1

# Estimating Hidden Markov Model Topologies

Thorsten Brants

*Estimating Hidden Markov Model Topologies.*
**Thorsten Brants**
Universität des Saarlandes, Computational Linguistics
P.O.Box 151150, D-66041 Saarbrücken, Germany
thorsten@coli.uni-sb.de.

**Abstract**

There are several ways of estimating parameters for HMMs when used for natural language models. One can use word-$n$-grams and $n$-grams of automatically derived categories for speech recognition. Or one can use part-of-speech $n$-grams for part-of-speech tagging, either by using a manually tagged corpus or by using the Baum-Welch algorithm. This paper shows how to use another method for parameter estimation: Model Merging. It exploits the advantages of the other methods, is applicable both for speech recognition and part-of-speech tagging and, unlike other techniques, it not only induces transition and output probabilities but also the model topology, i.e., the number of states and their respective possible outputs. Thus it automatically generates categories, but in addition to other categorization algorithms is capable of recognizing if a word belongs to more than one category. By adding optimizations the algorithm is used to generate language models that are used for part-of-speech tagging. The accuracy in a tagging task is better than the accuracy of HMMs derived by standard techniques.

## 1.1 Introduction

Hidden Markov Models are commonly used for statistical language models, e.g., in part-of-speech tagging and speech recognition (Rabiner 1989). The models need a large set of parameters which are induced from a (text-) corpus. The parameters should be optimal in the sense that the resulting models assign high probabilities to seen training data as well as new data that arises in an application.

There are several methods to estimate model parameters. The first one is to use each word (type) as a state and estimate the transition probabilities between two or three words by using the relative frequencies of a corpus. This method is commonly used in speech recognition and known as word-bigram or word-trigram model. The relative frequencies have to be smoothed to handle the sparse data problem and to avoid zero probabilities.

The second method is a variation of the first method. Now, words are automatically grouped, e.g., by similarity of distribution in the corpus (Pereira et al. 1993). The relative frequencies of pairs or triples of groups (categories, clusters) are used as model parameters, each group is represented by a state in the model. The second method has the advantage of drastically reducing the number of model parameters and thereby reducing the sparse data problem; there is more data per group than per word, thus estimates are preciser.

The third method uses manually defined categories. They are linguistically motivated and usually called *parts-of-speech*. An important difference to the second method with automatically derived categories is that with the manual definition a word can belong to more than one category. A corpus is (manually) tagged with the categories and transition probabilities between two or three categories are estimated from their relative frequencies. This method is commonly used for part-of-speech tagging (Church 1988).

The fourth method is a variation of the third method and is also used for part-of-speech tagging. This method does not need a pre-annotated corpus for parameter estimation. Instead it uses a lexicon stating the possible parts-of-speech for each word, a raw text corpus, and an initial bias for the transition and output probabilities. The parameters are estimated by using the Baum-Welch algorithm (Baum et al. 1970). The accuracy of the derived model depends heavily on the initial bias, but with a good choice results are comparable to those of method three (Cutting et al. 1992).

This paper proposes a fifth method for estimating natural language models, combining the advantages of the methods mentioned above. It

is suitable for both speech recognition and part-of-speech tagging, has the advantage of automatically deriving word categories from a corpus and is capable of recognizing the fact that a word belongs to more than one category. Unlike other techniques it not only induces transition and output probabilities, but also the model topology, i.e., the number of states, and for each state the outputs that have non-zero probabilities. A further advantage is that the method adapts to the amount of data available. If there is only little data, the level of abstraction is low, and the use of the induced HMM is similar to Lazy Learning approaches. When more data is collected, the level of abstraction becomes higher. The method is called Model Merging and was introduced by Omohundro 1992 to derive HMMs for regular languages from a few samples.

The rest of this paper is structured as follows. Section 1.2 gives the definitions of Hidden Markov Models, part-of-speech tagging and Model Merging. Section 1.3 motivates the use of model merging for natural language models and shows how to use it for a first application: part-of-speech tagging. Section 1.4 reports about experiments performed with models derived by Model Merging, and finally Section 1.5 shows further directions.

## 1.2 Definitions

### 1.2.1 Hidden Markov Models

A discrete output, first order *Hidden Markov Model (HMM)* consists of

- a finite set of states $\mathcal{Q} \cup \{q_s, q_e\}$, $q_s, q_e \notin \mathcal{Q}$, with $q_s$ the start state, and $q_e$ the end state;
- a finite output alphabet $\Sigma$;
- a set of state transitions $(q \rightarrow q')$, $q \in \mathcal{Q} \cup \{q_s\}$, $q' \in \mathcal{Q} \cup \{q_e\}$; for each transition $(q \rightarrow q')$ a probability $p(q \rightarrow q')$ is specified; for each state $q$, the sum of the outgoing transition probabilities is 1, $\sum_{q' \in \mathcal{Q}} p(q \rightarrow q') = 1$;
- a set of state-output pairs $(q \uparrow \sigma)$, $q \in \mathcal{Q}$, $\sigma \in \Sigma$; for each pair $(q \uparrow \sigma)$ a probability $p(q \uparrow \sigma)$ is specified; for each state $q$, the sum of the output probabilities is 1, $\sum_{\sigma \in \Sigma} p(q \uparrow \sigma) = 1$.

Figure 1 shows an example for a HMM. The HMM starts running in the start state $q_s$, makes a transition at each time step, and stops when reaching the end state $q_e$. The transition from one state to another is done according to the probabilities specified with the transitions. Each time a state is entered (except the start and end state) one of the out-

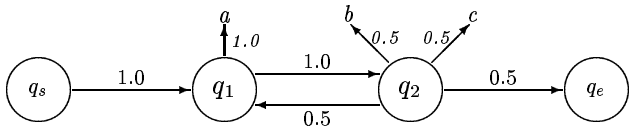



FIGURE 1 Example HMM. It generates the language $L = (a(b|c))^+$. The figure shows the states, outputs, transitions, and probabilities for each output and transition.

puts is chosen (again according to their probabilities) and emitted. The HMM in Figure 1 generates the language $L = (a(b|c))^+$. The probability associated with each output string $W \in \Sigma^*$ is

$$P(W) = \begin{cases} 0.5^{length(W)} & \text{if } W \text{ is of the form } (a(b|c))^+ \\ 0 & \text{otherwise} \end{cases}$$

If the state transitions depend on the previous state only, the HMM is of first order. If the state transitions depend on $n$ previous states, the HMM is of $n$-th order. Higher order HMMs can be reduced to first order HMMs by appropriately increasing the number of states.

The reason for calling HMMs *hidden* is that generally one cannot observe the sequence of states traversed by the HMM, but only the emitted outputs, thus the states are hidden. In the domain of language models this generally means one can observe the sequences of words but not the underlying linguistic structure.

When using HMMs for recognition, one is interested in the following questions:

1. Given a string $W \in \Sigma^*$, which sequence of states $Q \in \mathcal{Q}^*$ can have generated this string, and which is the most probable one?
2. Given a string $W \in \Sigma^*$, what is the probability for the HMM having generated the string?

Both questions can be answered very efficiently in time linear to the length of the string $W$, $O(length(W))$. This is done by using the Viterbi algorithm (Viterbi 1967).

### 1.2.2 Part-of-Speech Tagging

The task of part-of-speech (PoS) tagging is the unique annotation of a word with a syntactic category, called *part-of-speech* or *tag*.

Let $\mathcal{T}$ be defined as the set of all tags, and $\Sigma$ the set of all words. In a statistical tagging task, one is given a sequence of words $W = w_1 \dots w_k \in \Sigma^*$, and is looking for a sequence of tags $T = t_1 \dots t_k \in \mathcal{T}^*$ that maximizes the conditional probability $p(T \mid W)$, hence one is

looking for

$$\underset{T}{\operatorname{argmax}} \quad p(T|W) = \underset{T}{\operatorname{argmax}} \frac{p(T)\ p(W|T)}{p(W)}.$$

$p(W)$ is independent of the chosen tag sequence, thus it is sufficient to find

$$\underset{T}{\operatorname{argmax}} \quad p(T)\ p(W|T).$$

In an $n$-gram model for each pair $(w, t) \in \Sigma \times \mathcal{T}$, the lexical probabilities

$$p(w \mid t),$$

and for each $n$-tuple $(t_1 \ldots t_n) \in \mathcal{T} \times \ldots \times \mathcal{T}$ the transition probabilities

$$p(t_n \mid t_1 \ldots t_{n-1})$$

are defined. These approximate the lexical and conditional probabilities with

$$p(W|T) \approx p(w_1|t_1)\ p(w_2|t_2)\ \cdots\ p(w_k|t_k)$$

and

$$\begin{aligned} p(T) &= p(t_1)\ p(t_2|t_1)\ p(t_3|t_1, t_2)\ \cdots\ p(t_k|t_1 \ldots t_{k-1}) \\ &\approx \prod_{i=1}^{k} p(t_i \mid t_{i-n+1} \ldots t_{i-1}) \end{aligned}$$

Note that the beginning of the sequence requires some extra handling. Additional tags $t_{-n+2} \ldots t_0$ are introduced, marking the "start of sequence" position, or, when using HMM terms, initial states are introduced.

Now the joint probability of a sequence of words $W = w_1 \ldots w_k$ having a sequence of tags $T = t_1 \ldots t_k$ is the product of their lexical and transition probabilities

$$p(W, T) = p(T)p(W|T) \approx \prod_{i=1}^{k} p(t_i \mid t_{i-n+1} \ldots t_{i-1})p(w_i \mid t_i).$$

(making the Markov assumption) and finding the best sequence of tags $T$ for a given sequence of words $W$ is done by finding

$$\underset{t_1 \ldots t_k}{\operatorname{argmax}} \prod_{i=1}^{k} p(t_i \mid t_{i-n+1} \ldots t_{i-1})p(w_i \mid t_i).$$

Statistical PoS tagging is efficiently done with $n$-gram models. They are equivalent to Markov models of order $n - 1$. The best compromise between size of corpus that is needed for parameter estimation and quality of output has empirically proven to be the trigram models, having $n = 3$.

$n$-gram taggers have been applied successfully for several years and reach a level of accuracy of 95–96% for English texts. For implementations see for example Church 1988, DeRose 1988, and Cutting et al. 1992.

### 1.2.3 Model Merging

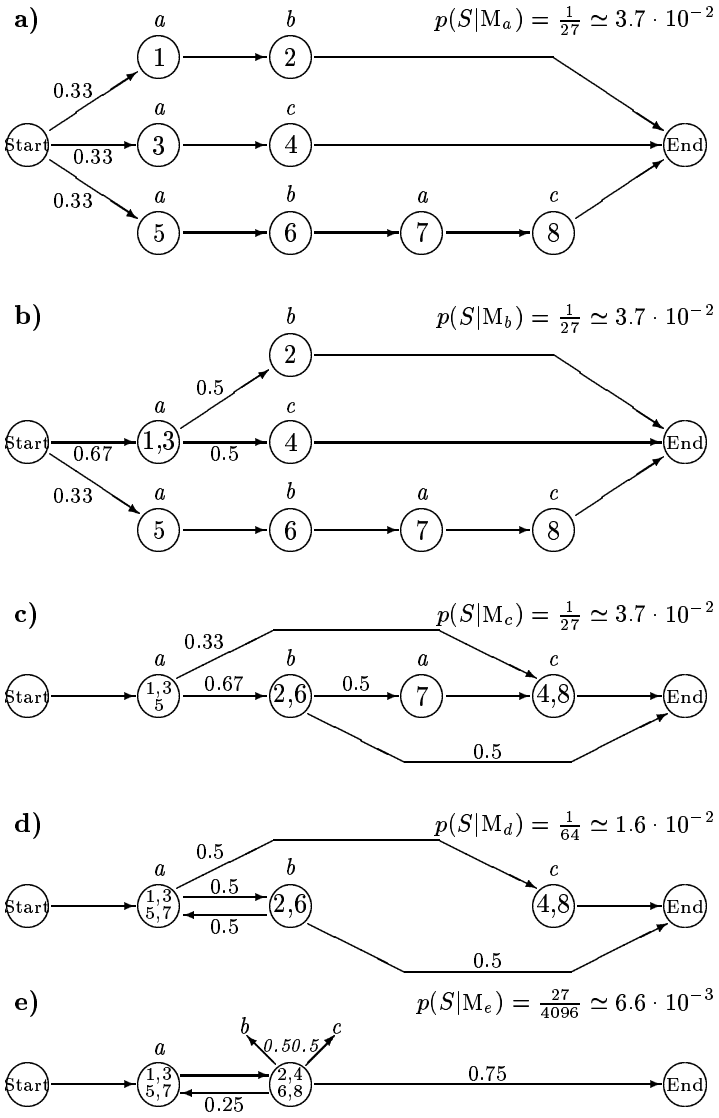
Model Merging is a technique for inducing model parameters for Hidden Markov Models from a text corpus (Omohundro 1992). Unlike other techniques it not only induces transition and output probabilities from the corpus, but also the model topology, i.e., the number of states and for each state the outputs that have non-zero probability. In $n$-gram approaches the topology is fixed. The states are mostly linguistically motivated, e.g., in a PoS-$n$-gram model, each state represents a syntactic category and only words belonging to the same category have non-zero output probabilities in a particular state. But the $n$-gram-models make the implicit assumption that all words belonging to the same category have similar distributions in a corpus. This is not true in most of the cases.

The first advantage of Model Merging is that it groups words together by their statistical distributions and estimates transition and output probabilities for a HMM at the same time. The second advantage is that Model Merging adapts to the amount of data available. If there is only little data the level of abstraction is low and the use of the induced HMM is similar to Lazy Learning approaches. When more data is collected the level of abstraction becomes higher. The limiting factor for Model Merging is that the process of merging is very time consuming (in general $O(l^4)$, $l$ length of corpus).

Model Merging induces HMMs in the following way: Merging starts with an initial model. For this we choose the trivial HMM, i.e., a model that exactly matches the corpus. There is exactly one path for each utterance in the corpus and each path is used by one utterance only. Each path gets the same probability $1/u$, with $u$ the number of utterances in the corpus. Figure 2.a shows the trivial HMM for a corpus with words $a, b, c$ and utterances $ab, ac, abac$.

Now states are merged successively, except for the start and end state. Two states are selected and removed and a new merged state is added. The transitions from and to the old states are redirected to the new state, the transition probabilities are adjusted to maximize the likelihood of the corpus; the outputs are joined and their probabilities are also adjusted to maximize the likelihood.

The criterion for selecting states to merge is the probability of the HMM generating the corpus. We want this probability to stay as high as possible. Of all possible merges (generally, there are $(n-2)(n-3)/2$




Model Merging for a corpus $S = \{ab, ac, abac\}$, starting with the trivial model in a) and ending with the generalization $(a(b|c))^+$ in e). Several steps of merging between model b) and c) are not shown. Unmarked transitions and outputs have probability 1.

FIGURE 2 Model Merging (I)

possible merges, with $n$ the number of states, incl. start and end state which are not allowed to merge) we take the merge that results in the minimal change of the probability. For the trivial HMM and $u$ pairwise different utterances the probability is $p(S|HMM_{triv}) = 1/u^u$. The probability either stays constant, as in Figure 2.b and c, or decreases, as in 2.d and e. The probability never increases because the trivial model is the maximum likelihood model which maximizes the probability of the corpus given the HMM.

Model Merging stops when a predefined threshold for the corpus probability is reached, a specified number of states is reached or some combination of both occurs. Statistically motivated criteria for termination are discussed in Stolcke and Omohundro 1994.

## 1.3 Model Merging and Natural Language Models

### 1.3.1 Classification of Words

Three important features of Model Merging are that it can classify words into categories, can recognize that a word belongs to more than one category and is able to recognize static sequences of words or parts-of-speech. No other estimation technique offers this combination. These features are described in the following sections. Table 1 shows a summary.

#### Several Words Have the Same Category

It is an essential feature of each classification algorithm to group several words into a category. The grouping is used in manual specifications of categories (as in pos-$n$-gram models) stating, e.g., that *aim* and *house* both are nouns. It is used also for conventional automatic clustering where exactly this is the intended action, grouping words by their statistical distribution. Model Merging handles grouping of words by merging states which have the appropriate outputs. An example of this behavior is shown in Figure 2. There, words $b$ and $c$ are combined into a category while $a$ is treated as belonging to another category.

#### One Word Has Several Categories

Recognizing the fact that one and the same word belongs to different categories is a more difficult task. It is exploited in manual specifications of categories which, for example, state that *aim* can be both a noun and a verb. But standard automatic grouping algorithms assume that each word belongs to exactly one category. This is an oversimplification, since a lot of words belong to more than one category and knowing the category of a word facilitates the classification of surrounding words, and transition probabilities heavily depend on the category of a word. We argue that the final performance of a language model could be im-

TABLE 1 Features of Categorizations for Language Models: a) $n$-grams with manually specified parts-of-speech, b) $n$-grams with automatically derived categories and c) HMMs derived with Model Merging

| | several words in one class | one word in several classes | sequences of arbitr. length |
|---|---|---|---|
| a) $n$-grams, manual | + | + | – |
| b) $n$-grams, automatic | + | – | – |
| c) HMMs, model merging | + | + | + |

proved if we add the possibility to assign one word to several categories to an automatic classification algorithm. Model Merging exhibits this behavior by deciding *not* to merge two states that have the same output. An example is shown in Figure 3. Word $a$ belongs to four categories in model 3.d ($a$'s before $baac$, first $a$ after $b$, second $a$ after $b$, and $a$'s after $baac$). Applying two more steps of merging yields model 3.e. Now, the word $a$ belongs to two categories ($a$'s outside $baac$, and $a$'s inside $baac$).

**Recognizing Sequences**

A big problem with using $n$-grams is the fixed context length. When working with trigrams, one uses a window of three words, and only transition probabilities within this window are considered. This holds for both manually specified categories and those derived with previous categorization algorithms.

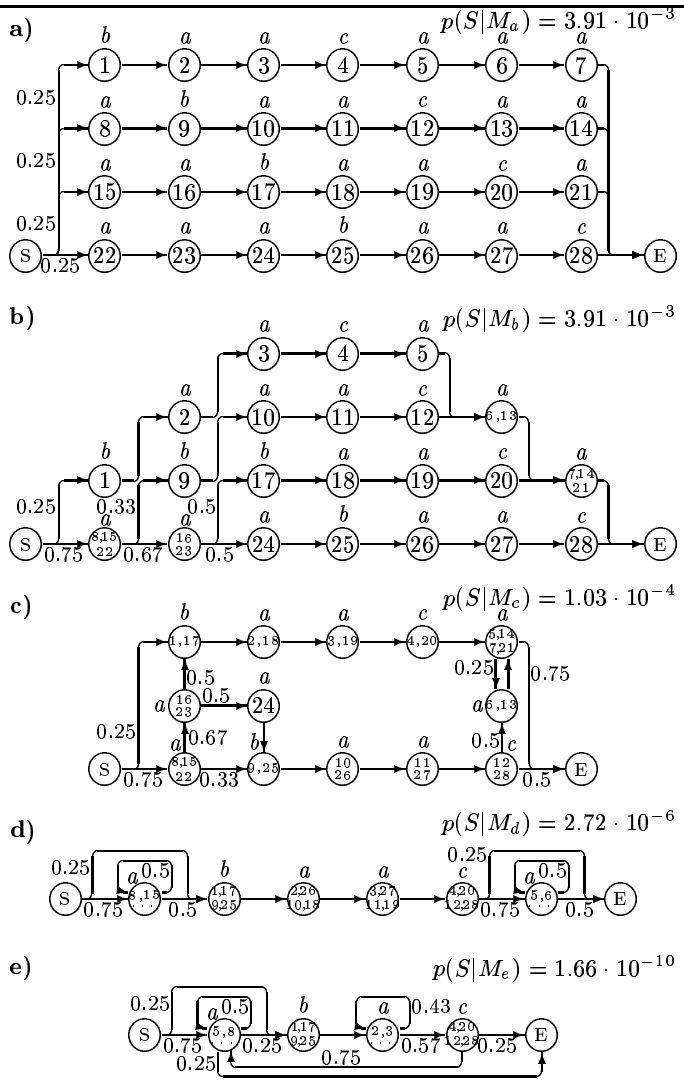
Model Merging has the ability to create window sizes that are adapted to particular contexts. Figure 3.d shows a derived HMM with a window size larger than three. In state 4 of that model it is known that the previous four words are $baac$. This feature of Model Merging is extremely useful in recognizing static sequences of words or categories and for limited recognition of long distance phenomena. The HMM in Figure 3.e "remembers" the $b$ and regardless how much material there is in-between it "waits" for a $c$. This is restricted to regular phenomena, but it is nonetheless more adequate than static context sizes.

### 1.3.2 Introducing Constraints

The Model Merging algorithm needs several optimizations to be applicable to large natural language corpora, the enormous time needed for deriving models (generally $O(l^4)$, where $l$ is the length of the corpus) has to be reduced.

Computational optimizations are mentioned in Stolcke and Omohundro 1994. We will here discuss data-dependent optimizations.

When applying Model Merging one can observe that first mainly states with the same output are merged. After several steps of mer-




Model Merging for a corpus $S = \{baacaaa, abaacaa, aabaaca, aaabaac\}$, starting with the trivial model in a) and yielding the generalizations $a^*baaca^*$ in d) and $(a^*ba^+ca^*)^+ \cup a^+$ in e). Model b) generates the same language as model a), but identical initial and final states are merged. Model c) is an intermediate model, having already merged two of the sequences $baac$. Several steps of merging between the models are not shown. Unmarked transitions and outputs have probability 1.

FIGURE 3 Model Merging (II)

ging, it is no longer the same output but still mainly states that output words of the same syntactic category are merged. This behavior can be exploited by introducing constraints on the merging process. In the beginning we consider only states with the same output. After a while this constraint is relaxed and states that output words of the same syntactic category are allowed to merge. Again, after a while this constraint is relaxed and all states are allowed to merge. Imposing these constraints on the merging speeds up the process drastically.

A further constraint that proved to be very useful involves bigrams. States are allowed to merge only if their emitted parts-of-speech are identical and if the parts-of-speech of their predecessors are identical, too. If applied between the two previously mentioned constraints the process of merging is sped up significantly but there is little difference in the derived HMMs.

### 1.3.3 Model Merging and Part-of-Speech Tagging

To use Markov models derived by Model Merging for part-of-speech tagging, we extend the model in such a way that the output no longer consists of a single word but of a word with its annotated part-of-speech. This is different from other HMM approaches where each state represents a particular category, and only words of this particular category are emitted by the state. We need this extension to allow all states to merge but at the same time to be able to identify the part-of-speech of an emitted word.

The task for part-of-speech tagging no longer is to find a particular sequence of states. Instead, we have to determine the sequence of tags $T = t_1 \dots t_k$ with the highest probability from all sequences of states $Q = q_1 \dots q_k$ that can output a given sequence of words $W = w_1 \dots w_k$. Thus, for a given sequence of words $W$ we have to find

$$\operatorname*{argmax}_T \sum_Q P(Q) \cdot P(W, T|Q).$$

Since most of the time there will be one main path through the model making the main contribution to the sum, it suffices to find the Viterbi approximation

$$\operatorname*{argmax}_T \max_Q P(Q) \cdot P(W, T|Q).$$

What follows immediately from the last formula is that it is useless to merge two states that output the same words but with two different categories when using that formula. Performing such a merge, the resulting model will always decide in favor of the part-of-speech with the higher probability and will never choose the other one. This means that merges of this kind should be excluded when using Viterbi approximation.

Two problems arise when actually using the derived models for part-of-speech tagging. First, a lot of sequences cannot be recognized, because there are unknown words in the sequence. Second, a lot of sequences cannot be recognized because even if all words are known there is no matching path in the HMM.

The problem of unknown words can be solved by introducing a mapping from the unknown words to known words and using the states emitting the known words for the unknown words. We use word suffixes for the mapping as introduced by Samuelsson 1993. An unknown word is supposed to be emitted by a state that can output a word with the same suffix.

The problem of non-existent paths can be solved by "smoothing" the transition probabilities between states. This can be done essentially in the same way as for HMMs derived with other methods (like expected likelihood estimation, Good-Turing-Estimation, or deleted interpolation).

## 1.4 Experiments

The experiments reported in this section were performed on three parts of the Susanne Corpus (Sampson 1995) using a tagset of 62 tags. We have chosen one training part (part A) and two test parts (parts B and C), they are mutually disjunct, and each part consists of approximately 10,000 words.

The results are compared with those of a standard trigram tagging procedure which serves as a baseline. The trigram tagger was trained on the same part as the HMM derived by Model Merging.

### 1.4.1 Tagging with a trivial HMM

A trivial model (maximum likelihood model) built from a corpus can be immediately used for tagging by adding transitions with small probabilities between all previously unconnected states. Of course, this yields very large HMMs and the application of the model is extremely slow even when narrowing the search space with a beam search, but surprisingly the tagging accuracy is higher than the accuracy of the standard trigram method (see table 2). The trivial HMM used in the experiment consists of 8900 states, identical initial and final states are already merged (this does neither influence the language generated by the HMM nor does it change the associated probabilities, as an example see Figure 3.a and b). The results for the trivial HMM are about 0.5% better than for the standard trigram tagger.

TABLE 2 Tagging with a trivial HMM

| training part | testing part | accuracy for known words | |
|---|---|---|---|
| | | trivial HMM | standard trigram |
| A | B | 95.17% | 94.69% |
| A | C | 95.53% | 95.17% |

TABLE 3 Tagging with merged HMMs

| training part | testing part | accuracy for known words, using HMMs after | | |
|---|---|---|---|---|
| | | 1500 merges | 3000 merges | 4000 merges |
| A | B | 95.28% | 95.42% | 95.35% |
| A | C | 95.59% | 95.72% | 95.69% |

### 1.4.2 Tagging with merged HMMs

For the next experiments, Model Merging was applied to the trivial model used for the previous experiment. Merging was restricted to states that output the same word for the first 1500 merges. Then, for the next 1500 merges, it was restricted to states that output words in the same part-of-speech bigram. The parts-of-speech of the merged state had to be identical as well as those of their predecessors. Then, for the last 1000 merges, merging was constrained to states that output words having the same part-of-speech. Deriving the HMM with 4000 merges took about 4 days on a SparcServer 1000. Table 3 shows the tagging results.

The tagging accuracy slightly increased within the first 3000 merges, then dropped a little bit after the last 1000 merges.

## 1.5 Conclusion

The first experiments with part-of-speech tagging using models derived by Model Merging are promising. The tagging results are all better than for a standard approach.

The advantage of Model Merging is that it offers both automatic generation of states (and thus categories) and reflecting the fact that a word can belong to more than one category. None of the other estimation techniques offers this combination. Model Merging is capable of estimating the model topology without restrictions.

The major difficulty with Model Merging is the enormous time needed for the process. We introduced three constraints that restricted the allowed merges to speed up the merging process. The constraints were applied one after each other, each was given up after a number of steps, and in the end all states were allowed to merge. This allowed the processing of a rather small training corpus of 10,000 words.

The constraints serve as a "guesser" which are the most promising pairs of states to merge. This guesser needs further investigation.

Future work includes the implementation of an incremental version of the algorithm, merging a corpus piecewise. This will speed up the merging and facilitates the processing of large corpora. With the incremental version the processing time mainly depends on the size of the increments and is linear in the number of increments.

## Acknowledgements

I would like to thank Christer Samuelsson for very useful comments on earlier versions of this paper. This work was supported by a studentship from the Graduiertenkolleg Kognitionswissenschaft, Saarbrücken.

## References

Baum, Leonard E., Ted Petrie, George Soules, and Norman Weiss. 1970. A maximization technique occuring in the statistical analysis of probabilistic functions in Markov chains. *The Annals of Methematical Statistics* 41:164–171.

Church, Kenneth Ward. 1988. A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. In *Proc. Second Conference on Applied Natural Language Processing*, 136–143. Austin, Texas, USA.

Cutting, Doug, Julian Kupiec, Jan Pedersen, and Penelope Sibun. 1992. A practical part-of-speech tagger. In *Proceedings of the 3rd Conference on Applied Natural Language Processing (ACL)*, 133–140.

DeRose, Steven J. 1988. Grammatical Category Disambiguation by Statistical Optimization. *Computational Linguistics* 14(1):31–39.

Omohundro, S. M. 1992. Best-First Model Merging for Dynamic Learning and Recognition. In *Advances in Neural Information Processing Systems 4*, ed. J. E. Moody, S. J. Hanson, and R. P. Lippmann. 958–965. San Mateo, CA: Kaufmann.

Pereira, Fernando, Naftali Tishby, and Lillian Lee. 1993. Distributional Clustering of English Words. In *Proceedings of the 31st ACL*. Columbus, Ohio.

Sampson, Geoffrey. 1995. *English for the Computer*. Oxford: Oxford University Press.

Samuelsson, Christer. 1993. Morphological Tagging Based Entirely on Bayesian Inference. In *9th Nordic Conference on Computational Linguistics*. Stockholm University, Stockholm, Sweden.

Stolcke, Andreas, and Stephen M. Omohundro. 1994. Best-first Model Merging for Hidden Markov Model Induction. Technical Report TR-94-003. Berkeley, California, USA: International Computer Science Institute.

Viterbi, A. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. In *IEEE Transactions on Information Theory*, 260–269.