Cubic-time Parsing and Learning Algorithms for Grammatical Bigram Models

Technical Report UCB//CSD-01-1148
Department of Computer Science
University of California, Berkeley
Berkeley, CA 94720

Mark A. Paskin
paskin@cs.berkeley.edu

June 20, 2001

Abstract

This technical report presents a probabilistic model of English grammar that is based upon "grammatical bigrams", i.e., syntactic relationships between pairs of words. Because of its simplicity, the grammatical bigram model admits cubic-time parsing and unsupervised learning algorithms, which are described in detail.

1 Introduction

This paper presents a probabilistic model of English grammar that is based upon "grammatical bigrams", i.e., syntactic relationships between pairs of words. Because of its simplicity, the grammatical bigram model admits cubic-time parsing and unsupervised learning algorithms, which are described in detail.

Like other probabilistic language models, the grammatical bigram model ascribes a probability model to a formal notion of sentence syntax. We begin by describing the dependency grammar formalism used by the grammatical bigram model (Section 2) and then its probability model (Section 3). Then, we present $O(n^3)$-time parsing (Section 4) and learning (Section 5) algorithms.

2 Dependency Grammars

In this section, we outline the simple dependency grammar formalism upon which the grammatical bigram model is based. The formalism described here (which is the same used in [Eis00, Col99]) is impoverished compared to the

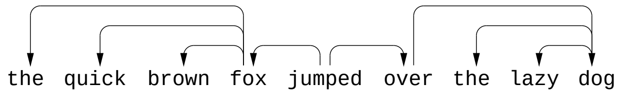

Figure 1: An example parse; arrows are drawn from head words to their dependents. The root word is `jumped`; `brown` is a predependent (adjunct) of `fox`; `dog` is a postdependent (complement) of `over`.

sophisticated models used in Linguistics; we refer the reader to [Hud90] for a comprehensive treatment of English syntax in a dependency framework.

The primary unit of syntactic structure in dependency grammars is the **dependency relationship**: a binary relation between a pair of words in the sentence. In each dependency relationship, one word is designated the **head**, and the other is its **dependent**. (Typically, different types of dependency are distinguished, e.g, subject, complement, adjunct, etc.; in our simple model, no such distinction is made.) Dependents that precede their head are called **predependents**, and dependents that follow their heads are called **postdependents**.

A **dependency parse** consists of a set of dependency relationships that satisfies three properties:

1. Every word except one (the **root**) is dependent to exactly one head.
2. The dependency relationships are acyclic; no word is, through a sequence of dependency relationships, dependent to itself.
3. When drawn as a graph above the sentence, no two dependency relations cross—a property known as **projectivity** or **planarity**.

The first two constraints ensure that when drawn as a graph (see Figure 1 for an example), dependency relationships form a tree; the planarity constraint ensures that a head word and its (direct or indirect) dependents form a contiguous subsequence of the sentence.

The notion of head words originates in phrase structure grammars, where it is often held that every constituent has one head word, and that its head word is the sole (or primary) determinant of how the constituent may combine with other constituents. (We will call this assumption the **head word hypothesis**.) Examples of particular head words in English include:

- the head noun of a noun phrase, e.g., [The ugly **duckling**]$_{\mathrm{NP}}$ swam away;
- the preposition of a prepositional phrase, e.g., He ran [**to** the store]$_{\mathrm{PP}}$; and,
- the finite verb of a verb phrase, e.g., They [**played** pool all day]$_{\mathrm{VP}}$.

The notion of a head word is exactly the same in dependency grammar, but it is formulated in terms of dependency relationships instead of constituents: a head word is the primary determinant of how all of its dependents (direct or indirect) may be syntactically combined with other words to form a sentence. The equivalence arises because the planarity assumption insures that a head word and all of its (direct or indirect) dependents form a contiguous subsequence of the sentence. Thus, there is a natural way to extract constituent structure from a dependency parse: each word with all of its (direct or indirect) dependents forms a constituent. For example, in Figure 1 we see that the constituent headed by the noun `dog` is the noun phrase `the lazy dog`.

Before proceeding, it is worth identifying some flaws in this simple dependency grammar formulation. The assumption that dependency parses are planar is akin to a context-free assumption, and therefore is violated by certain (infrequent) English constructions. Moreover, the lack of an explicit notion of constituency makes certain constructions, e.g., conjunctions, difficult to model. These and other problems are handled in sophisticated dependency grammars by augmenting the theory with further constructs; for the sake of simplicity, we will satisfy ourselves with the simple dependency grammar described above.

In order to formalize our dependency grammar model, we will view sentences as sequences of word tokens drawn from some set of word types. Let $V = \{t_1, t_2, \ldots, t_M\}$ be our vocabulary of M word types. A sentence with n words is therefore represented as a sequence $S = \langle w_1, w_2, \ldots, w_n \rangle$, where each word token w_i is a variable that ranges over V. We use the notation

$$w_{i:j} \triangleq \langle w_i, w_{i+1}, \ldots, w_j \rangle$$

as shorthand for contiguous subsequences of words. And, we use the notation $(i, j) \in L$ to express that w_j is a dependent of w_i in the parse L.

Because it simplifies the structure of our model, we will make the following three assumptions about S and L (without loss of generality):

1. the first word w_1 of S is a special symbol $\textsc{root} \in V$;
2. the root of L is w_1; and,
3. w_1 has only one dependent.

These assumptions are merely syntactic sugar: they allow us to treat all words in the true sentence (i.e., $w_{2:n}$) as dependent to one word. (The true root of the sentence is the sole child of w_1.) We will use the notation $\mathbb{L}^n$ to denote the set of all dependency parses over sentences with n words that satisfy the assumptions above.

3 Grammatical Bigram Probability Model

A probabilistic dependency grammar is a probability distribution $P(S, L)$ where $S = \langle w_1, w_2, \ldots, w_n \rangle$ is a sentence, L is a parse over S, and the words $w_{2:n}$ are random variables ranging over V. (Recall that $w_1 = \textsc{root}$ by assumption.) Of course, S and L exist in high dimensional spaces; therefore, tractable representations of this distribution make use of independence assumptions.

Conventional probabilistic dependency grammar models use the head word hypothesis as an independence assumption; it implies that the lexical model can be safely decomposed into a product over constituents:

$$P(S,L) = \prod_{i=1}^{n} P\left(\langle w_j : (i,j) \in L\rangle \text{ is the dependent sequence} \mid w_i \text{ is the head}\right)$$

For example, the probability of a particular sequence can be governed by a fixed set of probabilistic phrase-structure rules, as in [Col99]; alternatively, the predependent and postdependent subsequences can be modeled separately by Markov chains that are specific to the head word, as in [Eis96].

Consider a much stronger independence assumption: that all the dependents of a head word are independent of one another and their relative order. This is clearly an approximation; in general, there will be strong correlations between the dependents of a head word. More importantly, this assumption prevents the model from representing important argument structure constraints. For example: many words require dependents (for example, prepositions); some verbs can have optional objects, whereas others require or forbid them. However, this assumption relieves the parser of having to maintain internal state for each constituent it constructs, and therefore reduces the computational complexity of parsing and learning.

We can express this independence assumption in the following way: first, we forego modeling the length of the sentence, n, since in parsing applications it is always known; then, we expand $P(S,L \mid n)$ into $P(S \mid L)P(L \mid n)$ and choose $P(L \mid n)$ as uniform; finally, we select

$$P(S \mid L) \quad \triangleq \quad \prod_{(i,j)\in L} P(w_j \text{ is a [pre/post]dependent} \mid w_i \text{ is the head})$$

This distribution factors into a product of terms over syntactically related word pairs; therefore, we call this model the "grammatical bigram" model.

The parameters of the model are

$$\begin{aligned} \gamma_{xy}^{\leftarrow} &\triangleq P(\text{predependent is } t_y \mid \text{head is } t_x) \\ \gamma_{xy}^{\rightarrow} &\triangleq P(\text{postdependent is } t_y \mid \text{head is } t_x) \end{aligned}$$

We can make these parameters explicit by introducing the indicator variable

$$w_i^x \triangleq \begin{cases} 1 & \text{if } w_i = t_x \\ 0 & \text{otherwise} \end{cases}$$

Then we can express $P(S \mid L)$ as

$$P(S \mid L) \quad \triangleq \quad \prod_{\substack{(i,j)\in L \\ j<i}} \prod_{x=1}^{M} \prod_{y=1}^{M} \left[\gamma_{xy}^{\leftarrow}\right]^{w_i^x w_j^y} \prod_{\substack{(i,j)\in L \\ i<j}} \prod_{x=1}^{M} \prod_{y=1}^{M} \left[\gamma_{xy}^{\rightarrow}\right]^{w_i^x w_j^y}$$

4 Efficient parsing

Parsing a given sentence S consists of computing

$$\begin{aligned} L^* &\triangleq \arg\max_{L \in \mathbb{L}^n} P(L \mid S, n) \\ &= \arg\max_{L \in \mathbb{L}^n} P(L, S \mid n) \\ &= \arg\max_{L \in \mathbb{L}^n} P(S \mid L) P(L \mid n) \\ &= \arg\max_{L \in \mathbb{L}^n} P(S \mid L) \end{aligned}$$

since $P(L \mid n)$ is uniform.

Using a result by Yuret [Yur98], it is trivial to show that if $f(n) \triangleq |\mathbb{L}^n|$, then

$$f(n) = \frac{n-1}{2(n-2)+1} \binom{3(n-2)}{n-2}$$

As one would expect, exhaustive search for L^* is intractible.

Fortunately, our grammar model falls into the class of "Bilexical Grammars", for which efficient parsing algorithms have been developed. The parsing algorithm we describe here is derived from the span-based chart-parsing algorithm of [Eis00], and can find L^* in $O(n^3)$ time.

4.1 Spans

A **span** is a portion of a dependency parse that covers a contiguous subsequence of words $w_{i:j}$ with the **span property**: in the complete parse, no word in the interior of the span, i.e., $w_{i+1:j-1}$, has a parent outside the span. Formally, a span over $w_{i:j}$ is a directed, acyclic, planar graph whose nodes are the words $w_{i:j}$ with the property that all words in $w_{i+1:j-1}$ have exactly one parent. A span is **connected** if there is an undirected path between its endwords. A dependency parse is therefore a connected span over the words $w_{1:n}$.

Two adjacent spans that share an endword can be **joined** to form a larger span, so long as at least one of them is connected. (If both were not connected, their joining would *strand* words; i.e., there would be words that were unreachable from the new span's endwords). This longer span will be connected if both of its subspans were connected; if it is not, it can optionally be **closed** by adding an edge between its endwords w_i and w_j.

Spans play a role analogous to that of constituents in phrase structure parsing: every dependency parse has a *unique* span decomposition. Consider a span over the words $w_{i:j}$, and let w_k be the right-most word in $w_{i+1:j-1}$ such that there is an edge between w_j and w_k. (If there is no such word, let $k = i + 1$.) Because of the planarity property, $w_{i:k}$ and $w_{k:j}$ must also be spans. Therefore, the span over $w_{i:j}$ can be composed by joining the subspans over $w_{i:k}$ and $w_{k:j}$ and optionally closing the result, i.e., adding an edge between words w_i and w_j.

By this decomposition, the left subspan must have an edge between its endwords or else consist of two words; spans with this property are called **simple**. When the parser joins spans, it will require that the left span is simple in order to ensure that each span is derived only once. The span decomposition of the parse in Figure 1 is shown in Figure 2 (where a ROOT word has been added). Notice that in each joining, the left input span is simple.

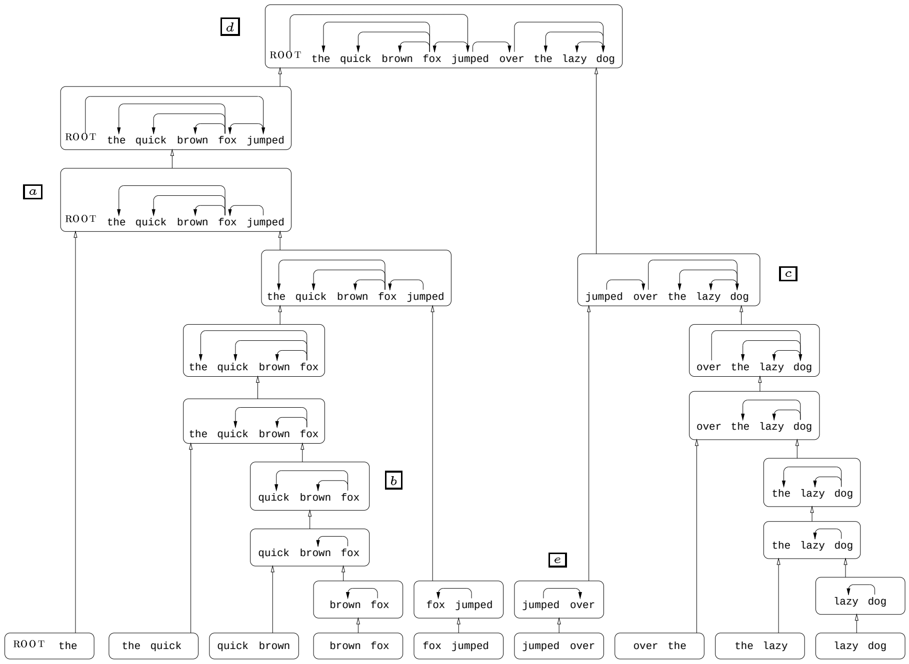

Figure 2: The span decomposition of the parse in Figure 1 (updated with a ROOT word). Each rounded rectangle represents a span; hollow-tipped arrows between spans indicate that one is used to create another by a joining or closing operation.

4.2 Span signatures

As usual, the key to efficient parsing is to use dynamic programming to avoid constructing spans more than once. Doing so requires abstracting away all information about a span that is irrelevant in deciding how it may participate in larger spans. To that end, each span is characterized by a **signature** of the form $\sigma = \langle i, j, b_L, b_R, s \rangle$, where:

- i and j are the indexes of its endwords w_i and w_j (with $i < j$);
- $s = T$ if the span is simple and $s = F$ otherwise; and

- $b_L, b_R \in \{T, F\}$ where $b_L = T$ iff w_i has a parent within the span and $b_R = T$ iff w_j has a parent within the span.

The values of b_L and b_R (when combined with the span property) have implications about the internal structure of a span. We will examine each case, using the spans in Figure 2 as examples:

- If $b_L = b_R = F$, then the span consists of two half-constituents; an example is the span labeled $\boxed{a}$.
- If $b_L = T$ and $b_R = F$, then w_i is a descendant of w_j; the span labelled $\boxed{b}$ is an example.
- If $b_L = F$ and $b_R = T$, then w_j is a descendant of w_i; the span labelled $\boxed{c}$ is an example.
- Finally, if $b_L = b_R = T$, then both w_i and w_j are descendants of a word in the span's interior; the span property implies that that word must be the root of the sentence. Because we are only interested in parses whose root is w_1, we will never derive a span with this signature.

For a sentence of length n, we will call signatures of the form $\langle 1, n, F, T, \cdot \rangle$ **top-level signatures**, since such signatures characterize valid parses. Note that the span labelled $\boxed{d}$ in Figure 2 has a top-level signature.

4.3 Span operators

Our parser will build larger spans out of smaller ones by joining and closing spans. To formalize these notions, we introduce four span operators, called SEED, JOIN, CLOSE-LEFT and CLOSE-RIGHT. The application of an operator to a specific set of arguments is called an **operation**. Because signatures sum up all the characteristics of a span necessary to produce larger spans, parser operations take signatures rather than spans as input, and produce signatures as well. If ω is an operation, we will write $\omega \to \sigma$ to indicate that the operation ω produces a span with signature σ.

The SEED operator (which takes no span signatures as input) creates a new unconnected, simple span over two adjacent words w_i and w_{i+1}:

$$\text{SEED}(i) \to \langle i, i+1, F, F, T \rangle$$

The leaf nodes in the span decomposition of Figure 2 are the results of SEED operations.

The CLOSE-LEFT and CLOSE-RIGHT operators take as input an unconnected span and add an edge between its endwords; CLOSE-LEFT makes the left endword the parent of the right endword by adding an edge (i, j) to L:

$$\text{CLOSE-LEFT}\left(\langle i, j, F, F, \cdot \rangle\right) \to \langle i, j, F, T, T \rangle$$

and CLOSE-RIGHT makes the right endword the parent of the left endword by adding an edge (j, i) to L:

$$\text{CLOSE-RIGHT}\left(\langle i, j, F, F, \cdot\rangle\right) \stackrel{\mathbf{i>1}}{\longrightarrow} \langle i, j, T, F, T\rangle$$

These operators require that neither endword in their input has a parent within the span. (If one or both did, the tree property or the span property would be violated.) Additionally, the CLOSE-RIGHT operation requires that its input span not start the sentence; this prevents w_1 from being chosen as the dependent of another word. In Figure 2, the span labelled $\boxed{b}$ is the result of a CLOSE-RIGHT operation; the span labelled $\boxed{e}$ is the result of a CLOSE-LEFT operation.

Finally, the JOIN operator takes as input two spans that share an endword and joins them:

$$\text{JOIN}\left(\langle i, k, b_L, b, T\rangle, \langle k, j, \neg b, b_R, s\rangle\right) \stackrel{\mathbf{i>1}}{\longrightarrow} \langle i, j, b_L, b_R, F\rangle$$

This definition enforces that:

1. the input spans share an endword;
2. the shared endword has one parent; and,
3. the left input is simple (to ensure no span is constructed more than once—see Section 4.1).

The JOIN rule above only applies when the left input span does not start the sentence; for example, the span labelled $\boxed{c}$ was produced by the rule above. For cases when it does, we define two special JOIN rules that ensure w_1 will have exactly one child:

$$\begin{array}{rcl}\text{JOIN}\left(\langle 1, k, F, T, T\rangle, \langle k, n, F, T, s\rangle\right) & \rightarrow & \langle 1, n, F, T, F\rangle \\ \text{JOIN}\left(\langle 1, 2, F, F, T\rangle, \langle 2, j, T, F, s\rangle\right) & \rightarrow & \langle 1, j, F, F, F\rangle\end{array}$$

The span labelled $\boxed{d}$ was produced by the first of these two rules, and the span labelled $\boxed{a}$ was produced by the second.

4.4 Derivations and dynamic programming

The operator presented in the previous section constitute an algebra over span signatures which we will call the **span signature algebra**. A **derivation** D is an expression in the span signature algebra; like operations, derivations evaluate to span signatures. For example, consider the span labelled $\boxed{b}$ in Figure 2; its derivation is

$$D = \text{CLOSE-RIGHT}(\text{JOIN}(\text{SEED}(3), \text{CLOSE-RIGHT}(\text{SEED}(4)))) \tag{1}$$

As with operations, we will write $D \rightarrow \sigma$ to notate that the derivation D evaluates to a span of signature σ.

As with other algebras, we can represent expressions in the span signature algebra (i.e., derivations) as trees, where the nodes are operations. For example, the tree corresponding to expression (1) would look like that shown in Figure 3. Because our operations are defined in such a way to derive every dependency parse exactly once, there is an isomorphism between dependency parses and their corresponding derivations; thus, D_L will identify the derivation that yields parse L.

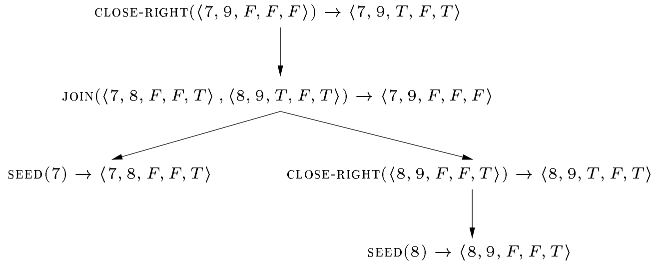

Figure 3: A tree representation of the derivation (1); its structure matches the structure of subtree rooted by span $\boxed{b}$ in Figure 2.

We can associate potentials with parser operations such that the probability of a sentence given a particular parse is equal to the product of the potentials of operations in the parse's derivation. For example, SEED and JOIN operations do not make any edge commitments, so we set

$$\phi(\text{SEED}(\cdot)) \triangleq 1 \qquad \phi(\text{JOIN}(\cdot,\cdot)) \triangleq 1$$

Close operations do make edge commitments; therefore, we set

$$\begin{aligned} \phi(\text{CLOSE-LEFT}(\langle i,j,F,F,\cdot\rangle)) &\triangleq \prod_{x=1}^{M}\prod_{y=1}^{M}\left[\gamma_{\overrightarrow{xy}}\right]^{w_i^x w_j^y} \\ \phi(\text{CLOSE-RIGHT}(\langle i,j,F,F,\cdot\rangle)) &\triangleq \prod_{x=1}^{M}\prod_{y=1}^{M}\left[\gamma_{\overleftarrow{xy}}\right]^{w_j^x w_i^y} \end{aligned}$$

If we define the potential of a derivation D as the product of potentials of the operations in D,

$$\phi(D) \triangleq \prod_{\omega \in D} \phi(\omega) \tag{2}$$

then we see that

$$P(S, L \mid n) = P(S \mid L) P(L \mid n) = \frac{\phi(D_L)}{f(n)}$$

Thus, in order to maximize the left hand side over L, it is sufficient to find the derivation D_L with maximum potential, as defined by equation (2).

The derivations corresponding to spans with a particular signature will have common subexpressions; building this shared structure once is what permits us to explore all possible parses efficiently. Derivations have an **optimal substructure property**: the optimal derivation of a particular signature must consist of an operation over the results of optimal sub-derivations; if it did not, we could substitute a better sub-derivation and achieve a better result. Therefore, for each signature σ, our parser need only record the parse operation $\omega^*(\sigma)$ yielding the most likely derivation with signature σ (and its probability $\pi^*(\sigma)$) in order to reconstruct the most likely derivation of the entire sentence.

Algorithm 1 Chart-parsing

```
Require: ∀σ : π*(σ) = 0
  for i ← 1 to n − 1 do
    ADD(SEED(i))
    if i ≠ 1 then
      ADD(CLOSE-RIGHT(⟨i, i + 1, F, F, T⟩))
    ADD(CLOSE-LEFT(⟨i, i + 1, F, F, T⟩))
  for length ← 2 to n − 1 do
    for i ← 1 to n − length do
      j ← i + length
      for k ← i + 1 to j − 1 do
        for b_L, b, b_R, s ∈ {T, F} do
          σ_L = ⟨i, k, b_L, b, T⟩
          σ_R = ⟨k, j, ¬b, b_R, s⟩
          if JOIN(σ_L, σ_R) is defined then
            ADD(JOIN(σ_L, σ_R))
      ADD(CLOSE-LEFT(⟨i, j, F, F, F⟩))
      if i ≠ 1 then
        ADD(CLOSE-RIGHT(⟨i, j, F, F, F⟩))
  return EXTRACT-OPT-PARSE()

proc ADD(ω)
  σ ← EVALUATE(ω)
  if ω = SEED(i) then
    π ← 1
  else if ω = CLOSE-LEFT(σ_U) or ω = CLOSE-RIGHT(σ_U) then
    π ← φ(ω)π*(σ_U)
  else if ω = JOIN(σ_L, σ_R) then
    π ← φ(ω)π*(σ_L)π*(σ_R)
  if π > π*(σ) then {check for a new best derivation}
    π*(σ) ← π
    ω*(σ) ← ω
```

Algorithm 1 gives the chart-parsing algorithm. The EXTRACT-OPT-PARSE() subprocedure constructs the optimal parse by finding the top-level signature σ

with maximum optimal probability $\pi^*(\sigma)$ and recursively backtracking through the optimal derivation defined by $\omega^*(\cdot)$. When CLOSE operations are encountered, edges are recorded in the parse. This algorithm requires $O(n^3)$ time and $O(n^2)$ space.

5 Training the model

Suppose that we have a labelled i.i.d. data set

$$\mathcal{D}_{\text{labelled}} = \{(S_1, L_1, n_1), (S_2, L_2, n_2), \ldots, (S_N, L_N, n_N)\}$$

where $S_k = (w_{1,k}, w_{2,k}, \ldots, w_{n_k,k})$ and $L_k \in \mathbb{L}^{n_k}$. The maximum likelihood values for our parameters γ can be calculated by maximizing the likelihood $P(\mathcal{D} \mid \gamma)$ with respect to γ subject to some normalization constraints; we must have

$$\sum_{y=1}^{M} \gamma_{xy}^{\rightarrow} = \sum_{y=1}^{M} \gamma_{xy}^{\leftarrow} = 1 \text{ for all } 1 \le x \le M \tag{3}$$

Our likelihood $P(\mathcal{D} \mid \gamma)$ can be written

$$\begin{aligned} P(\mathcal{D} \mid \gamma) &= \prod_{k=1}^{N} P(S_k, L_k \mid n_k) = \prod_{k=1}^{N} \frac{1}{f(n_k)} P(S_k \mid L_k) \\ &= \prod_{k=1}^{N} \frac{1}{f(n_k)} \prod_{\substack{(i,j)\in L_k \\ j<i}} \prod_{x=1}^{M} \prod_{y=1}^{M} \left[\gamma_{xy}^{\leftarrow}\right]^{w_{i,k}^{x} w_{j,k}^{y}} \prod_{\substack{(i,j)\in L_k \\ i<j}} \prod_{x=1}^{M} \prod_{y=1}^{M} \left[\gamma_{xy}^{\rightarrow}\right]^{w_{i,k}^{x} w_{j,k}^{y}} \end{aligned}$$

To maximize $P(\mathcal{D} \mid \gamma)$ with respect to γ, we can equivalently maximize its logarithm:

$$\begin{aligned} \log P(\mathcal{D} \mid \gamma) = \sum_{k=1}^{N} -\log f(n_k) &+ \sum_{\substack{(i,j)\in L_k \\ j<i}} \sum_{x=1}^{M} \sum_{y=1}^{M} w_{i,k}^{x} w_{j,k}^{y} \log \gamma_{xy}^{\leftarrow} \\ &+ \sum_{\substack{(i,j)\in L_k \\ i<j}} \sum_{x=1}^{M} \sum_{y=1}^{M} w_{i,k}^{x} w_{j,k}^{y} \log \gamma_{xy}^{\rightarrow} \end{aligned}$$

Let us introduce the following indicator variables:

$$e_{ij}^{k} \triangleq \begin{cases} 1 & \text{if } (i,j) \in L_k \\ 0 & \text{otherwise} \end{cases}$$

Then, we must maximize

$$\log P(\mathcal{D} \mid \gamma) = \sum_{k=1}^{N} - \log f(n_k) + \sum_{1 \le j < i \le n_k} e_{ij}^k \sum_{x=1}^{M} \sum_{y=1}^{M} w_{i,k}^x w_{j,k}^y \log \gamma_{xy}^{\leftarrow}$$
$$+ \sum_{1 \le i < j \le n_k} e_{ij}^k \sum_{x=1}^{M} \sum_{y=1}^{M} w_{i,k}^x w_{j,k}^y \log \gamma_{xy}^{\rightarrow}$$

subject to (3). Forming the Lagrangian and setting partial derivatives equal to zero leads to the following maximum likelihood parameter estimates:[1]

$$\widehat{\gamma}_{xy}^{\rightarrow} = \frac{\sum_{k=1}^{N} \sum_{1 \le i < j \le n_k} e_{ij}^k w_{i,k}^x w_{j,k}^y}{\sum_{k=1}^{N} \sum_{1 \le j < i \le n} e_{ij}^k w_{i,k}^x} \tag{4}$$

$$\widehat{\gamma}_{xy}^{\leftarrow} = \frac{\sum_{k=1}^{N} \sum_{1 \le j < i \le n_k} e_{ij}^k w_{i,k}^x w_{j,k}^y}{\sum_{k=1}^{N} \sum_{1 \le i < j \le n_k} e_{ij}^k w_{i,k}^x} \tag{5}$$

In the unsupervised acquisition problem, our data set has no parses:

$$\mathcal{D}_{\text{unlabelled}} = \{(S_1, n_1), (S_2, n_2), \ldots, (S_N, n_N)\}$$

Our approach will be to introduce a hidden variable L_k for each S_k and to employ the EM algorithm to learn (locally) optimal values of the parameters γ. As we have shown above, the e_{ij}^k are sufficient statistics for our model; given them, we can compute maximum likelihood estimates of the γ parameters.

Thus, our EM algorithm will repeatedly perform the following two steps:

- **E-step** (state estimation): compute the conditional expectation of the sufficient statistics e_{ij}^k (denoted ε_{ij}^k) given the training data and the current value of the parameters;

- **M-step** (maximization of the expected complete likelihood): update the parameters to maximize the likelihood of the data given the current values of the ε_{ij}^k. (This is accomplished simply by substituting the computed ε_{ij}^k values for e_{ij}^k into equations (4) and (5).)

In describing the computation of the expected sufficient statistics ε_{ij}^k, we first define a useful quantity called a conditional signature probability; then we give an efficient dynamic programming algorithm to calculate these quantities; finally, we relate the expected sufficient statistics to conditional signature probabilities.

[1] In reality, the maximum-likelihood estimates are not very useful by themselves because of the zero-frequency problem: many dependency relationships that are possible (yet infrequent) will not be present in the test set and will therefore be given zero probability in the optimization defined above. A standard *ad hoc* smoothing solution seems to work quite well: a Katz backoff model [Kat87] that uses Witten-Bell smoothing (i.e, Model C of [WB91]).

5.1 Inside, outside, and conditional signature probabilities

First, let us define some useful quantities:

- The **inside probability** $\beta(\sigma)$ of a signature σ is the sum of the potentials of all derivations D evaluating to σ:

$$\beta(\sigma) = \sum_{D \to \sigma} \phi(D)$$

- Let $\Delta^{\text{out}}(\sigma, S)$ be the set of derivations of S evaluating to a top-level signature *but which are missing a subtree evaluating to σ*. The **outside probability** $\alpha(\sigma)$ of σ is the sum of the potentials of all these derivations:

$$\alpha(\sigma) = \sum_{D \in \Delta^{\text{out}}(\sigma,S)} \phi(D)$$

- The **conditional signature probability** $P(\sigma \in D \,|\, S)$ of σ is the probability that an operation yielding a span with signature σ is present in the derivation given the sentence.

(Note that the inside and outside "probabilities" are actually *potential* functions, and not probability mass functions; however, we will follow convention in this abuse of terminology.) Figure 5.1 gives a picture of how the inside and outside probabilities relate to a generic derivation.

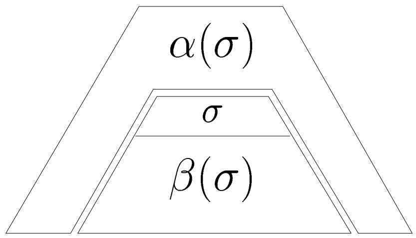

Figure 4: An iconic depiction of the inside and outside probabilities.

From the definitions of the inside and outside probabilities, we see that the joint probability of a sentence S and a span with signature σ appearing in its derivation is:

$$\begin{aligned} P(\sigma \in D, S \,|\, n) &= \sum_{L:\sigma \in D_L} P(L, S \,|\, n) = \frac{1}{f(n)} \sum_{L:\sigma \in D_L} P(S \,|\, L) \\ &= \frac{1}{f(n)} \sum_{D:\sigma \in D} \phi(D) = \frac{\alpha(\sigma)\beta(\sigma)}{f(n)} \end{aligned}$$

Furthermore, we can compute the probability of a sentence S as

$$P(S \mid n) = \sum_{L \in \mathbb{L}^n} P(S, L \mid n) = \frac{1}{f(n)} \sum_{L \in \mathbb{L}^n} P(S \mid L) = \frac{1}{f(n)} \sum_{\substack{\text{top-level} \\ \text{signatures } \sigma}} \beta(\sigma)$$

Thus, if we can compute the inside and outside probabilities, we can compute the conditional signature probability

$$P(\sigma \in D \mid S) \quad = \quad \frac{P(\sigma \in D, S \mid n)}{P(S \mid n)} = \frac{\alpha(\sigma)\beta(\sigma)}{\sum_{s \in \{T,F\}} \beta(\langle 1, n, F, T, s \rangle)} \tag{6}$$

of any σ.

5.2 An efficient Inside-Outside algorithm

We can derive a dynamic programming algorithm to compute the conditional signature probabilities in analogy to the Inside-Outside algorithm for PCFG state estimation [LY90]. Let us start by expressing the inside probabilities via a recurrence:

$$\beta(\sigma) = \begin{cases} 1 & \text{if } \sigma = \langle i, j, F, F, T \rangle \\ \displaystyle\sum_{\substack{\sigma_U \text{ such that} \\ \text{CLOSE-LEFT}(\sigma_U) \to \sigma \\ \text{is defined}}} \beta(\sigma_U)\phi(\text{CLOSE-LEFT}(\sigma_U)) & \text{if } \sigma = \langle i, j, F, T, T \rangle \\ \displaystyle\sum_{\substack{\sigma_U \text{ such that} \\ \text{CLOSE-RIGHT}(\sigma_U) \to \sigma \\ \text{is defined}}} \beta(\sigma_U)\phi(\text{CLOSE-RIGHT}(\sigma_U)) & \text{if } \sigma = \langle i, j, T, F, T \rangle \\ \displaystyle\sum_{\substack{\sigma_L, \sigma_R \text{ such that} \\ \text{JOIN}(\sigma, \sigma_R) \to \sigma \\ \text{is defined}}} \beta(\sigma_L)\beta(\sigma_R)\phi(\text{JOIN}(\sigma_L, \sigma_R)) & \text{if } \sigma = \langle i, j, \cdot, \cdot, F \rangle \end{cases}$$

Because the inside probabilities are defined in terms of a bottom-up recurrence, our chart parsing algorithm can be modified in order to compute them at minimal extra cost. We introduce variables $\beta(\sigma)$ which store the inside probabilities (and are initialized to zero) and update the ADD(ω) procedure as shown in Algorithm 2. This change does not affect the time or space complexity of the parsing algorithm.

Using the inside probabilities, the outside probabilities can be computed by top-down recursion. Consider a fixed operation ω (yielding a signature σ) and its role in a fixed derivation D. Depending upon the value of σ, ω might have served as the left or right child of a JOIN operation, and it might have served as the child of a CLOSE-LEFT or CLOSE-RIGHT operation. Since only one of these cases occurs in any fixed D, the outside probability of σ can be computed as the sum of the outside probabilities computed when these cases are considered

Algorithm 2 Reimplementation of ADD that computes inside probabilities

proc ADD(ω)
  $\sigma \leftarrow$ EVALUATE(ω)
  if $\omega =$ SEED(i) **then**
    $\pi \leftarrow 1$
    $\beta(\sigma) \leftarrow 1$
  else if $\omega =$ CLOSE-LEFT(σ) or $\omega =$ CLOSE-RIGHT(σ) **then**
    $\pi \leftarrow P(\omega)\pi^*(\sigma)$
    $\beta(\sigma) \stackrel{+}{\leftarrow} P(\omega)\beta(\sigma)$ {$a \stackrel{+}{\leftarrow} b$ is shorthand for $a \leftarrow a + b$}
  else if $\omega =$ JOIN(σ_L, σ_R) **then**
    $\pi \leftarrow P(\omega)\pi^*(\sigma_L)\pi^*(\sigma_R)$
    $\beta(\sigma) \stackrel{+}{\leftarrow} P(\omega)\beta(\sigma_L)\beta(\sigma_R)$
  if $\pi > \pi^*(\sigma)$ **then** {check for a new best derivation}
    $\pi^*(\sigma) \leftarrow \pi$
    $\omega^*(\sigma) \leftarrow \omega$
  return σ

exclusively:

$$\begin{aligned}
\alpha(\sigma) &= \sum_{\substack{D \in \Delta^{\text{out}}(\sigma, S) \\ \text{JOIN}(\sigma, \cdot) \in D}} \phi(D) + \sum_{\substack{D \in \Delta^{\text{out}}(\sigma, S) \\ \text{JOIN}(\cdot, \sigma) \in D}} \phi(D) + \sum_{\substack{D \in \Delta^{\text{out}}(\sigma, S) \\ \text{CLOSE-LEFT}(\sigma) \in D}} \phi(D) + \sum_{\substack{D \in \Delta^{\text{out}}(\sigma, S) \\ \text{CLOSE-RIGHT}(\sigma) \in D}} \phi(D) \\
&= \sum_{\substack{\sigma_R \text{ such that} \\ \text{JOIN}(\sigma, \sigma_R) \to \sigma_J \\ \text{is defined}}} \alpha(\sigma_J)\beta(\sigma_R)\phi(\text{JOIN}(\sigma, \sigma_R)) \\
&\quad + \sum_{\substack{\sigma_L \text{ such that} \\ \text{JOIN}(\sigma_L, \sigma) \to \sigma_J \\ \text{is defined}}} \alpha(\sigma_J)\beta(\sigma_L)\phi(\text{JOIN}(\sigma_L, \sigma)) \\
&\quad + \sum_{\substack{\sigma_C \text{ such that} \\ \text{CLOSE-LEFT}(\sigma) \to \sigma_C \\ \text{is defined}}} \alpha(\sigma_C)\phi(\text{CLOSE-LEFT}(\sigma)) \\
&\quad + \sum_{\substack{\sigma_C \text{ such that} \\ \text{CLOSE-RIGHT}(\sigma) \to \sigma_C \\ \text{is defined}}} \alpha(\sigma_C)\phi(\text{CLOSE-RIGHT}(\sigma))
\end{aligned}$$

Thus, we have a top-down recursion, because the outside probability of each signature can be computed using the outside probabilities of signatures that are "larger". Our base case for this recursion is that for all top-level signatures σ, $\alpha(\sigma) = 1$. Algorithm 3 is a top-down dynamic programming algorithm that computes the outside probabilities in $O(n^3)$ time and $O(n^2)$ space.

5.3 Calculating ε_{ij}^k with conditional signature probabilities

The conditional probability of an edge (i, j) being in the parse given the sentence is equal to the conditional probability of a CLOSE-LEFT($\langle i, j, F, F, \cdot \rangle$) operation

Algorithm 3 Outside Algorithm

Require: $\alpha(\sigma) = 0$ for all σ
  for $s \in \{T, F\}$ **do**
    $\alpha(\langle 1, n, F, T, s\rangle) \leftarrow 1$
  for $length \leftarrow n - 1$ down to 1 **do**
    for $i \leftarrow 1$ to $n - length$ **do**
      $j \leftarrow i + length$
      $s \leftarrow (length = 1)$
      $\alpha(\langle i, j, F, F, s\rangle) \overset{+}{\leftarrow} \alpha(\langle i, j, F, T, T\rangle) \cdot \phi(\text{CLOSE-LEFT}(\langle i, j, F, F, s\rangle))$
      $\alpha(\langle i, j, F, F, s\rangle) \overset{+}{\leftarrow} \alpha(\langle i, j, T, F, T\rangle) \cdot \phi(\text{CLOSE-RIGHT}(\langle i, j, F, F, s\rangle))$
      if $length > 1$ **then**
        for $b_L, b_R, s \in \{T, F\}$ **do**
          $\sigma \leftarrow \langle i, j, b_L, b_R, s\rangle$
          for $k \leftarrow i + 1$ to $j - 1$ **do**
            for $b, s_R \in \{T, F\}$ **do**
              $\sigma_L \leftarrow \langle i, k, b_L, b, T\rangle$
              $\sigma_R \leftarrow \langle k, j, \neg b, b_R, s_R\rangle$
              if σ is a valid signature and $\text{JOIN}(\sigma_L, \sigma_R)$ is defined **then**
                $\alpha(\sigma_L) \overset{+}{\leftarrow} \beta(\sigma_R) \cdot \alpha(\sigma) \cdot \phi(\text{JOIN}(\sigma_L, \sigma_R))$
                $\alpha(\sigma_R) \overset{+}{\leftarrow} \beta(\sigma_L) \cdot \alpha(\sigma) \cdot \phi(\text{JOIN}(\sigma_L, \sigma_R))$

appearing in the derivation (if $i < j$) or a $\text{CLOSE-RIGHT}(\langle j, i, F, F, \cdot\rangle)$ operation appearing in the derivation (if $j < i$). However, these operations are uniquely identified by their output signatures, i.e.,

$$\begin{aligned} P(\text{CLOSE-LEFT}(\langle i, j, F, F, \cdot\rangle) \in D \mid S) &= P(\langle i, j, F, T, T\rangle \in D \mid S) \\ P(\text{CLOSE-RIGHT}(\langle i, j, F, F, \cdot\rangle) \in D \mid S) &= P(\langle i, j, T, F, T\rangle \in D \mid S) \end{aligned}$$

Thus, we can compute ε_{ij}^k as:

$$\varepsilon_{ij}^k \triangleq E[e_{ij}^k \mid S] = P(e_{ij}^k = 1 \mid S) = \begin{cases} P(\langle i, j, F, T, T\rangle \in D \mid S) & \text{if } i < j \\ P(\langle i, j, T, F, T\rangle \in D \mid S) & \text{if } j < i \end{cases} \quad (7)$$

Therefore, the E-step proceeds as follows:

1. Compute the inside probabilities.
2. Compute the outside probabilities using the inside probabilities.
3. Compute the ε_{ij}^k using equations (7) and (6) and the inside and outside probabilities.

This computation requires $O(n^3)$ time and $O(n^2)$ space.

References

[Col99] Michael Collins. *Head-driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, Philadelphia, Pennsylvania, 1999.

[Eis96] Jason M. Eisner. An empirical comparison of probability models for dependency grammars. Technical Report ICRS-96-11, CIS Department, University of Pennsylvania, 220 S. 33$^{\text{rd}}$ St. Philadelphia, PA 19104–6389, 1996.

[Eis00] Jason Eisner. Bilexical grammars and their cubic-time parsing algorithms. In Harry Bunt and Anton Nijholt, editors, *Advances in Probabilistic and Other Parsing Technologies*, chapter 1. Kluwer Academic Publishers, October 2000.

[Hud90] Richard A. Hudson. *English Word Grammar*. B. Blackwell, Oxford, UK, 1990.

[Kat87] S. M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Trans. Acoustics, Speech, and Signal Processing*, ASSP-35(3):400, 1987.

[LY90] K. Lari and S. J. Young. The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

[WB91] Ian H. Witten and Timothy C. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37:1085–1094, 1991.

[Yur98] Deniz Yuret. *Discovery of Linguistic Relations Using Lexical Attraction*. PhD thesis, Massachusetts Institute of Technology, May 1998.