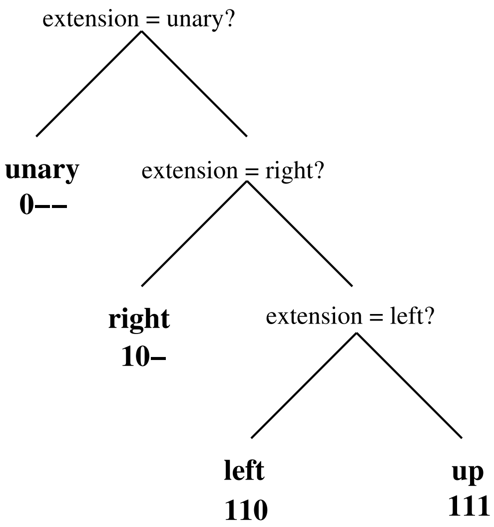

Figure 3: Classification tree for extension feature values with three binary questions. The hyphen (-) in the binary encoding indicates that the value of that bit can be either 0 or 1. The value of the bit in those cases can be selected to maximize the information content of that bit with respect to the value being predicted.

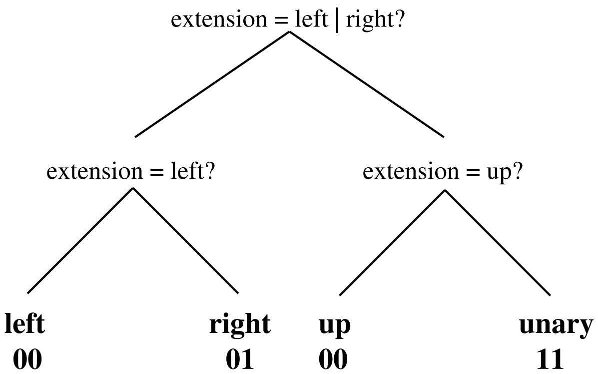

Figure 4: Classification tree for extension feature values with two binary questions.

Each of these sequences of questions corresponds to a binary classification tree on the extension vocabulary, with each extension feature value assigned a binary representation. The word, tag, and label feature vocabularies are also assigned binary representations using the same scheme.

Ideally, the classification trees used to represent the features should be constructed so that the set of values beneath each node in the classification tree provide information about the future. For instance, if one were trying to predict the label of a constituent by asking about the extension value of the node to the right of that constituent, then the representation for extensions suggested by figure 3 would be particularly useful if, in general, the distribution of the labels of constituents next to *unary* constituents are significantly different from those next to nodes extended in any other direction either to the *left*, to the *right*, or *up*.

While bit representations based on distributional behavior are ideal, it is very computationally expensive to create representations which produce this attractive predictive behavior. Previous work on n-gram class models [6] illustrates a method for distributionally clustering words based on bigram mutual information. However, this clustering method generates classes which reduce uncertainty about the next word, but do not necessarily reduce uncertainty about syntactic structure in a sentence. Also, it is not clear how to extrapolate these methods to clustering constituent labels and extensions, since these feature values cannot easily be ordered, from left to right, as bigrams in the same way as words or part-of-speech tags in a sentence. Finally, since a given node feature value is used to predict all four feature values elsewhere in the parse tree, it would be best to produce different binary representations for *each* type of decision made by the models. The amount of data and computation time necessary for such a calculation is surely prohibitive.

So, in the interests of expediency, the classification trees for the word and tag features are generated using the mutual information clustering algorithm, and the remaining feature value classification trees are generated by hand. For the hand-generated classes, some redundancy is encoded in the bit representations, with the hope that each statistical decision tree will find some sequence of questions to ask which will predict the future well.

3.3.2 More Questions?

The previous section discusses predicting node feature values by asking binary questions about feature values of nearby nodes. However, this information does not easily capture generalizations about the local context. For instance, the number of children nodes in a constituent is useful when assigning a constituent label, but this information isn't readily accessible from the node feature values without asking a number of questions about the specific *extension* values of the node's children. Also, the span of a constituent, i.e. the number of words contained within the constituent, also might alter the distributions of *label* and *extension* feature value assignments. This information is extremely difficult to extract by asking only about feature values.

There are many such questions about the structure of the parse tree which might provide predictive information for the decision tree to ask about. However, using the principle that the best representation to start with is the most straightforward, only the most obvious aspects of a parse tree are encoded as questions: the number of children of a node, the span of a node, and the number of constituent nodes in the sentence.[3]

3.4 A Feature for Conjunction

After analyzing preliminary parsing results, it was clear that the SPATTER was not able to predict the scope of conjunctions very accurately. The precise reason is not clear, but a contributing factor is that there is no single feature which represents the fact that two or more constituents will be conjoined. And while there are tell-tale signs that a conjunction is going to occur in a sentence, e.g. the words *and* or *or*, or a comma (,) or semi-colon (;), the cues for the boundaries of the conjuncts are not obviously encoded in the four feature values or the other questions described above.[4]

[3]Note that the answer to this final question, about the number of constituent nodes, changes as the parser adds structure to the sentence.

[4]Now try to parse that sentence automatically!

Coincidentally, the Lancaster Treebank annotation scheme encodes information about conjunction which was not used in the initial experiments. The treebank indicates which constituents are conjoined by appending a symbol to the end of the constituent label. This information can be viewed as one bit of information attached to each constituent, in other words, a fifth feature of the parse tree. This binary-valued node feature, the *conjunction* feature, takes on a true value only when the constituent node is going to be a part of a conjoined phrase.

4 SPATTER Parsing: The Algorithm

In SPATTER parsing, a parse tree is represented as a connected, single-rooted graph with feature values at each node, one of which indicates the geometry of the parse (the *extension* feature). Since the probability of a feature value assignment at a particular node is conditioned on the information available at other nodes in the partially-constructed tree, the probability of a parse tree derived in a certain order is different from the probability of the same parse tree derived in a different order.

4.1 Derivations of Parse Trees

With five feature values assigned at each internal node and four feature values assigned at each leaf, there are far too many different derivations of a single parse tree to consider. If derivations were completely free order, a parse tree with 20 nodes would have over 10^{150} (100!) possible derivations.

For the purposes of this work, the only derivations which will be considered are bottom-up derivations. In a bottom-up derivation, a node is not constructed until all of the node feature values beneath it are assigned. One simple derivation is the bottom-up leftmost derivation, in which the parsing process begins with the leaf nodes, and feature values are filled out in order from left to right. But there are many other derivations which might allow more predictive information to be available at crucial points in the parsing process. For instance, the extension feature value of a noun phrase node which is sandwiched between a verb and a prepositional phrase determines the prepositional phrase attachment. In the bottom-up leftmost derivation, the noun phrase extension would be predicted before any information was known about the prepositional phrase and its argument noun phrase. In other derivations both constituent nodes would both be within the extension model's five node window. On the other hand, not all derivations provide information by deviating from the bottom-up leftmost derivation. Predicting the part-of-speech tag of the last word in a long sentence is unlikely to affect the distribution of the part-of-speech tag of the first word in the sentence.

To constrain the number of derivations while still allowing informative derivations to be considered, a *derivational window constraint* is used.[5] Under a derivational window constraint of size n, the first n possible feature value assignments, from left to right, are considered. The nodes at which these candidate feature value are to be assigned are called *active* nodes.

As an example, at the start of the parsing process, when only the word feature values are assigned, a window constraint of 3 would allow any one of the first three part-of-speech tags to be assigned, and thus the first three nodes in the sentence are active. Let's say the second word is tagged. Then, the window constraint permits either the first or third word to be tagged or the second word to be extended, and again the first three nodes are active.

Even with a derivational window constraint of 2, there are too many possible derivations to consider. To eliminate this difficulty, the derivational window constraint is set to 1 for all but the extension and tag feature assignments. This means that, initially, the tag feature value can be assigned for the first n words according to the derivational window constraint of n. However, as soon as an internal node is created (i.e. a constituent is recognized), the label and conjunction feature values for that node are assigned.

[5] The idea of a derivational window constraint was first suggested by Fred Jelinek, and the idea was developed by the Jelinek, Bob Mercer, Salim Roukos, Adwait Ratnaparkhi, and Barbara Gates.

4.2 Stepping Through an Example

With so many different derivations, even under the constraints used in SPATTER, it is useful to step through the beginning of one derivation of a parse tree. For this purpose the sentence and parse tree in Figure 2 will be used. The constraints described above, including a derivational window constraint of 2, are used in the example.

Before parsing begins, each token in the sentence is assigned to a leaf node. Each leaf node is assigned a word feature value deterministically based on the node's token. The parsing algorithm then proceeds as follows:

The first 2 active nodes in the sentence are identified. (An active node is a node which has an unassigned feature value.) One of these two nodes is selected, and the next unfilled feature slot in that node is assigned a value. If the action of filling the feature value caused the node to be complete, the adjacent nodes are examined to see if a constituent is indicated by the extension feature values (as discussed earlier). If so, the appropriate nodes are combined into a new node, and the label and conjunction feature values are assigned for that new node. This process repeats until only one node remains.

In the example sentence, initially the first two active nodes are the leaf nodes for **each** and *character.* Two different derivations are generated depending upon which of these two nodes is selected. Let's say the second node is selected, and the part-of-speech tag, **NN1,** is assigned. The same two nodes are still active, and let's say the second node is selected again. This time the extension feature value is set to **up.** Now, the second node is no longer active, since it is *completed,* i.e. all of its feature value are assigned. Also, it cannot be combined with adjacent nodes to form a constituent. At this point, the first and third nodes are identified to be active. Let's say the first node can be selected, and is assigned the tag value, **DD1.**

Skipping a few steps, consider the situation where the leaf node for the word **the** is completed, and the **computer** node is selected as the active node and assigned the **left** extension value. The parser recognizes that these two adjacent nodes form a constituent, since a node extended to the right is followed by a node extended to the left. These nodes are connected to an empty parent node, and label and conjunction feature values are assigned to the node.

To simplify the parsing process further, the conditional models used by the parser to assign probabilities to feature values and active node selections do not ask explicit questions about the derivational order of operations. Certainly, the probabilities are affected by the absence of information in certain derivations. But, if the first two words in a sentence have already been tagged, the probability of the next step is not sensitive to the order in which the first two words were tagged. This significantly aids the resources necessary for parsing, since all partial parses with identical feature values represent the same state in the space of partial parses, and can be merged without loss of information.

5 Statistical Parsing Models

The SPATTER parser assigns probability to a parse tree T given the sentence S, $\mathcal{P}(T \mid S)$. This is referred to as a *parsing* model, which is distinguished from a *generative* model, i.e. one which assigns probability to a parse tree and a sentence, $\mathcal{P}(T, S)$.

The most common type of probabilistic parsing model is the P-CFG model [13] [3], which assigns a probability to each rule in a context-free grammar and computes the probability of the parse tree by assuming that each grammar rule application is independent of all other rule applications in the sentence. As illustrated in [10] [4] and [11], probabilistic parsers are much more accurate when their models incorporate lexical information from the context, and when the applications of the models are not assumed to be independent. The development of history-based grammar models is based on the premise that each decision potentially affects future decisions in the parsing process.

The statistical models in SPATTER can ask questions about all feature value assignments made prior to the current parse action. Due to constraints on resources, both computational and physical, the models are limited to asking questions about two constituent nodes to the left of the current node, two constituent nodes to the right of the current node, and up to four children of the current node. However, considering the limited amount of training data available, it would be difficult to estimate accurately the information content of nodes outside of this five node window.

5.1 Probabilistic Models for Node Features

The probability distribution for each feature value is estimated using conditional models in the form of statistical decision trees.

As stated previously, the decision tree models are conditioned on information from a five node window, including the node processed and its children. Since the ordering of feature value assignments is not fixed, some feature value slots in the nodes within this window will be empty, i.e. they may not have been assigned yet. If a node feature value being queried by the decision tree is unassigned, the decision tree is licensed to ask about that same feature value from the nearest child of that node.

5.1.1 Notation

In the remainder of this section, the following notational scheme will be used. w_i and t_i refer to the word corresponding to the ith token in the sentence and its part-of-speech tag, respectively. N^k refers to the 4-tuple of feature values at the kth node in the current parse state, where the nodes are numbered from left to right. N_l^k, N_w^k, N_t^k, N_c^k, and N_e^k refer, respectively, to the label, word, tag, conjunction, and extension feature values at the node k. N^{c_j} refers to the jth child of the current node where the leftmost child is child 1. $N^{c_{-j}}$ refers to the jth child of the current node where the rightmost child is child 1. The symbol Q_{etc} refers to miscellaneous questions about the current state of the parser, such as the number of nodes in the sentence and the number of children of a particular node.

5.1.2 The Tagging Model

The tag feature value prediction is conditioned on the two words to the left, the two words to the right, and all information at two nodes to the left and two nodes to the right.

$$\mathcal{P}(t_i \mid context) \approx \mathcal{P}(t_i \mid w_i w_{i-1} w_{i-2} w_{i+1} w_{i+2} t_{i-1} t_{i-2} t_{i+1} t_{i+2} N^{k-1} N^{k-2} N^{k+1} N^{k+2})$$

5.1.3 The Extension Model

The extension feature value prediction is conditioned on the node information at the node being extended, all information from two nodes to the left and two nodes to the right, and the two leftmost children and the two rightmost children of the current node (these will be redundant if there are less than 4 children at a node).

$$\mathcal{P}(N_e^k \mid context) \approx \mathcal{P}(N_e^k \mid N_w^k N_t^k N_l^k N_c^k N^{k-1} N^{k-2} N^{k+1} N^{k+2} N^{c_1} N^{c_2} N^{c_{-1}} N^{c_{-2}})$$

5.1.4 The Label Model

The label feature value prediction is conditioned on questions about the presence of selected words in the constituent, all information from two nodes to the left and two nodes to the right, and the two leftmost children and the two rightmost children of the current node.

$$\mathcal{P}(N_l^k \mid context) \approx \mathcal{P}(N_l^k \mid Q^k N^{k-1} N^{k-2} N^{k+1} N^{k+2} N^{c_1} N^{c_2} N^{c_{-1}} N^{c_{-2}})$$

5.1.5 The Conjunction Model

The conjunction feature value prediction is conditioned on the node information at the node being extended, all information from two nodes to the left and two nodes to the right, and the two leftmost children and the two rightmost children of the current node (these will be redundant if there are less than 4 children at a node). However, questions about the conjunction bit of adjacent nodes are omitted from the set of candidate questions.

$$\mathcal{P}(N_c^k \mid context) \approx \mathcal{P}(N_c^k \mid N_w^k N_t^k N_l^k N^{-1} N^{-2} N^1 N^2 N^{c_1} N^{c_2} N^{c_{-1}} N^{c_{-2}})$$

5.1.6 The Derivation Model

In initial experiments, the active node selection process was modelled by a uniform ($\mathcal{P}(active) = 1/n$) model. However, different derivations are considered in the parser, at great computational expense, in order to allow better derivations to supersede the leftmost bottom-up derivation when appropriate. When a uniform distribution on the active node selection is assumed, all derivations, better or worse, contribute to the probability of a parse. Int experiments, this characteristic was preventing good derivations, in which the information available suggested the correct parse, from distinguishing themselves from less good derivations, in which the information available was either inconclusive or misleading. The solution to this problem is to model the active node selection, conditioning the prediction on the current node information and the node information available within the five node window.

$$\mathcal{P}(active \mid context) \approx \mathcal{P}(active \mid Q^k N^k N^{k-1} N^{k-2} N^{k+1} N^{k+2})$$

5.1.7 The Parsing Model

The overall model used in SPATTER is defined in terms of the previous approximations.

First, the probability of a parse tree given the sentence is the sum over all derivations of that parse tree:

$$\mathcal{P}(T \mid S) = \sum_d \mathcal{P}(T, d \mid S)$$

The probability of a derivation of a parse tree is the product of the each of the feature value assignments in that derivation and the probability of each active node selection made in that derivation:

$$\mathcal{P}(T, d \mid S) = \prod_{N \in T, j < |d|} \mathcal{P}(active = N \mid context(d_j)) \mathcal{P}(N_x \mid context(d_j))$$

where x ranges over all feature values predicted at a node.

6 Expectation Maximization Training

The most obvious way to estimate a probability distribution from a corpus of examples is to count the occurrences of the phenomena of interest and to estimate the probability of an event to be the frequency of the event divided by the total number of events. However, this is not a very good way to train a model which considers as much contextual information as the models in SPATTER.

One reason why this is true is that there is a strong tendency to overtrain models when using empirical estimates directly. The SPATTER parsing models consider so much contextual information that almost every new sequence of words would introduce new parameters into the model, and the models would be doing no more than memorizing the parse for each sentence. A model trained in this way would perform very poorly on new sentences. The overtraining problem is treated by smoothing the decision trees with a separate *held-out* training set using an expectation-maximization (E-M) algorithm.

However, smoothing issues aside, the SPATTER parsing model certainly cannot be estimated by direct frequency counts because the model contains a *hidden* component: the derivation model. The order in which the treebank parse trees were constructed is not encoded in the treebank, but the SPATTER parser assigns probabilities to specific derivations of a parse tree. The training process must discover which derivations assign higher probability to the treebank parses, and favor those derivations over others. The decision tree which assigns a distribution to the possible *active* nodes directly assigns weights to different derivations. The reestimation procedure which tries to maximize the probability of the parse trees of a corpus is called the *forward-backward* algorithm.

6.1 Forward-Backward Reestimation

The *forward-backward* (F-B) algorithm, which is an instantiation of the E-M algorithm, reestimates the parameters of a model P_i to generate a new model P_{i+1} which is guaranteed to assign a higher probability to the training corpus than the original model, $P_{i+1}(C) \geq P_i(C)$, when certain assumptions about the model hold. For proofs concerning this and other properties of the forward-backward algorithm, see [12].

The intuition behind the F-B algorithm is that when there are multiple paths from an initial state to a goal state, the different paths should contribute to the model according to their relative probabilities. For example, let's say there are two different derivations, D_1 and D_2, of a parse tree of a sentence, T, where $P(T) = P(D_2, T) + P(D_1, T)$, If the goal is to maximize the probability of the parse tree $P(T)$, then it would make sense to try to increase the probabilities assigned to the actions in D_1 and D_2. However, the reestimation procedure is complicated by the constraint that the the probabilities of all actions at a given state must sum to 1. It may be impossible to increase the probabilities of all of the actions in both derivations. Consider the first step in each derivation. If the probability of the first step of D_1 is set to 1, then the probability of the first step of D_2 must be zero, since both derivations begin with the same initial state. The F-B algorithm manages the constraints of probabilistic models while improving the probability of the corpus from one iteration to the next.

6.2 Forward-Backward in SPATTER

To train the SPATTER parser, all legal derivations of a parse tree (according to the derivational window constraint) are computed. Each derivation can be viewed as a path from a common initial state, the words in the sentence, to a common final state, the completed parse tree. These derivations form a lattice of states, since different derivations of the same parse tree inevitably merge. For instance, the state created by tagging the first word in the sentence and then the second is the same state created by tagging the second word and then the first. These two derivations of this state have different probability estimates, but the state can be viewed as one state for future actions, since it represents a single history.

Adjacent states in this state lattice differ by only a single feature value assignment. For a state s, let s_h represent a state which precedes s in the state lattice, and let $f(s_h, s)$ be the feature value assignment which was made to get from state s_h to state s. The probability of a state s is computed by

$$P(s) = \sum_{s_h} P(s_h) P(f(s_h, s) | s_h).$$

The construction of the state lattice and assignment of transition probabilities according to the current model is called the forward pass. The probability of a given state, $P(s)$, is referred to as $\alpha(s)$. The backward probability of a state, referred to as $\beta(s)$, is calculated according to the following recursive formula:

$$\beta(s_h) = \sum_{s} \beta(s) P(f(s_h, s) | s_h)$$

where the backward probability of the goal state is set equal to the forward probability of the goal state,

$\beta(s_goal) = \alpha(s_goal)$. The count associated with a feature value assignment, $f(s_h, s)$, is

$$count(f(s_h, s)) = \frac{\beta(s)\alpha(s_h)P(f(s_h, s)|s_h)}{\alpha(s_goal)}.$$

This value, $count(f(s_h, s))$, is the contribution of the event $f(s_h, s)$ to the distribution which predicts the feature value f given the history h. The reestimate of the probability of a feature value assignment given a history is the ratio of the total count for that feature value assignment and the total count of all feature value assignments given that history:

$$P_new(f|h) \approx \frac{count(f(s_h, s))}{\sum_{s'} count(f(s_h, s'))}.$$

6.3 Decision Trees and the Forward-Backward Algorithm

Each leaf of decision tree represents the distribution of a class of histories. The parameters of these distributions can be updated using the F-B algorithm as described in section 6.2.

In addition to increasing the probability of a corpus, the algorithm assigns weights to the events in a corpus, where the weights represent the relative contribution of the event to the probability of the corpus. This attribute of the F-B algorithm is useful for growing the decision tree models used in SPATTER.

Initially, the models in SPATTER are assumed to be uniform. Accordingly, each event in each derivation contributes equally to the process which selects which questions to ask about the history in order to predict each feature value. However, the uniform model is certainly not a very good model of feature value assignments. And, since some derivations of a parse tree are better than others, the events generated by the better derivations should contribute more to the decision tree-growing process. The decision trees grown using the uniform assumption collectively form a parsing model, M_1. The F-B count for each event in the training corpus using M_1 can be used to grow a new set of decision trees, M_2. The decision trees in M_2 are constructed in a way which gives more weight to the events which contributed most to the probability of the corpus. However, there is no guarantee that M_2 is a better model than M_1. It isn't even guaranteed that the probability of the training corpus according to M_2 is higher than the probability according to M_1. However, based on experimental results, the use of F-B counts in the construction of new decision trees is effective in acquiring a better model of the data.

6.4 Training Algorithm

There is no way of knowing, a priori, which combination of the previously mentioned applications of the forward-backward algorithm will produce the best model. It might be best to grow an initial set of decision trees, perform F-B training on these trees for a few iterations, and then grow new trees. It might be best to grow more than two sets of trees, generating model M_3 from the F-B counts of M_2, and M_4 from the F-B counts of M_3.

What is known is that growing the decision tree models takes almost 2 days on one machine, and so the number of variations which can be attempted are limited. After initial experimentation, the following sequence of training steps proved effective:

1. Grow initial decision trees (M_1) based on uniform models
2. Create M_2 by pruning trees in M_1 to a maximum depth of 10.
3. Grow decision trees (M_3) from F-B counts from M_2.
4. Perform F-B reestimation for leaves of decision trees in M_3.

There is no obvious justification for the decision to prune the trees in step two to a depth of 10, as opposed to a depth of 5 or a depth of 20. If computational resources permit it, the F-B reestimation may also be applied to M_1, with the resulting model compared to the fully trained M_3 model. Also, steps two and three may be applied to M_3 to generate a new model M_5, which may then be trained using F-B.

7 SPATTER Results

SPATTER was trained on 28,000 sentences from the Computer Manuals domain, and tested on 1100 unseen sentences of length 1 - 25 words. SPATTER produced 1 parse for each sentence, and this parse was compared to the treebank parse for that sentence. On this test set, SPATTER produced the correct parse, i.e. a parse which matched the treebank parse exactly, for 38% of the sentences. Ignoring part-of-speech tagging errors, SPATTER produced the correct parse tree for 47% of the sentences. Further, the correct parse tree is present in the top 20 parses produced by the parser for 64% of the sentences.

No other parsers have reported results on exactly matching treebank parses, so SPATTER was also evaluated on the *crossing brackets* measure from [2], which represents the percentage of sentences for which none of the constituents in the parser's analysis violate the constituent boundaries of the treebank parse. The crossing-brackets measure is a very weak measure of parsing accuracy, since it does not verify prepositional phrase attachment or any other decision which is indicated by omitting structure. However, based on analysis of parsing errors, in the current state-of-the-art, increases in the crossing brackets measure appear to correlated with improvements in overall parsing performance. This may not remain true as parsers become more accurate.

The 1100 sentence corpus which SPATTER analyzed was one of the test corpora used in experiments reported in [2]. The parser discussed in [2] uses a P-CFG based on a rule-based grammar developed by a grammarian over the course of 10 years. This P-CFG parser produced parses which passed the crossing brackets test for 69% of the 1100 sentences. SPATTER improves upon this result, passing the crossing brackets test for 78% of the sentences.

References

[1] Baker, J. K., 1975. Stochastic Modeling for Automatic Speech Understanding. In Speech Recognition, edited by Raj Reddy, Academic Press, pp. 521-542.

[2] Black, E., Garside, R., and Leech, G., 1993. *Statistically-driven Computer Grammars of English: The IBM/Lancaster Approach.* Rodopi. Atlanta, Georgia.

[3] Black, E., Lafferty, J., and Roukos, S., 1992. Development and Evaluation of a Broad-Coverage Probabilistic Grammar of English-Language Computer Manuals. In Proceedings of the Association for Computational Linguistics, 1992. Newark, Delaware.

[4] Black, E., Jelinek, F., Lafferty, J., Magerman, D. M., Mercer, R., and Roukos, S., 1992. Towards History-based Grammars: Using Richer Models for Probabilistic Parsing. In Proceedings of the February 1992 DARPA Speech and Natural Language Workshop. Arden House, New York.

[5] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J., 1984. *Classification and Regression Trees.* Wadsworth and Brooks. Pacific Grove, California.

[6] Brown, P. F., Della Pietra, V. J., deSouza, P. V., Lai, J. C., and Mercer, R. L. Class-based n-gram Models of Natural Language. In *Proceedings of the IBM Natural Language ITL,* March, 1990. Paris, France.

[7] Cover, T. M. and Thomas, J. A., 1991. *Elements of Information Theory.* John Wiley and Sons, Inc. New York, New York.

[8] Hindle, D. and Rooth, M. 1990. Structural Ambiguity and Lexical Relations. In Proceedings of the June 1990 DARPA Speech and Natural Language Workshop. Hidden Valley, Pennsylvania.

[9] Jelinek, F. 1985. Self-organizing Language Modeling for Speech Recognition. IBM Report.

[10] Magerman, D. M. and Marcus, M. P. 1991. Pearl: A Probabilistic Chart Parser. In Proceedings of the February 1991 DARPA Speech and Natural Language Workshop. Asilomar, California.

[11] Magerman, D. M. and Weir, C. 1992. Efficiency, Robustness, and Accuracy in Picky Chart Parsing. In Proceedings of the Association for Computational Linguistics, 1992. Newark, Delaware.

[12] Poritz, A. B., 1988. Hidden Markov Models: A Guided Tour. In Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, 1988. New York, New York.

[13] Sharman, R. A., Jelinek, F., and Mercer, R. 1990. Generating a Grammar for Statistical Training. In Proceedings of the June 1990 DARPA Speech and Natural Language Workshop. Hidden Valley, Pennsylvania.