Hiding a Semantic Class Hierarchy in a Markov Model

Steven Abney and Marc Light

August 1, 1998

Abstract

This paper introduces a new model of selectional preference induction. Unlike previous approaches we provide a stochastic generation model for the words that appear as arguments of a predicate. More specifically, we define a hidden Markov model with the general shape of our semantic class hierarchy and we use an EM-like algorithm for parameter estimation. This model enables us to handle word sense ambiguity in the input in a principled fashion. In addition, it can be used to estimate a number of distributions such as $Pr(word|predicate)$, $Pr(word|semantic\text{-}class)$, $Pr(word\text{-}sense|word, predicate)$. These distributions can be used directly to help solve ambiguity resolution problems such as word sense disambiguation and syntactic structure disambiguation. One can also derive a traditional list of semantic classes that capture the selectional preferences of a predicate for a complement. In addition, the $Pr(word|predicate)$ distribution can be seen as a very specific language model, i.e. , a language model for the head of the argument of the predicate. This work is of general interest because it weds two important entities in natural language processing systems: hierarchical class structures and stochastic generation models.

1 Introduction

We describe here an approach to inducing selectional preferences from text corpora. In the traditional view, a predicate constrains its arguments by selecting for particular semantic classes, or *concepts*. **Selectional restriction** of the traditional sort can be characterized as a relation $\rho(v, r, c)$ over predicates v, syntactic roles r, and argument concepts c. Individual instances (v, r, c) are **selectional tuples**. Examples are given in table 1.

Predicate	Role	Argument Class
splatter	*subj*	CAUSAL-AGENT
splatter	*obj*	FLUID
splatter	*on*	SURFACE
tasty	*subj*	EDIBLE-ITEM
shrug	*subj*	PERSON
shrug	*obj*	SHOULDER

Table 1: Selectional tuples

Of more interest to computational linguistics is **selectional preference**, a continuous-valued generalization of selectional restriction. Selectional preference is a mapping $\sigma : (v, r, c) \mapsto a$ that maps each tuple (v, r, c) to a real number a, the **degree of preference** of v for c with respect to role r. Positive degrees of preference are intended to correlate with intuitive judgments of "plausibility" or "typicality", and negative judgments are intended to correlate with intuitive judgments of "implausibility". But there is no extant proposal on how to quantify such intuitions. The only obvious course, designing a psycholinguistic experiment, would entail a prohibitive effort for a large sample of tuples, and would likely have large variances that limit its usefulness for evaluating automatic preference estimation techniques.

For this reason, we have chosen to characterize selectional preference as a side-effect of a more readily evaluable model, namely, a stochastic model for generating what we will call **co-occurence tuples**: triples (v, r, n) for v a predicate, r a syntactic role, and n the headword of the argument filling the role r with respect to v. An example of a co-occurence tuple is (*splatter*, obj, *water*). Co-occurence tuples can be obtained from text corpora, and can be used to make inferences about the probability of selectional tuples. For example, the co-occurence tuple (*splatter*, obj, *water*) may be taken as evidence for the selectional tuple (*splatter*, obj, FLUID).

For this study, we have used the British National Corpus (100M words), from which we have extracted co-occurence tuples using the Cass parser [1]. By way of illustration, table 2 shows the values of n in tuples (*eat*, obj, n) along with their frequencies in the corpus. In this data, and throughout the paper, any inflectional morphology has been eliminated from both predicate and argument. That is, the values of v and n are properly speaking **lemmas**—words in uninflected base form.

On the assumption that the selectional tuple (*eat, obj,* EDIBLE-ITEM) is

meat	45	bucket	1	ice	2
tape	1	investment	1	soup	2
proportion	2	kitchen	1	fry	4
root	4	salad	2	top	1
bread	14	feast	1	scrap	2
majority	2	sauce	1	sugar	1
principle	1	food	77	hole	2
roll	4	pack	1	bag	2
race	1	mouthful	3	dinner	11
sheep	1	salt	1	meal	46
trout	2	pasta	1	slice	7
dish	2	spaghetti	6	chicken	5
stick	1	egg	18	average	1
sandwich	13	yogurt	1	mustard	1
breakfast	30	garlic	1		

Table 2: Objects of *eat* in the BNC

responsible for these observed co-occurence tuples, one would expect to get arguments that are edible. As is exemplified by *investment, average, tape*, and *race*, this is not always the case. This noise is sometimes due to tagging or parsing errors, and sometimes due to metaphorical uses of *eat*. Note that the "good" examples such as *food* and *meal* are much greater in number and frequency.

Another factor that complicates the inference of selectional tuples from co-occurence tuples is word sense ambiguity. The word *bread* in table 2 provides an example. *Bread* can be used to refer to a food, e.g., *the multigrain bread in Germany is wonderful*, but it can also refer to money, e.g., *I could really use the bread since my car just broke down*. For this reason, it is not immediately clear which concepts the 14 tokens of *bread* provide evidence for. The model we propose handles this sort of uncertainty in a natural way; we return to the question of word sense ambiguity shortly.

Our model generates co-occurence tuples as follows. The probability $p(v, r, n)$ of a co-occurence tuple can be expressed as $p(v, r)p(n|v, r)$. Our central concern is the conditional probability $p(n|v, r)$. We associate a hidden Markov model (HMM) with each pair (v, r), in order to characterize the distribution $p(n|v, r)$. We also construct an HMM to characterize the distribution of nouns in the corpus as a whole. The latter HMM we call the **background model**, and the HMM's associated with particular (v, r) pairs we

call **foreground models**.

The states and transitions of the HMM's are identified with the nodes and arcs of a semantic class hierarchy. We do not attempt to induce a hierarchy, but take one as given. The nodes of the hierarchy represent semantic classes (concepts), and the arcs represent hyponymy (that is, the "is-a" relation). Some concepts are expressible as words: these concepts are **word senses**. (We assume for the moment that all and only the terminal nodes of the hierarchy are word senses, though we later relax this restriction.) A sense may be expressible by multiple words (synonyms), and, conversely, a single word may be an expression of more than one sense (word sense ambiguity).

The only constraint we place on the shape of the hierarchy is that it have a single root. The shape of the hierarchy is otherwise unrestricted: it may include multiple inheritance, non-partioning sub-classes, etc.

A "run" of one of our HMM's begins at the root of the semantic hierarchy. A child concept is chosen in accordance with the HMM's transition probabilities. This is done repeatedly until a terminal node (word sense) c is reached, at which point a word w is emitted in accordance with the probability of expressing sense c as word w. Hence, each HMM "run" can be identified with a path through the hierarchy from the root to a word sense, plus the word that was generated from the word sense. Hence, also, every observation sequence generated by our HMM's consists of a single noun: each run leads to a final state, at which point exactly one word is emitted.

All HMM's have the same structure, determined by the semantic hierarchy. Where they differ is in the values of the associated parameters: state-to-state transition probabilities and state-to-word emission probabilities. To estimate parameters, we require a training sample of observation sequences. Since each observation sequence consists of a single word, a training sample is simply a collection of word tokens. For a foreground model, the training sample consists of the nouns filling the associated "slot" (v, r)—that is, a token of the noun n is included in the training sample for each token of the tuple (v, r, n) that occurs in the corpus. For the background model, the training sample consists of all noun tokens in the corpus. HMM parameter values are then estimated using a variant of the forward-backward training algorithm [**?**].

This approach permits us to address both word sense disambiguation and selectional preference. An ambiguous word is one that could have been generated by means of more than one state sequence. For a given ambiguous word n appearing in a slot (v, r), we can readily compute the posterior

probability that word sense c was used to generate n, according to the (v, r) foreground model. We can disambiguate by choosing the word sense with maximum posterior probability, or we can use the probabilities in a more sophisticated model that uses more contextual information than just the slot that the word appears in.

Selectional preference can be characterized as the difference between the foreground and background models. We use the generation model to characterize the "prominence" of a state. We have considered several definitions of prominence. The most satisfactory seems to be the log probability of visiting the state while generating a random word. If state c is significantly more prominent in the (v, r) foreground model than it is in the background model, we take this as evidence that v selects for c in role r; and v selects *against* c if c is significantly *less* prominent in the foreground model than in the background model.

Another way of thinking about selectional preference is as a distribution over words. For example, the selectional preference of the verb *eat* for its direct object would be expressed by high probabilities for words like *breakfast, meat,* and *bread* and low probabilities for words like *thought, computer,* and *break*. This conception of selectional preference is related to language modeling in speech recognition. In fact, the selectional preference of a predicate-role pair can be thought of as a very specific language model. Thus, the selectional preference of *eat* for its direct object would be a language model for the nouns that can appear as the head of the direct object of *eat*. This way of thinking about selectional preferences is useful because it points to possible applications in speech recognition and because it allows us to evaluate our selectional preferences using cross-entropy against an empirically derived distribution.

In sum, we characterize selection and word sense disambiguation via a collection of HMM's, one for each (v, r) pair. The HMM's have a common state-graph that is determined by a semantic hierarchy. For induction of selectional preferences and disambiguation models, we require a semantic hierarchy and a corpus of co-occurence tuples. The input-output mapping of our induction algorithm are illustrated in Figure 1.[1]

[1] Glenn Carroll provided us with the initial idea and sketch of this figure.

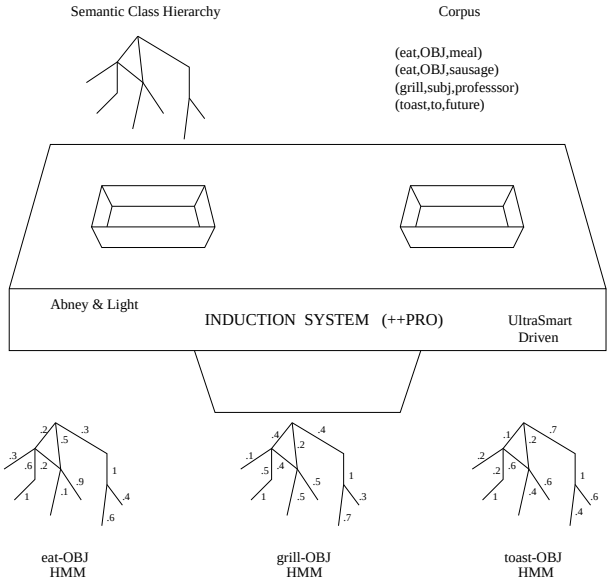

Figure 1: Input and output of induction system

1.1 Related work

There have been a number of attempts to derive selectional preferences using as input parsed corpora and a semantic class hierarchy. One of the first such approaches was [5]. Our work is closely related to Resnik's and it provided a starting point for us. He uses the same input illustrated in Figure 1 but the system provides only a distribution over classes conditioned on a predicate-role pair: $p(c|v,r)$. He estimates $p(c|v,r)$ as $f(v,r,c)/\sum_{c'} f(v,r,c')$, where $f(v,r,c)$ is in turn approximated by allocating the frequency of the co-occurence tuple (v,r,n) among the classes $C(n)$ to which the senses of n belong.

For example, suppose the word *bread* has two senses, BREAD and MONEY. Suppose further that BREAD is a hyponym of BAKED-GOODS, FOOD, ARTIFACT, and TOP, and MONEY is a hyponym solely of TOP. Then $C(bread)$ is {BREAD, BAKED-GOODS, FOOD, ARTIFACT, TOP, MONEY}. Tokens of *bread* are taken as ambiguous evidence for all concepts in $C(bread)$; the weight of evidence is divided uniformly across $C(bread)$. Hence each token of (*eat*, obj, *bread*) counts as 1/6 of a token of (*eat*, obj, BREAD), 1/6 of a token of (*eat*, obj, BAKED-GOODS), and so on.

Resnik is not very explicit about how the probability $p(c|v,r)$ is to be

interpreted, but the apparent intent is that one generates a noun to fill the slot (v, r) by first choosing a concept c according to $p(c|v, r)$ and then choosing a noun according to an (unspecified) distribution $p(n|c)$. This model of generation (assuming it is in fact the one intended by Resnik) differs from ours in that nouns are not generated by a walk down the hierarchy, but rather a noun is generated in one step from a class. There is no necessary connection between the distribution of nouns generated from a class c and a class c', even if c' is the sole child of c and c the sole parent of c'. (Though of course Resnik's estimation technique would guarantee the same distribution for c and c' in such a case.)

[6] modifies Resnik's method by apportioning counts $f(v, r, n)$ among the *paths* through the semantic hierarchy from the root node to a sense of n. The count for noun n is first divided uniformly among its senses, then the count for a sense s is divided uniformly among the paths leading from the root to s. The count for an arbitrary concept c is the sum of the counts of paths on which c lies. The intended generation model is apparently one in which one chooses a concept, then chooses a path leading downward from the concept to a word sense, then chooses a word that expresses the word sense. This is not explicitly stated, however, and no attempt is made to estimate $p(path|c)$ or $p(w|path)$.

Resnik uses $p(c|v, r)$ to quantify selectional preference by comparing it to $p(c)$, the marginal probability of class c appearing as an argument. He measures the difference between these distributions as their relative entropy (D):

$$D(p(c|v, r)||p(c)) = \sum_c p(c|v, r) \log \frac{p(c|v, r)}{p(c)}.$$

The total amount of "selection" that a predicate v imposes on the filler of role r is quantified as $D(p(c|v, r)||p(c))$. The selectional preference of v for c in role r is quantified as the contribution of the c to the total amount of selection:

$$\mathit{selpref}(v, r, c) = \frac{p(c|v, r) \log \frac{p(c|v,r)}{p(c)}}{D(p(c'|v, r)||p(c'))}$$

The class or classes produced as the output for the predicate are those with the highest *selpref* value.

In our terms, Resnik identifies the prominence of concept c with $\log p(c|v, r)$, yielding a selectional strength of $\log p(c|v, r) - \log p(c)$, which Resnik then scales by $p(c|v, r)/D(p(c'|v, r)||p(c'))$ for purposes of comparing selectional strengths across predicates and roles.

Other work on the induction of selectional preferences includes that of Abe and Li. They characterize the selectional restriction of a predicate with a horizontal cut through a semantic hierarchy, and use Rissanen's principle of Minimum Description Length (MDL) to choose a cut that optimally balances simplicity and descriptive adequacy. More specifically, a cut is a set of concepts that partition the set of nouns belonging to the hierarchy. A cut is deemed simpler if it cuts the hierarchy at a higher place (i.e., the cut contains fewer concepts), and descriptive adequacy is measured by comparing the actual distribution of nouns filling a slot (v, r) to the closest approximation one can obtain by estimating $p(n|c)$ for only the concepts c in the cut.

Again, the intended stochastic generation model is not clear. One simple interpretation is that one generates a noun to fill the slot (v, r) by first choosing a concept c from the cut according to the weights along the cut and then choosing a noun according to an (unspecified) distribution $p(n|c)$.

The primary difference between our approach and previous work is that our approach is based on a stochastic model for generating the same sort of objects as make up the input corpus, namely, co-occurence tuples. In previous models, the interpretation of expressions such as $p(c|v, r)$ is obscure, and without clarity about what stochastic process is producing the data, it is impossible to gauge how well probabilities are being estimated. Adopting an explicit stochastic process permits us to replace Resnik's and Ribas' ad hoc apportionment of evidence with motivated inferences from incomplete data using the Expectation-Maximization (EM) algorithm.

Having a complete generation model also permits us to do word sense disambiguation in a motivated way, something the Resnik model does not support. The Viterbi algorithm can be used to select the most-probable path through the hierarchy, for example, or an adaptation can be used to find the word sense with the highest posterior probability given word w in the context (v, r): $p(\textit{word-sense}|(v, r))$.

Having an explicit stochastic model also permits us to embed the selectional model in other applications. For example, selectional preference information in this form can be readily integrated with probabilistic information from other aspects of context to create more sophisticated word sense disambiguation models. Or prepositional phrase attachment can be performed by comparing the respective probabilities of generating the nominal head of the PP given that the PP occurs as an argument of the respective predicates of the attachment sites: if one was trying to disambiguate the attachment of *on the cheek* in *she kisses the man on the cheek*, one could compare $Pr(\textit{cheek}|\textit{kiss-on})$ and $Pr(\textit{cheek}|\textit{man-on})$. One could also use the

$Pr(word|p)$ values as part of a language model; this would allow the language model to make use of selectional preferences and semantic classes for smoothing.

In the following section we will describe our stochastic model and its training. Then we will discuss sample selectional preferences and provide an evaluation of the system based on the cross entropy of the models' distributions and different empirical distributions collected from parsed corpora. Finally, we will discuss the applications discussed briefly above in more detail and conclude.

2 The Model

2.1 Semantic Hierarchies as HMM's

As mentioned above, the shape of our HMM's is determined by the semantic class hierarchy we receive as input. We would like our approach to be compatible with as wide a range of semantic assumptions as possible. For our purposes, a semantic hierarchy consists of a directed acyclic graph whose nodes represent **classes** or **concepts** and whose arcs represent hyponymy ("is-a" links). We also assume an *expressibility* relation that associates concepts with the words that can be used to express them. For example, in Figure 2, the concept PERSON can be expressed by *person, someone,* or *mortal.* **Word senses** are defined to be expressible concepts; e.g., in Figure 2, PERSON and WORKER are word senses but ENTITY and LOCATION are not. We make no assumptions at all about what concepts "mean"—whether they are to be interpreted extensionally or intensionally, what sorts of inferences they support, etc. To inform intuition, we use the labels assigned to concepts by the creators of the semantic hierarchy, but these labels play no role in our models. We assume without loss of generality that the graph has a unique root node from which all other nodes can be reached. (If the hierarchy has multiple roots, we add a new unique root node with arcs leading to each of the former roots.)

The hierarchy that we have used for our experiments is WordNet. In WordNet, the role of concepts is played by what are called **synsets**—sets of synonymous words. That is, concepts are identified with the set of words that can be used to express them. As a consequence, every concept is a word sense; every concept is associated with one or more words. Hyponymy relations are defined between synsets, providing the arcs of the graph. A fragment of the hierarchy is illustrated in Figure 2. The solid arrows rep-

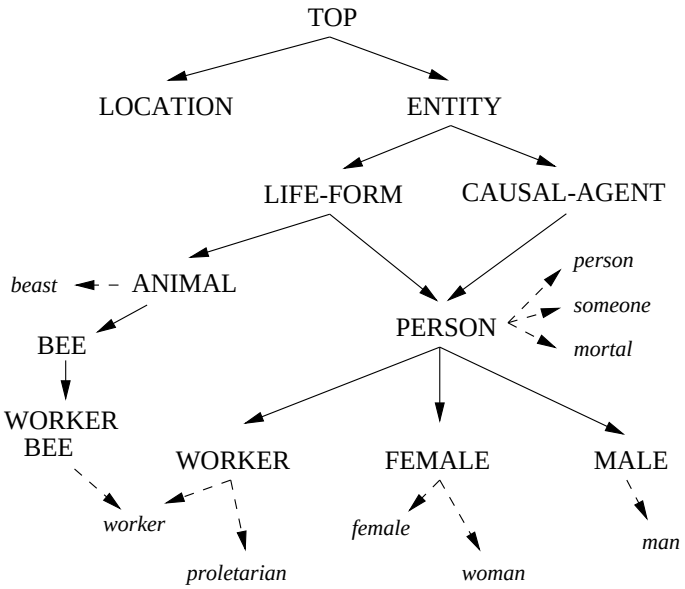

Figure 2: Example Semantic Class Hierarchy

resent hyponymy and the dashed arrows represent expressibility. Note that there is multiple parentage: PERSON is a hyponym of both LIFE-FORM and CAUSAL-AGENT. *Worker* is an example of an ambiguous word: it is a word that expresses two different concepts, i.e., it "has" two word senses (where word sense has the meaning defined above). As labels, we have used the first word in a synset. (In the complete graph, there are cases in which multiple synsets have the same first word. In such a case, we add an arbitrary digit to keep the labels distinct.)

Next we define the stochastic generation model, by constructing an HMM based on the concept graph. In the simplest case, the states of the HMM are identified with the nodes of the concept graph, the transitions of the HMM are identified with the arcs (hyponymy links) of the concept graph, and the emissions of the HMM are identified with the expressibility relation. In a bit more detail, the HMM is defined as follows. A concept graph is given, and an expressibility relation from nodes to words. The nodes of the graph are identified with concepts $\mathcal{C} = \{c_1, \ldots, c_n\}$, and the expressibility relation relates concepts to words $\mathcal{W} = \{w_1, \ldots, w_m\}$. The HMM consists of a set of **states** $\{q_1, \ldots, q_n\}$, which we identify with the nodes of the concept graph; a set of **possible emissions** which we identify with $\mathcal{W} \cup \{\epsilon\}$ (that is, we permit

non-emitting states); and three parameter matrices:

$A = \{a_{ij}\}$ The transition probabilities. The value a_{ij} represents the probability of making a transition from state q_i to state q_j. a_{ij} is nonzero only if there is an arc in the concept graph from concept c_i to concept c_j.

$B = \{b_j(k)\}$ The emission probabilities. The value $b_j(k)$ represents the probability of emitting word w_k while in state q_j. States corresponding to nonterminal nodes in the concept graph are non-emitting (that is, they emit ϵ with probability 1), and states corresponding to terminal nodes are emitting states (they emit ϵ with probability 0).

$\pi = \{\pi_i\}$ The initial state distribution. π_i is identically 1 for the start state (corresponding to the root node), and 0 for all other states.

Following [3], we characterize HMM's as probabilistic Moore machines—that is, transducers that generate output from states rather than transitions. Rabiner's formulae do not explicitly allow for non-emitting states, but the required generalization is straightforward.

In the definition just given, it is assumed that all and only the terminal nodes in the concept graph are word senses. We would like to relax that condition—WordNet, for example, associates words with all nodes of the concept graph. For now, we will require that all non-final states of the HMM be non-emitting states, and that all final states be emitting states.[2] This is to guarantee that every observation sequence consists of a single word. In addition, it guarantees that the probability of the observation symbols other than ϵ form a distribution. To accommodate a more general concept graphs, we complicate the mapping from concepts to states. We continue to require that all terminal nodes in the concept graph be word senses, but we now relax the requirement that all word senses be terminal nodes. In Figure 2, PERSON is such a nonterminal word sense. A nonterminal word sense c is associated with a pair of states (q_f, q_{nf}); q_f is a final state and q_{nf} is non-final. The incoming and outgoing arcs of node c correspond to transitions involving q_{nf}, whereas the words expressing c correspond to emissions from q_f. In addition, there is a single transition from q_{nf} to q_f. The mapping is illustrated in Figure 3.

[2] Later in section 3 we will loosen this restriction to allow for final states that emit ϵ with probability 1.

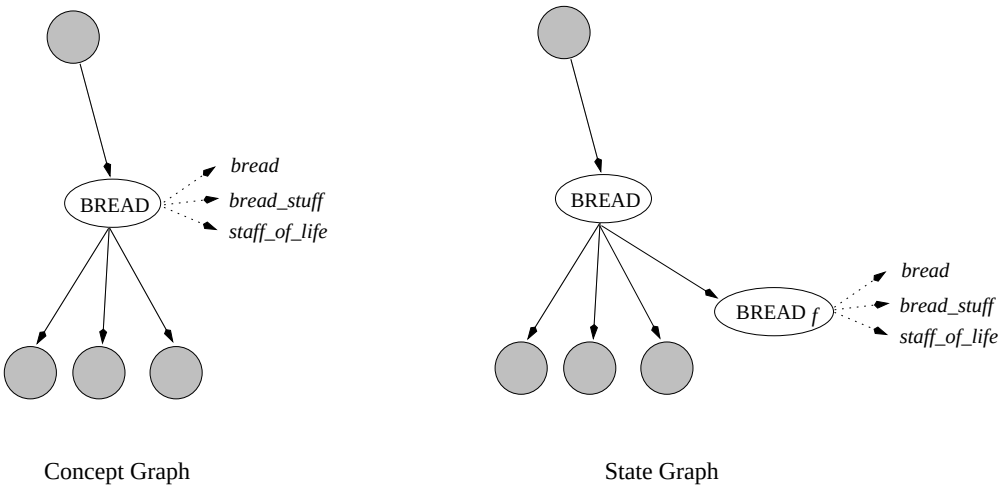

Figure 3: Converting WordNet

We construct one such HMM for each predicate-role pair (v, r), and one background model trained on all nouns in the corpus, regardless of context. Each HMM has the same structure and the same set of parameters, but they differ in their values for the parameters. The differences in parameter values is determined by the differing frequencies of words in their training samples. The foreground model for context (v, r) is trained on the words appearing in co-occurence tuples (v, r, n), one token of n for each token of (v, r, n) in the corpus. We use the a forward-backward-like algorithm to estimate parameter values from training samples.

3 Parameter Estimation

We had originally hoped that after turning our semantic hierarchy into an HMM as described above, we could simply run the standard forward-backward algorithm on the training corpus and we would get a useful model. Unfortunately, there are a number of reasons why this does not work. We will describe these problems and our solutions in the context of disambiguating the words in the training data with multiple word senses, a fundamental task in the estimation of selectional preferences. In each of the three subsections below we describe a problem and our solution.

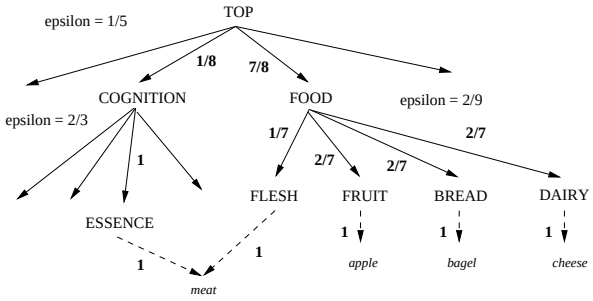

Figure 4: Smoothing

3.1 Smoothing

It was our original hope that, by treating the choice of word sense as just another hidden variable in the HMM, word sense disambiguation would be accomplished as a side effect of EM estimation. In fact, however, there is no pressure in the model in favor of parameter settings in which occurences of an ambiguous word are all accounted for by a single word sense. If the initial parameter settings account for an ambiguous word as a mixture of word senses, the converged model does likewise.

For example, consider Figure 4. We assume a miniature training corpus, containing one instance each of four words, *meat, apple, bagel, cheese*. The *meat* is ambiguous, having both sense ESSENCE and sense FLESH. The training corpus is perfectly accounted for by the weights in Figure 4, and this is indeed a fixed point of the EM algorithm.

One would like to introduce some pressure toward consolidating word occurances under a single word sense. Further, one would like the set of word senses one ends up with to be as closely related as possible. In Figure 4, for example, one would like word *meat* to shift as much of its weight as possible to sense FLESH, not sense ESSENCE.

We have sought to accomplish this in a natural way by smoothing transition probabilities, as follows. The transition probabilities out of a given state constitute a probability distribution. At a given iteration of the EM algorithm, the "empirical" distribution for a given state is the distribution of counts across outgoing transitions, where the counts are estimated using the model produced by the previous iteration. (Hence the scare quotes around *empirical*. For want of a better term, let us call this distribution *pseudo-empirical*.)

For example, assume the parameter settings shown in Figure 4 to be the

output of the previous iteration, and assume that each word appears once in the training corpus. Then the (estimated) count for the path through transition FOOD $\rightarrow$ FLESH is 1/2, and the count for the paths through transitions FOOD $\rightarrow$ FRUIT, FOOD $\rightarrow$ BREAD, FOOD $\rightarrow$ DAIRY, is 1 each. Hence, the total count for state FOOD is 31/2. Dividing each transition count by the count for state FOOD yields the pseudo-empirical probabilities $\{1/7, 2/7, 2/7, 2/7\}$.

The pseudo-empirical probabilities would normally be installed as transition weights in the new model. Instead, we mix them with the uniform distribution $\{1/4, 1/4, 1/4, 1/4\}$. Let $p(t)$ be the pseudo-empirical probability of transition t, and let $u(t)$ be the uniform probability of transition t. Instead of setting the new weight for t to $p(t)$, we set it to $\varepsilon\ u(t) + (1-\varepsilon)p(t)$.

Crucially, we make the mixing parameter, ε, a function of the total count for the state. Intuitively, if there is a lot of empirical evidence for the distribution, we rely on it, and if there is not much empirical evidence, we mix in a larger proportion of the uniform distribution. To be precise, we compute ε as $1/(c+1)$, for c the total count of the state. This has the desirable property that ε is 1 when c is 0, and ε decreases exponentially with increasing c.

It is probably not immediately obvious how smoothing in this manner helps to prune undesired word senses. To explain, consider what happens in Figure 4. There are two paths from the root to word *meat*, one leading through word sense ESSENCE and the other leading through word sense FLESH. In the "previous" model (i.e., the weights shown), each of those paths has the same weight (namely, 1/8), hence each instance of word *meat* in the training corpus is taken as evidence in equal parts for word senses ESSENCE and FLESH.

The difference lies in states COGNITION and FOOD. Words *apple, bagel, cheese*, along with half of *meat*, provide evidence for state FOOD, giving it a total count of 31/2; but the only evidence for state COGNITION is the other half of *meat*, giving it a total count of 1/2. The new distribution for COGNITION has a large admixture of the uniform distribution, whereas the distribution of FOOD has a much smaller uniform component.

The large proportion of uniform probability for state COGNITION causes much of its probability mass to be "bled off" onto siblings of ESSENCE (not shown, but indicated by the additional outgoing edges from COGNITION). Since none of these sibling are attested in the training corpus, this makes COGNITION's fit to the training corpus very poor. Intuitively, this

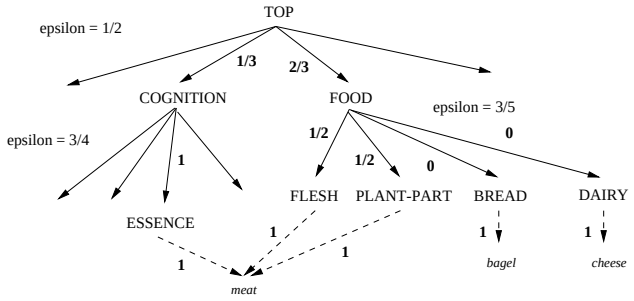

Figure 5: Imbalanced Senses

creates pressure for TOP to reduce the weight it apportions to COGNITION and increase its weight for FOOD; doing so improves the model's overall fit to the training corpus.

This decreases the relative count for word sense ESSENCE in the next iteration, increasing the pressure to shift weight from COGNITION to FOOD. Ultimately, an equilibrium is reached in which most of the count for word *meat* is assigned to word sense FLESH. (What prevents a total shift to word sense FLESH is smoothing at TOP , which keeps a small amount of weight on COGNITION. In a large hierarchy, this translates to a vanishingly small amount of weight on ESSENCE.)

3.2 Sense Balancing

In Figure 4, our smoothing method produces the desired bias for the corpus *meat, apple, bagel, cheese*. However, in different circumstances the bias produced is not the desired one. Consider training the hierarchy in Figure 5 on a corpus made up of one token of *meat*.

The hierarchy in Figure 5 differs from the hierarchy in Figure 4 in that *meat* has three senses two of which share a prefix path, i.e., the transition from TOP to FOOD. When training on the corpus of one token of *meat*, 2/3 of the count would go down the FOOD side and the other third down the COGNITION side; thus with respect to the forward-backward algorithm there is little difference between the current example and the previous one and thus the two senses of *meat* under FOOD will be preferred. Intuitively this is wrong, there is no information in the corpus on which to derive a bias for any one sense and we would like our parameter settings to reflect this. In addition, this is also not simply a border case problem, since if, as in the corpus in Table 2, *meat* is very frequent, it could easily happen that such

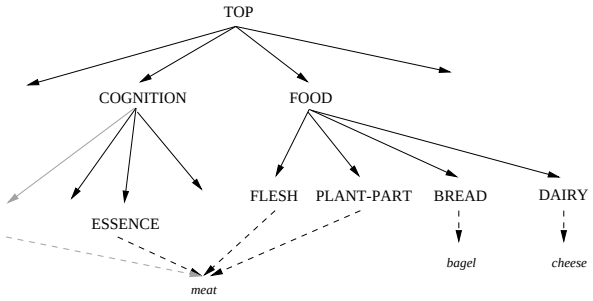

Figure 6: Sense Balancing

an a priori bias for certain senses of *meat* drowns out the bias that should result from the other words in the corpus.

In concrete terms, the problem is the shared path prefix that exists for the senses under FOOD, namely the transition from TOP to FOOD. More abstractly, the problem is that the hierarchy is not balanced with respect to the senses of *meat*—if there were another sense under ESSENCE there would be no problem (see Figure 6).

One can simulate such a phantom sense within the forward-backward algorithm. (Remember that the forward-backward algorithm is an Expectation Maximization (EM) algorithm and thus consists of two primary steps: the Expectation step where, based on the current model and the corpus, frequencies, or counts, of the events of the model are estimated and the the Maximization step which consists of a normalization of these frequences to produce the next parameter settings of the model.) First the count for the transitions in the prefix path have to be reduced. This can be done by modifying the E step such that the expectation, $\widehat{E_w}(X_{i\to j})$, for the random variable, $X_{i\to j}$, which corresponds to the transition from state i to state j for a single token of word w, is calculated as follows.

$$\widehat{E_w}(X_{i\to j}) = \frac{E_w(X_{i\to j})}{\mathcal{D}(j,w)}$$

where $E_w()$ is the expectation based on the model and corpus and $\mathcal{D}(j,w)$ is the number of unique paths starting at j and ending in a state that can generate w. One then sums over all tokens of the corpus to get the expectation for the corpus.

The second step is to reduce the probability of the paths to the sister sense of the phantom sense, e.g., COGNITION$\to$ ESSENCE. This can be

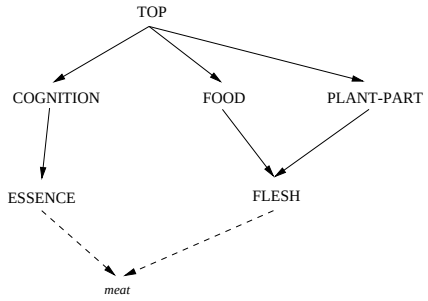

Figure 7: Graph with Reentrancy

achieved by increasing the normalization factor used in the M step:

$$\widehat{A_w} = \widehat{E_w}(\mathcal{D}(r,w) - \mathcal{D}(i,w))$$

where again we focus on the contribution of a single token of a word w and thus the normalization factor used in the M step would be the sum $\widehat{A_w}$ over the tokens in the corpus. The state r is the starting state of the model, i.e., the state corresponding to the root of the hierarchy. The exception to this formula occurs when $\mathcal{D}(r,w) - \mathcal{D}(i,w) = 0$ in which case $\widehat{A_w} = \widehat{E_w}$.

There are other ways of modifying the algorithm to simulate the phantom sense. However this method is easy and efficient to implement since the E and M steps remain simple local calculations—the only global information comes through the function d which can be efficiently and easily computed.

Another kind of sense imbalance is shown in Figure 7. this imbalance can be corrected by further modifying the E step as follows:

$$\widehat{E_w}(X_{i\to j}) = \frac{E_w(X_{i\to j})}{\mathcal{D}(j,w)\mathcal{U}(j)}$$

where $\mathcal{U}(j)$ is the number of unique paths up to the root from j.

3.3 Length and Width Balancing

Most of the example hierarchies/models we have considered so far have been balanced with respect to length and width, i.e., the length of the paths to the generating states has been uniform and the number of transitions out of a state has been uniform across states. It turns out that uniform length and width are important characteristics with respect to our modified forward-backward algorithm: shorter paths are preferred to longer ones (see Figure 8)

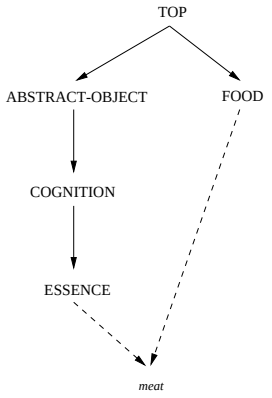

Figure 8: Path on the Right is Preferred Due to its Shorter Length

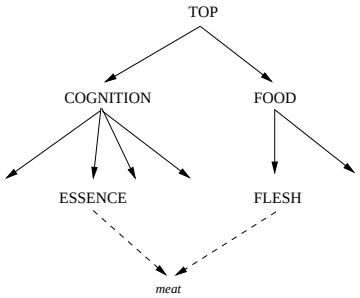

Figure 9: Path on the Right is Preferred Due to its Relatively Narrow Distributions

and paths that go through states with few exiting transitions are preferred to ones that go through states with many (see Figure 9). In fact, short paths are preferred to longer ones by the standard forward-backward algorithm, since in an HMM the probabilities of events in a sequence are multiplied to get the probability of the sequence as a whole. Width only comes into play when introduces the smoothing. Remember that in our smoothing, we mix in the uniform probability. Consider the transitions coming out of the state COGNITION in Figure 9; there are four transitions and thus the uniform probability would be 1/4. In contrast the transitions coming out of the state FOOD in the same figure number only 2 and thus the uniform distribution would be 1/2. Thus, if their are many transitions the probability mixed for the uniform distribution will be smaller than if their where fewer transitions.

We can solve the problem by balancing the hierarchy: all paths that

result in generating a symbol should be of the same length and all distributions should contain the same number of members. As in the previous section, we can simulate this balancing by modifying the forward-backward algorithm.

First, to balance for width, the smoothing can be modified as follows: instead of mixing in the uniform probability for a particular parameter, always mix in the same probability, namely the uniform probability of the largest distribution, u_{max}, (i.e., the state with the largest number of exiting transitions, in Figure 9, this maximum uniform probability would be 1/4). Thus the smoothing formula becomes $\varepsilon\ u_{max} + (1-\varepsilon)p(t)$. This modification has the effect that it is as if there are always the same number of transitions out of a class. Width balancing for emission parameters is performed in an analogous fashion.

Let us turn to length balancing. Conceptually, in order to balance for length, extra transitions and states need to be added to short paths so that they are as long as the maximum length path of the hierarchy. It should be noted that we are only concerned with paths that end in a state that generates words. The extension of short paths can be simulated by multiplying the probability of a path by a factor that is dependent on its length:

$$\widehat{Prob}(p) = Prob(p)\, u_{max}{}^{(length_{max} - length(p))}$$

This additional factor can be worked into the forward and backward variable calculations so that there is no loss in efficiency. It is, thus, as if $length_{max} - length(p)$ states have been added and that each of these states has $u_{max}{}^{-1}$ exiting transitions.

The gray portions of Figures 10 and 11 illustrate the effect of width and length balancing. Note that these balancing extensions force us to allow final states to generate ϵ. For example, that states introduced for width balancing are final and ϵ generating.

3.4 Some Final Comments on Parameter Estimation

We started this section by showing why the straightforward application of an EM algorithm, namely the forward-backward algorithm, would not sense disambiguate the input words as desired. Thus, we introduced a type of smoothing which produced the desired bias in the example at hand. Then we showed how this smoothing when used on certain graphs produced unwanted biases which then necessitated further modifications in the E and M

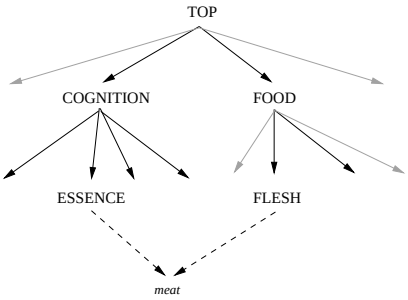

Figure 10: Width Balancing

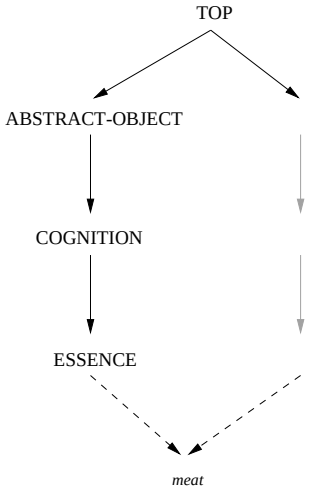

Figure 11: Length Balancing

steps of the algorithm. It is important to note that these modifications result in a version of the forward-backward algorithm with unknown behavior. One of the advantages of EM algorithms has been lost: it has been proven that they converge to local maxima with respect to the likelyhood of the training corpus [?]. It is conceivable that an analogous proof of desirable convergence properties could be constructed which would incorporate the smoothing extension. However, it is unlikely that the smoothing combined with the balancing extensions produce an algorithm with any such convergence properties. This is because the balancing extensions introduce paths that do not generate any observable word and, because of the smoothing, these paths will receive non-zero probability. Thus, the weights for observable symbols that the model defines do not sum to 1. The situation is thus analogous to the one for stochastic attribute-value grammars when trained using the straightforward extension of the inside-outside algorithm described in [?] where it is shown that the inside-outside algorithm does not converge to local maxima with respect to the likelyhood of the training corpus. Furthermore, we have observed, for some training corpora, an osilating behaviour in the cross entropy with the empirical distribution from iteration to iteration.

One possible conclusion that could be drawn from the foregoing is that HMM's are not the right kind of stochastic model to be used in conjunction with "is-a" hierarchies: in other words HMM's do not match the semantics of the "is-a" hierarchies. The need for smoothing and balancing, as described above, is a symptom of the mismatch. Perhaps a better approach would be to work with log linear model, since log linear models do not calculate the probability of an observation as a sequence of events but instead as a set of properties. Log linear models have the following form

$$Prob(w) = \frac{1}{Z} e^{\lambda_1 f_1(w) + \lambda_2 f_2(w) + \ldots + \lambda_i f_i(w)}$$

where Z is a normalizing factor, $f_i(w)$ denotes the frequency of the property i with respect to w, and the λ values are the parameters of the model and denote the weights for the properties of the words. For our purposes such a model would generate a distribution over words w, the properties would be class membership, and thus the frequency values would be either 0 or 1.

If having all classes as properties proves to be inefficient or otherwise undesirable, one could employ a greedy property selection algorithm, as in [?], where one would only add those features that correspond to classes that cover as many words in the training set as possible and at the same time

are as small as possible. Such an algorithm would produce non-optimal solutions, in many cases but note that the problem of picking an optimal set of classes with respect to a corpus is similar to the set-covering problem which is NP-complete. Such an algorithm would try to pick properties to add to the model that make the largest improvement in the ability of the model to maximize the probability of the training corpus.

Once the properties have been selected, one could use an EM algorithm to train the model on the training corpus on word tokens. Since such training would be an instance of training with incomplete data, the results and methods described in [**?**] would be relevant.

4 Using the Trained Models

Although problems with parameter estimation tarnish HMM's appeal, using trained HMM's is both straightforward and efficient. In this section we will show how to calculate three distributions which have numerous applications. First, however, let us review Figure 1. The input to the system consists of a semantic hierarchy and a set of corpora, one for each predicate-role pair. For a given predicate-role pair, such as *eat-OBJ*, a corpus might look like the word in Table 2. The output of the system is then a trained HMM which represents the selectional preferences of the particular predicate-role pair.

In the following we will make use of a number of functions:

$$\begin{aligned} \alpha_t(i) &= Prob(w, q_t = S_i) \\ \beta_t(i) &= Prob(w|q_t = S_i) \\ \gamma_t(i) &= Prob(q_t = S_i|w) \end{aligned}$$

where w is the word generated, S_i is a state in the HMM, and q_t is the state which the HMM finds itself in after t transitions.

The α and β values can be computed efficiently using dynamic programming techniques. They provide the basis for other calculations:

$$\begin{aligned} Prob(w) &= \beta_0(TOP) \\ \gamma_t(i) &= \frac{\alpha_t(i)\beta_t(i)}{Prob(w)} \end{aligned}$$

We denote the probability function of an HMM trained on data from a particular predicate-role pair as $Prob_{\textit{pred-role}}$.

4.1 Word Sense Disambiguation

We have discussed at length the problem of disambiguating the input corpus during training. Here we will discuss how to calculate a distribution over word senses given a trained HMM.

Consider the sentence

He splattered **OJ** *on the table.*

The noun **OJ** is ambiguous: it could refer to the former star running back of the Buffalo Bills or to orange juice. However, due to the selectional preferences of *splatter*-OBJ, one would expect that the orange juice reading would be more likely.

We would like to have a distribution over the senses of the word based on the predicate-slot pair. This can be expressed in terms of the HMM trained on the corpus corresponding to the predicate-slot pair as follows.

$$Prob(sense|word, \textit{pred-slot}) = Prob_{\textit{pred-role}}(S_{sense}|word) = \sum_{t=0}^{l_{max}} \gamma_t(S_{sense})$$

Where l_{max} is the length of the longest path in the HMM. Remember that word senses correspond to classes of the semantic hierarchy which in turn correspond to states in an HMM. In words, the probability of a sense is the probability of generating the word from the state which corresponds to that sense. In Section 5, we will present results based on this calculation.

4.2 Producing Distributions over Words

For many applications, one needs to know how likely one word is as opposed to another. For example, consider translating the german sentence:

Machen wir einen **Termin** *aus, oder?*

into English and let us assume that we are trying to decide which translation of *Termin* is most appropriate given our translation of the rest of the sentence:[3]

Let's arrange a	time-slot, *ok?*
	date
	appointment

[3] We would like to thank Detlef Prescher for providing us with this example which comes from the Verbmobil project domain.

It would be useful to know which word, *time-slot, date,* or *appointment*, is the most likely in this english sentence. Both multilingual information retrieval and speech recognition have similar needs.

The probability of a word given a predicate-slot pair is simply the probability that the HMM for that pair generates this word. As mentioned above, this is the β value at time 0 for the TOP class or more intuitively it is the sum of the probabilities of all the paths through the HMM that result in the word being generated.

$$Prob(word|\textit{pred-slot}) = Prob_{\textit{pred-slot}}(word) = \beta_0(S_{TOP})$$

4.3 Produce Traditional List of Classes

Consider the HMM in Figure 12 where transitions have been added from all leaf states to the root state. Our HMM's, when extended in this way, turn out to be ergodic, as shown in Appendix A, and thus the probability of being in a particular state at a time t converges to a single value as t approaches ∞. These steady-state probabilities can be put entirely in terms of the parameters of the model as shown in Appendix B. Thus, once an HMM has been trained, the steady state probabilities can be easily calculated. Because of the correspondence between states and classes, these steady state distributions can be interpreted as a distribution over classes.

$$Prob(class|\textit{pred-slot}) = \lim_{t\to\infty} Prob_{\textit{pred-slot-looped}}(S_{class})$$

One can then train background and foreground models as defined in Section 1 and then, following Resnik, use pointwise divergence between the foreground and background model class distribution to provide selectional preferences defined in terms of weights on classes.

Many natural language processing applications have been built to make use of selectional preferences represented as a list of semantic classes. And thus such a system would be useful for automatically producing selectional preferences preferences for words in a new domain.

5 Evaluation

5.1 Word Sense Disambiguation

As was described above, distributions over word sense can be calculated from trained HMM's. The evaluate described here based on that described

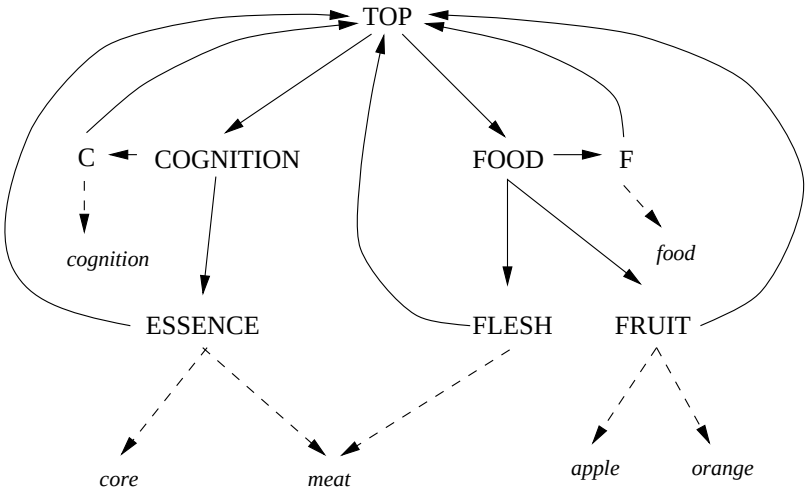

Figure 12: HMM Used for Calculating Steady State State Probabilities

in [4].[4]

Resnik started with the UPenn Tree Bank parses of the Brown Corpus and the Semcor WordNet Sense tagging of two-eighths of the Brown Corpus. First he estimated distibutions over WordNet classes for each verb which occurred more than once. And then computed the divergence between these distributions and a background distribution. The 100 verbs with the largest divergence were selected. Training corpora were extracted for each verb from the six-eighths of the Brown Corpus which were not part of Semcor using the UPenn Tree Bank parses to find the heads of relevant complements. Here we will only be concerned with the corpora corresponding to the tokens that filled the direct object position of the verbs. The test corpora were similarly extracted except that the Semcor two-eighths of the Brown Corpus were used. In addition, the Semcor sense tagging for each complement head token was also retained.

For each of the 100 verbs we trained an HMM model based on the WordNet hierarchy using the training corpora described above. Then we used the $Prob_{\textit{pred-role}}(S_{sense}|word)$ described above to do word sense disambiguation for the tokens of the test corpora. The results are presented in Table 3.

In addition, to training corpora derived from the Brown corpus, we made use of training corpora derived from the British National Corpus which we

[4]We would like to thank Philip Resnik for providing us with the training and test data which he made use of in the above mentioned work.

Method	Brown Corpus Training Set	BNC Training Set
Random	28.5%	28.5%
First Sense	82.8%	82.8%
Resnik	44.3%	
HMM unbalanced	35.6%	36.5%
HMM balanced	42.3%	54.2%
HMM Brown BG-model only	46.7%	
HMM BNC BG-model only	42.2%	

Table 3: Word Sense Disambiguation Results

parsed using the Cass finite-state parser [2].

References

[1] S. Abney. Partial parsing via finite-state cascades. *Natural Language Engineering*, 2(4), 1996.

[2] Steven Abney. Partial parsing via finite-state cascades. In John Carroll, editor, *Proc. Robust Parsing Workshop, ESSLLI Summer School 1996*, pages 8–15, 1996.

[3] L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, February 1989.

[4] P. Resnik. Selectional preference and sense disambiguation. In *Proceedings of the ANLP-97 Workshop: Tagging Text with Lexical Semantics: Why, What, and How?*, Washington, D.C., 1997.

[5] P. S. Resnik. *Selection and Information: A Class-Based Approach to Lexical Relationships*. PhD thesis, University of Pennsylvania, Philadephia, PN, 1993.

[6] F. Ribas. On learning more appropriate selectional restrictions. In *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics.*, Dublin, Ireland, 1995.

A Appendix A: Proof of Ergodicity

We follow [?] in uses the following notation:

> $p_{ij}(n)$ is the probability of going from state i to state j in n steps;
>
> $P_j(n)$ is the probability of being in a state j after n time clicks.

The Ergodic Theorem as stated in [?] on page 170 is

> "Let M_k be the matrix of k-step transition probabilities of a markov process with a finite number of states $S_1, S_2, ..., S_m$. If there exists an integer k such that the terms $p_{ij}(k)$ of the matrix M_k satisfy the relation
>
> $$\min_{1 \leq i \leq m} p_{ij}(k) = \delta > 0$$
>
> for at least one column of M_k, then the equalities
>
> $$\lim_{n \to \infty} p_{ij}(n) = P_j j = 1, 2, ..., m; \sum_j P_j = 1$$
>
> are satisfied."

He goes on to say that

> "The restriction
>
> $$\min_{1 \leq i \leq m} p_{ij}(k) = \delta > 0$$
>
> for at least one column of M_k simply requires that there be at least one state S_j and some number k such that it be possible to get to S_j from every state in exactly k transitions. This requirement happens to correspond to the conditions that the recurrent states of the system form a single chain and that there be no periodic states."

We will provide a k and an S_j for our HMM's extended by leaf-root transitions as displayed in Figure **??**. The strategy of the proof will be to use the following facts about our HMM's:

- every state can get to the root in one or two transitions because of the shape of the branches,
- loops from the root back to itself exist of size $3, 4, 5, \ldots, length_{max}+1$,

where $length_{max}$ is the longest path in the unextended HMM. Thus, for a k such that $5 < k < length_{max} + 1$, every state can get to the root and then use the $k-1$ or $k-2$ loop to make a transition to root in k steps. In the following r is the root or top state. In addition, we assume that $length_{max} > 3$.

Due to the shape of our HMM's branches and the effects of our smoothing:

$$p_{rr}(n) > 0 || for || n = 3, 4, \ldots, length_{max} + 1$$

Note also that due to the leaf-root transitions

$$p_{ir}(1) > 0 || or || p_{ir}(2) > 0 i = 1, 2, \ldots, m.$$

Now we can specify S_j and k: S_j is r. Remember that what we want to do is specify a k such that one can get to the root state in exactly k steps from every state S_i with a non-zero probability. As it turns out, any k such that $5 < k < length_{max} + 1$ will work.

Because of the short paths for every state back to the root and because of the loops of various lengths:

For all i

if $p_{ir}(1) > 0$ then $p_{ir}(1)p_{rr}(k-1) > 0$ and thus $p_{ir}(k) > 0$

if $p_{ir}(2) > 0$ then $p_{ir}(2)p_{rr}(k-2) > 0$ and thus $p_{ir}(k) > 0$

q.e.d.

B Appendix B: Derivation of Steady State Equations

```
N   = the number of states
r   = root state steady state probability
i   = ith state steady state probability
g   = a generating (leaf) state steady state probability
i_j = the transition probability from state i to j

The system of equations:
```

For each state in N (excluding r)

1. $j = \sum_i i * i_j$

The probability of a state is the sum over all states with immediate transitions to it.

2. $r = \sum_g g$

The probability of the root state is the sum of the probabilities of the leaf states since forall g, $g_r = 1$.

3. $1 = \sum_i i$

The steady state probabilities of the states must sum to 1.

We want to solve these equations stating the steady state probabilities in terms of the transitions probabilities.

Since there is a path from the root to i for every non-zero i_j in the equations of form (1), we can rewrite each equation as follows:

4. $j = r * \sum_a A(a,j)$

where

$A(a,j) = \prod i_q$

and the product includes all transitions along the path a from r to j. The equations of (4) say that the probability of state is the sum of the probabilities of the paths from the root to state times the probability of the root.

You get from (1) to (4) by substituting successively for the state probability i with the equation for i of form (1).

The next step is to solve for r. Take the equations of (4) and substitute them into (3), factor out r, and divide through by the sum which yields:

5. $r = 1/(1 + \sum_{i} \sum_{a} A(a,i))$ where i varies over all states except r

Let

$Z = 1 + \sum_{i} \sum_{a} A(a,i)$ where i varies over all states except r

Now we can use this value of r plus the equations of (4) to derive the steady state probabilities of the other states;

6. $j = (\sum_{a} A(a,i))/Z$

One can make the calculation of Z a bit more efficient by noting that if you take the equations of (4) for the g states and substitute them into equation (2), and then factor out r, you get the following:

7. $1 = \sum_{g} \sum_{a} A(a,g)$ where g varies over all leaf states

This means that

$Z = 2 + \sum_{i} \sum_{a} A(a,i)$ where i varies over NON-leaf states except r