Look-Back and Look-Ahead in the Conversion of Hidden Markov Models into Finite State Transducers

André Kempe
Xerox Research Centre Europe – Grenoble Laboratory
6, chemin de Maupertuis – 38240 Meylan – France
andre.kempe@xrce.xerox.com
http://www.xrce.xerox.com/research/mltt

Abstract

This paper describes the conversion of a Hidden Markov Model into a finite state transducer that closely approximates the behavior of the stochastic model. In some cases the transducer is equivalent to the HMM. This conversion is especially advantageous for part-of-speech tagging because the resulting transducer can be composed with other transducers that encode correction rules for the most frequent tagging errors. The speed of tagging is also improved. The described methods have been implemented and successfully tested.

1 Introduction

This paper presents an algorithm[1] which approximates a *Hidden Markov Model* (HMM) by a *finite-state transducer* (FST). We describe one application, namely part-of-speech tagging. Other potential applications may be found in areas where both HMMs and finite-state technology are applied, such as speech recognition, etc. The algorithm has been fully implemented.

An HMM used for tagging encodes, like a transducer, a relation between two languages. One language contains sequences of ambiguity classes obtained by looking up in a lexicon all words of a sentence. The other language contains sequences of tags obtained by statistically disambiguating the class sequences. From the outside, an HMM tagger behaves like a sequential transducer that deterministically maps every class sequence to a tag sequence, e.g.:

$$\frac{\texttt{[DET,PRO]}\ \texttt{[ADJ,NOUN]}\ \texttt{[ADJ,NOUN]}\ \ldots\ldots\ \texttt{[END]}}{\texttt{DET}\qquad \texttt{ADJ}\qquad \texttt{NOUN}\ \ \ldots\ldots\ \texttt{END}} \tag{1}$$

The main advantage of transforming an HMM is that the resulting transducer can be handled by finite state calculus. Among others, it can be composed with transducers that encode:

- correction rules for the most frequent tagging errors which are automatically generated (Brill, 1992; Roche and Schabes, 1995) or manually written (Chanod and Tapanainen, 1995), in order to significantly improve tagging accuracy[2]. These rules may include long-distance dependencies not handled by HMM taggers, and can conveniently be expressed by the replace operator (Kaplan and Kay, 1994; Karttunen, 1995; Kempe and Karttunen, 1996).
- further steps of text analysis, e.g. light parsing or extraction of noun phrases or other phrases (Aït-Mokhtar and Chanod, 1997).

These compositions enable complex text analysis to be performed by a single transducer.

The speed of tagging by an FST is up to six times higher than with the original HMM.

The motivation for deriving the FST from an HMM is that the HMM can be trained and converted with little manual effort.

An HMM transducer builds on the data (probability matrices) of the underlying HMM. The accuracy of this data has an impact on the tagging accuracy of both the HMM itself and the derived transducer. The training of the HMM can be done on either a tagged or untagged corpus, and is not a topic of this paper since it is exhaustively described in the literature (Bahl and Mercer, 1976; Church, 1988).

An HMM can be identically represented by a weighted FST in a straightforward way. We are, however, interested in non-weighted transducers.

[1] There are other (different) algorithms for HMM to FST conversion: An unpublished one by Julian M. Kupiec and John T. Maxwell (p.c.), and n-type and s-type approximation by Kempe (1997).

[2] Automatically derived rules require less work than manually written ones but are unlikely to yield better results because they would consider relatively limited context and simple relations only.

André Kempe (1998) **Look-Back and Look-Ahead in the Conversion of Hidden Markov Models into Finite State Transducers.** In D.M.W. Powers (ed.) *NeMLaP3/CoNLL98: New Methods in Language Processing and Computational Natural Language Learning,* ACL, pp 29-37.

2 b-Type Approximation

This section presents a method that approximates a (first order) Hidden Markov Model (HMM) by a finite-state transducer (FST), called *b-type* approximation[3]. Regular expression operators used in this section are explained in the annex.

Looking up, in a lexicon, the word sequence of a sentence produces a unique sequence of ambiguity classes. Tagging the sentence by means of a (first order) HMM consists of finding the most probable tag sequence T given this class sequence C (eq. 1, fig. 1). The joint probability of the sequences C and T can be estimated by:

$$p(C,T) = p(c_1 \ldots c_n, t_1 \ldots t_n) =$$
$$\pi(t_1)\ b(c_1|t_1) \cdot \prod_{i=2}^{n} a(t_i|t_{i-1})\ b(c_i|t_i) \qquad (2)$$

2.1 Basic Idea

The determination of a tag of a particular word cannot be made separately from the other tags. Tags can influence each other over a long distance via transition probabilities.

In this approach, an ambiguity class is disambiguated with respect to a context. A context consists of a sequence of ambiguity classes limited at both ends by some selected tag[4]. For the left context of length β we use the term *look-back*, and for the right context of length α we use the term *look-ahead*.

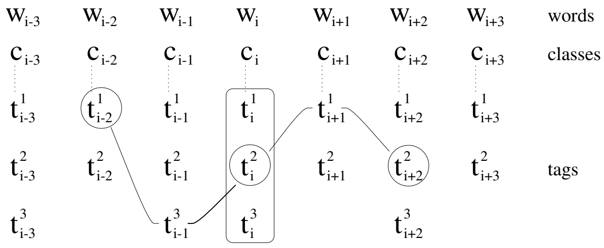

Figure 1: Disambiguation of classes between two selected tags

In figure 1, the tag t_i^2 can be selected from the class c_i because it is between two selected tags[4] which are t^1_{i-2} at a look-back distance of $\beta = 2$ and t^2_{i+2} at a look-ahead distance of $\alpha = 2$. Actually, the two selected tags t^1_{i-2} and t^2_{i+2} allow not only the disambiguation of the class c_i but of all classes inbetween, i.e. c_{i-1}, c_i and c_{i+1}.

We approximate the tagging of a whole sentence by tagging subsequences with selected tags at both ends (fig. 1), and then overlapping them. The most probable paths in the tag space of a sentence, i.e. valid paths according to this approach, can be found as sketched in figure 2.

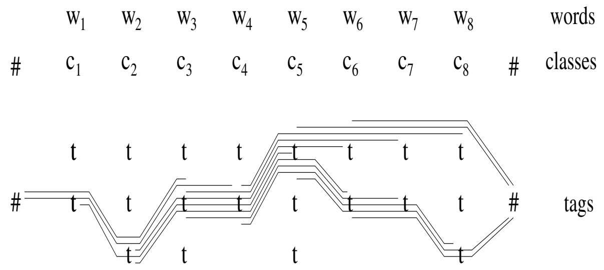

Figure 2: Two valid paths through the tag space of a sentence

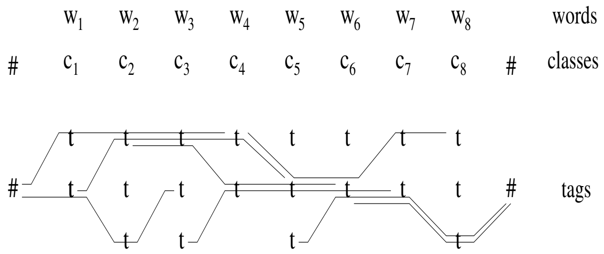

Figure 3: Incompatible sequences in the tag space of a sentence

A valid path consists of an ordered set of overlapping sequences in which each member overlaps with its neighbour except for the first or last tag. There can be more than one valid path in the tag space of a sentence (fig. 2). Sets of sequences that do not overlap in such a way are incompatible according to this model, and do not constitute valid paths (fig. 3).

2.2 b-Type Sequences

Given a length β of look-back and a length α of look-ahead, we generate for every class c_0, every look-back sequence $t_{-\beta}\ c_{-\beta+1}\ \ldots\ c_{-1}$, and every look-ahead sequence $c_1\ \ldots\ c_{\alpha-1}\ t_\alpha$, a b-type sequence[4]:

$$t_{-\beta}\ c_{-\beta+1} \ldots c_{-1}\ \ c_0\ \ c_1 \ldots c_{\alpha-1}\ t_\alpha \qquad (3)$$

[3] Name given by the author, to distinguish the algorithm from n-type and s-type approximation (Kempe, 1997).

[4] The algorithm is explained for a first order HMM. In the case of a second order HMM, b-type sequences must begin and end with two selected tags rather than one.

For example:

$$\texttt{CONJ [DET,PRON] [ADJ,NOUN,VERB] [NOUN,VERB] VERB} \quad (4)$$

Each such *original b-type sequence* (eq. 3,4; fig. 4) is disambiguated based on a first order HMM. Here we use the Viterbi algorithm (Viterbi, 1967; Rabiner, 1990) for efficiency.

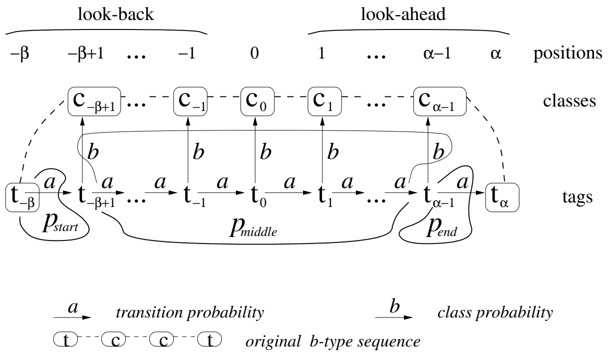

Figure 4: b-Type sequence

For an original b-type sequence, the joint probability of its class sequence C with its tag sequence T (fig. 4), can be estimated by:

$$p(C,T) = p(c_{-\beta+1} \ldots c_{\alpha-1}\ ,\ t_{-\beta} \ldots t_{\alpha}) = \left[\prod_{i=-\beta+1}^{\alpha-1} a(t_i|t_{i-1})\ b(c_i|t_i)\right] \cdot a(t_\alpha|t_{\alpha-1}) \quad (5)$$

At every position in the look-back sequence and in the look-ahead sequence, a boundary # may occur, i.e. a sentence beginning or end. No look-back ($\beta = 0$) or no look-ahead ($\alpha = 0$) is also allowed. The above probability estimation (eq. 5) can then be expressed more generally (fig. 4) as:

$$p(C,T) = p_{start} \cdot p_{middle} \cdot p_{end} \quad (6)$$

with p_{start} being

$$p_{start} = a(t_{-\beta+1}|t_{-\beta}) \quad \text{for selected tag } t_{-\beta} \quad (7)$$
$$p_{start} = \pi(t_{-\beta+1}) \quad \text{for boundary \#} \quad (8)$$
$$p_{start} = 1 \quad \text{for } \beta=0 \quad (9)$$

with p_{middle} being

$$p_{middle} = b(c_{-\beta+1}|t_{-\beta+1}) \cdot \prod_{i=-\beta+2}^{\alpha-1} a(t_i|t_{i-1})\ b(c_i|t_i) \quad \text{for } \alpha+\beta>0 \quad (10)$$
$$p_{middle} = b(c_0|t_0) \quad \text{for } \alpha+\beta=0 \quad (11)$$

and with p_{end} being

$$p_{end} = a(t_\alpha|t_{\alpha-1}) \quad \text{for selected tag } t_\alpha \quad (12)$$
$$p_{end} = 1 \quad \text{for boundary \# or } \alpha=0 \quad (13)$$

When the most likely tag sequence is found for an original b-type sequence, the class c_0 in the middle position (eq. 3) is associated with its most likely tag t_0. We formulate constraints for the other tags $t_{-\beta}$ and t_α and classes $c_{-\beta+1} \ldots c_{-1}$ and $c_1 \ldots c_{\alpha-1}$ of the original b-type sequence. Thus we obtain a *tagged b-type sequence* [5]:

$$t^{B\beta}_{-\beta}\ c^{B(\beta-1)}_{-\beta+1} \ldots c^{B2}_{-2}\ c^{B1}_{-1}\ \ c_0{:}t_0\ \ c^{A1}_{1}\ c^{A2}_{2} \ldots c^{A(\alpha-1)}_{\alpha-1}\ t^{A\alpha}_{\alpha} \quad (14)$$

stating that t_0 is the most probable tag in the class c_0 if it is preceded by $t^{B\beta}\ c^{B(\beta-1)} \ldots c^{B2}\ c^{B1}$ and followed by $c^{A1}\ c^{A2} \ldots c^{A(\alpha-1)}\ t^{A\alpha}$.

In expression 14 the subscripts $-\beta\ -\beta+1 \ldots 0 \ldots \alpha-1\ \alpha$ denote the position of the tag or class in the b-type sequence, and the superscripts $B\beta\ B(\beta-1) \ldots B1$ and $A1 \ldots A(\alpha-1)\ A\alpha$ express constraints for preceding and following tags and classes which are part of other b-type sequences. In the example[5]:

```
CONJ–B2 [DET,PRON]–B1
        [ADJ,NOUN,VERB]:ADJ
                [NOUN,VERB]–A1 VERB–A2     (15)
```

`ADJ` is the most likely tag in the class `[ADJ,NOUN,VERB]` if it is preceded by the tag `CONJ` two positions back (`B2`), by the class `[DET,PRON]` one position back (`B1`), and followed by the class `[NOUN,VERB]` one position ahead (`A1`) and by the tag `VERB` two positions ahead (`A2`).

Boundaries are denoted by a particular symbol # and can occur at the edge of the look-back and look-ahead sequence:

$$t^{B\beta}\ c^{B(\beta-1)} \ldots c^{B2}\ c^{B1}\ \ c{:}t\ \ c^{A1}\ c^{A1} \ldots c^{A(\alpha-1)}\ \#^{A\alpha} \quad (16)$$
$$t^{B\beta}\ c^{B(\beta-1)} \ldots c^{B2}\ c^{B1}\ \ c{:}t\ \ c^{A1}\ c^{A1} \ldots \#^{A(\alpha-1)} \quad (17)$$
$$\#^{B\beta}\ c^{B(\beta-1)} \ldots c^{B2}\ c^{B1}\ \ c{:}t\ \ \#^{A1} \quad (18)$$
$$\#^{B1}\ \ c{:}t\ \ \#^{A1} \quad (19)$$
$$\#^{B2}\ c^{B1}\ \ c{:}t\ \ c^{A1}\ c^{A1} \ldots c^{A(\alpha-1)}\ t^{A\alpha} \quad (20)$$

For example:

```
#–B2 [DET,PRON]–B1
        [ADJ,NOUN,VERB]:ADJ
                [NOUN,VERB]–A1 VERB–A2     (21)
```

[5] Regular expression operators used in this article are explained in the annex.

```
CONJ-B2 [DET,PRON]-B1
              [ADJ,NOUN,VERB]:NOUN
                          #-A1
```
$$(22)$$

Note that look-back of length β and look-ahead of length α also include all sequences shorter than β or α, respectively, that are limited by #.

For a given length β of look-back and a length α of look-ahead, we generate every possible original b-type sequence (eq. 3), disambiguate it statistically (eq. 5-13), and encode the tagged b-type sequence B_i (eq. 14) as an FST. All sequences B_i are then unioned

$$^{\cup}B = \bigcup_i B_i \tag{23}$$

and we generate a preliminary tagger model B'

$$B' = [\ ^{\cup}B\ ]* \tag{24}$$

where all sequences B_i can occur in any order and number (including zero times) because no constraints have yet been applied.

2.3 Concatenation Constraints

To ensure a correct concatenation of sequences B_i, we have to make sure that every B_i is preceded and followed by other B_i according to what is encoded in the look-back and look-ahead constraints. E.g. the sequence in example (21) must be preceded by a sentence beginning, #, and the class `[DET,PRON]` and followed by the class `[NOUN,VERB]` and the tag `VERB`.

We create constraints for preceding and following tags, classes and sentence boundaries. For the look-back, a particular tag t_i or class c_j is required for a particular distance of $\delta \leq -1$, by[5]:

$$R^{\delta}(t_i) = {\sim}[\ {\sim}[\ ?*\ \ t_i\ [\backslash{}^{\cup}t]*\ \ [{}^{\cup}t\ [\backslash{}^{\cup}t]*]\,\text{\textasciicircum}(\text{-}\delta\text{-}1)\,]\ t_i^{B(-\delta)}\ \ ?*\,] \tag{25}$$

$$R^{\delta}(c_j) = {\sim}[\ {\sim}[\ ?*\ \ c_j\ [\backslash{}^{\cup}c]*\ \ [{}^{\cup}c\ [\backslash{}^{\cup}c]*]\,\text{\textasciicircum}(\text{-}\delta\text{-}1)\,]\ c_j^{B(-\delta)}\ \ ?*\,] \tag{26}$$

$$\text{for} \quad \delta \leq -1$$

with $^{\cup}t$ and $^{\cup}c$ being the union of all tags and all classes respectively.

A sentence beginning, #, is required for a particular look-back distance of $\delta \leq -1$, on the side of the tags, by:

$$R^{\delta}(\#) = {\sim}[\ {\sim}[\ [\backslash{}^{\cup}t]*\ \ [{}^{\cup}t\ [\backslash{}^{\cup}t]*]\,\text{\textasciicircum}(\text{-}\delta\text{-}1)\,]\ \#^{B(-\delta)}\ \ ?*\,] \tag{27}$$

$$\text{for} \quad \delta \leq -1$$

In the case of look-ahead we require for a particular distance of $\delta \geq 1$, a particular tag t_i or class c_j or a sentence end, #, on the side of the tags, in a similar way by:

$$R^{\delta}(t_i) = {\sim}[\ ?*\ \ t_i^{A\delta}\ {\sim}[\ [\backslash{}^{\cup}t]*\ \ [{}^{\cup}t\ [\backslash{}^{\cup}t]*]\,\text{\textasciicircum}(\delta-1)\ t_i\ \ ?*\,]\,] \tag{28}$$

$$R^{\delta}(c_j) = {\sim}[\ ?*\ \ c_j^{A\delta}\ {\sim}[\ [\backslash{}^{\cup}c]*\ \ [{}^{\cup}c\ [\backslash{}^{\cup}c]*]\,\text{\textasciicircum}(\delta-1)\ c_j\ \ ?*\,]\,] \tag{29}$$

$$R^{\delta}(\#) = {\sim}[\ ?*\ \ \#^{A\delta}\ {\sim}[\ [\backslash{}^{\cup}t]*\ \ [{}^{\cup}t\ [\backslash{}^{\cup}t]*]\,\text{\textasciicircum}(\delta-1)\,]\,] \tag{30}$$

$$\text{for} \quad \delta \geq 1$$

All tags t_i are required for the look-back only at the distance of $\delta = -\beta$ and for the look-ahead only at the distance of $\delta = \alpha$. All classes c_j are required for distances of $\delta \in [-\beta+1, -1]$ and $\delta \in [1, \alpha-1]$. Sentence boundaries, #, are required for distances of $\delta \in [-\beta, -1]$ and $\delta \in [1, \alpha]$.

We create the intersection R_t of all tag constraints, the intersection R_c of all class constraints, and the intersection $R_{\#}$ of all sentence boundary constraints:

$$R_t \quad = \quad \bigcap_{\substack{i\,\in\,[1,n] \\ \delta\,\in\,\{-\beta,\alpha\}}} R^{\delta}(t_i) \tag{31}$$

$$R_c \quad = \quad \bigcap_{\substack{j\,\in\,[1,m] \\ \delta\,\in\,[-\beta+1,-1]\cup[1,\alpha-1]}} R^{\delta}(c_j) \tag{32}$$

$$R_{\#} \quad = \quad \bigcap_{\delta\,\in\,[-\beta,-1]\cup[1,\alpha]} R^{\delta}(\#) \tag{33}$$

All constraints are enforced by composition with the preliminary tagger model B' (eq. 24). The class constraint R_c is composed on the upper side of B' which is the side of the classes (eq. 14), and both the tag constraint R_t and the boundary constraint[6] $R_{\#}$ are composed on the lower side of B', which is the side of the tags[5]:

$$B'' = R_c \ .\text{o}.\ B' \ .\text{o}.\ R_t \ .\text{o}.\ R_{\#} \tag{34}$$

Having ensured correct concatenation, we delete all symbols r that have served to constrain tags, classes or boundaries, using D_r:

$$r \quad = \quad \left[\bigcup_{i,\delta} t_i^{\delta}\right] \cup \left[\bigcup_{j,\delta} c_j^{\delta}\right] \cup \left[\bigcup_{\delta} \#^{\delta}\right] \tag{35}$$

[6] The boundary constraint $R_{\#}$ could alternatively be computed for and composed on the side of the classes. The transducer which encodes $R_{\#}$ would then, however, be bigger because the number of classes is bigger than the number of tags.

$$D_r \quad = \quad r \texttt{->} [\,] \tag{36}$$

By composing[7] B'' (eq. 34) on the lower side with D_r and on the upper side with the inverted relation D_r`.i`, we obtain the final tagger model B:

$$B \;=\; D_r.\texttt{i} \;\texttt{.o.}\; B'' \;\texttt{.o.}\; D_r \tag{37}$$

We call the model a *b-type model*, the corresponding FST a *b-type transducer*, and the whole algorithm leading from the HMM to the transducer, a *b-type approximation* of an HMM.

2.4 Properties of b-Type Transducers

There are two groups of b-type transducers with different properties: FSTs without look-back and/or without look-ahead ($\beta \cdot \alpha = 0$) and FSTs with both look-back and look-ahead ($\beta \cdot \alpha > 0$). Both accept any sequence of ambiguity classes.

b-Type FSTs with $\beta \cdot \alpha = 0$ are always sequential. They map a class sequence that corresponds to the word sequence of a sentence, always to exactly one tag sequence. Their tagging accuracy and similarity with the underlying HMM increases with growing $\beta + \alpha$. A b-type FST with $\beta = 0$ and $\alpha = 0$ is equivalent to an *n0-type* FST, and with $\beta = 1$ and $\alpha = 0$ it is equivalent to an *n1-type* FST (Kempe, 1997).

b-Type FSTs with $\beta \cdot \alpha > 0$ are in general not sequential. For a class sequence they deliver a set of different tag sequences, which means that the tagging results are ambiguous. This set is never empty, and the most probable tag sequence according to the underlying HMM is always in this set. The longer the look-back distance β and the look-ahead distance α are, the larger the FST and the smaller the set of resulting tag sequences. For sufficiently large $\beta + \alpha$, this set may contain always only one tag sequence. In this case the FST is equivalent to the underlying HMM. For reasons of size however, this FST may not be computable for particular HMMs (sec. 4).

3 An Implemented Finite-State Tagger

The implemented tagger requires three transducers which represent a lexicon, a guesser and an approximation of an HMM mentioned above.

Both the lexicon and guesser are sequential, i.e. deterministic on the input side. They both unambiguously map a surface form of any word that they accept to the corresponding ambiguity class (fig. 5, col. 1 and 2): First of all, the word is looked for in the lexicon. If this fails, it is looked for in the guesser. If this equally fails, it gets the label `[UNKNOWN]` which denotes the ambiguity class of unknown words. Tag probabilities in this class are approximated by tags of words that appear only once in the training corpus.

As soon as an input token gets labeled with the tag class of sentence end symbols (fig. 5: `[SENT]`), the tagger stops reading words from the input. At this point, the tagger has read and stored the words of a whole sentence (fig. 5, col. 1) and generated the corresponding sequence of classes (fig. 5, col. 2).

The class sequence is now mapped to a tag sequence (fig. 5, col. 3) using the HMM transducer. A b-type FST is not sequential in general (sec. 2.4), so to obtain a unique tagging result, the finite-state tagger can be run in a special mode, where only the first result found is retained, and the tagger does not look for other results[8]. Since paths through an FST have no particular order, the result retained is random.

The tagger outputs the stored word and tag sequence of the sentence, and continues in the same way with the remaining sentences of the corpus.

`The`	`[AT]`	`AT`
`share`	`[NN,VB]`	`NN`
`of`	`[IN]`	`IN`
...	...	...
`tripled`	`[VBD,VBN]`	`VBD`
`within`	`[IN,RB]`	`IN`
`that`	`[CS,DT,WPS]`	`DT`
`span`	`[NN,VB,VBD]`	`NN`
`of`	`[IN]`	`IN`
`time`	`[NN,VB]`	`NN`
`.`	`[SENT]`	`SENT`

Figure 5: Tagging a sentence

The tagger can be run in a statistical mode where the number of tag sequences found per sentence is counted. These numbers give an overview of the degree of non-sequentiality of the concerned b-type transducer (sec. 2.4).

[7]For efficiency reasons, we actually do not delete the constraint symbols r by composition. We rather traverse the network, and overwrite every symbol r with the empty string symbol ϵ. In the following determinization of the network, all ϵ are eliminated.

[8]This mode of retaining the first result only is not necessary with n-type and s-type transducers which are both sequential (Kempe, 1997).

Transducer or HMM	Accuracy test corp. in %	Tagging speed in words/sec		Transducer size		Creation time
		ultra2	sparc20	#states	#arcs	ultra2
HMM	97.35	4 834	1 624	———	———	———
s+n1-FST (1M, F1)	97.33	19 939	8 986	9 419	1 154 225	22 min
s+n1-FST (1M, F8)	96.12	22 001	9 969	329	42 560	4 min
b-FST (β=0, α=0), =n0	87.21	26 585	11 000	1	181	6 sec
b-FST (β=1, α=0), =n1	95.16	26 585	11 600	37	6 697	11 sec
b-FST (β=2, α=0)	95.32	21 268	7 089	3 663	663 003	4 h 11
b-FST (β=0, α=1)	93.69	19 939	7 877	252	40 243	12 sec
b-FST (β=0, α=2)	93.92	19 334	9 114	10 554	1 246 686	10 min
b-FST (β=1, α=1)	*95.78	16 360	7 506	3 514	640 336	56 sec
b-FST (β=2, α=1)	*97.34	15 191	6 510	54 578	8 402 055	2 h 17
b-FST (β=3, α=1)	FST was not computable					

Language:	English
Corpora:	19 944 words for HMM training, 19 934 words for test
Tag set:	36 tags, 181 classes
*	Multiple, i.e. ambiguous tagging results: Only first result retained
Types of FST (Finite-State Transducers) :	
n0, n1	n-type transducers (Kempe, 1997)
s+n1 (1M,F8)	s-type transducer (Kempe, 1997), with subsequences of frequency $\geq$ 8, from a training corpus of 1 000 000 words, completed with n1-type
b (β=2, α=1)	b-type transducer (sec. 2), with look-back of 2 and look-ahead of 1
Computers:	
ultra2	1 CPU, 512 MBytes physical RAM, 1.4 GBytes virtual RAM
sparc20	1 CPU, 192 MBytes physical RAM, 827 MBytes virtual RAM

Table 1: Accuracy, speed, size and creation time of some HMM transducers

4 Experiments and Results

This section compares different FSTs with each other and with the original HMM.

As expected, the FSTs perform tagging faster than the HMM.

Since all FSTs are approximations of HMMs, they show lower tagging accuracy than the HMMs. In the case of FSTs with $\beta \geq 1$ and $\alpha = 1$, this difference in accuracy is negligible. Improvement in accuracy can be expected since these FSTs can be composed with FSTs encoding correction rules for frequent errors (sec. 1).

For all tests below an English corpus, lexicon and guesser were used, which were originally annotated with 74 different tags. We automatically recoded the tags in order to reduce their number, i.e. in some cases more than one of the original tags were recoded into one and the same new tag. We applied different recodings, thus obtaining English corpora, lexicons and guessers with reduced tag sets of 45, 36, 27, 18 and 9 tags respectively.

FSTs with $\beta = 2$ and $\alpha = 1$ and with $\beta = 1$ and $\alpha = 2$ were equivalent, in all cases where they could be computed.

Table 1 compares different FSTs for a tag set of 36 tags.

The b-type FST with no look-back and no look-ahead which is equivalent to an n0-type FST (Kempe, 1997), shows the lowest tagging accuracy (b-FST ($\beta = 0, \alpha = 0$): 87.21 %). It is also the smallest transducer (1 state and 181 arcs, as many as tag classes) and can be created faster than the other FSTs (6 sec.).

The highest accuracy is obtained with a b-type FST with $\beta = 2$ and $\alpha = 1$ (b-FST ($\beta = 2, \alpha = 1$): 97.34 %) and with an *s-type* FST (Kempe, 1997) trained on 1 000 000 words (s+n1-FST (1M, F1): 97.33 %). In these two cases the difference in accuracy with respect to the underlying HMM (97.35 %) is negligible. In this particular test, the s-type FST comes out ahead because it is considerably smaller than the b-type FST.

The size of a b-type FST increases with the size of the tag set and with the length of look-back plus look-ahead, $\beta + \alpha$. Accuracy improves with growing $\beta + \alpha$.

b-Type FSTs may produce ambiguous tagging results (sec. 2.4). In such instances only the first result was retained (sec. 3).

Transducer or HMM	Tagging accuracy and agreement with the HMM for tag sets of different sizes					
	74 tags 297 cls.	45 tags 214 cls.	36 tags 181 cls.	27 tags 119 cls.	18 tags 97 cls.	9 tags 67 cls.
HMM	96.78	96.92	97.35	97.07	96.73	95.76
s+n1 FST (1M, F1)	96.76 99.89	96.88 99.93	97.33 99.90	97.06 99.95	96.72 99.95	95.74 99.94
s+n1-FST (1M, F8)	95.09 97.00	95.25 97.35	96.12 98.15	96.36 98.90	96.05 98.99	95.29 98.96
b-FST (β=0, α=0), =n0	83.53 84.00	83.71 84.40	87.21 88.04	94.47 96.03	94.24 96.22	93.86 95.76
b-FST (β=1, α=0), =n1	94.19 95.61	94.09 95.92	95.16 96.90	95.60 97.75	95.17 97.66	94.14 96.74
b-FST (β=2, α=0)	———	94.28 96.09	95.32 97.01	95.71 97.84	95.31 97.77	94.22 96.83
b-FST (β=0, α=1)	92.79 93.64	92.47 93.41	93.69 94.67	95.26 96.87	95.19 97.06	94.64 97.09
b-FST (β=0, α=2)	93.46 94.35	92.77 93.70	93.92 94.90	95.37 96.99	95.30 97.20	94.80 97.29
b-FST (β=1, α=1)	*94.94 *97.86	*95.14 *97.93	*95.78 *98.11	*96.78 *99.58	*96.59 *99.72	*95.36 *99.26
b-FST (β=2, α=1)	———	———	*97.34 *99.97	*97.06 *99.98	*96.73 *100.00	*95.73 *99.97
b-FST (β=3, α=1)	———	———	———	———	———	95.76 100.00

Language:	English
Corpora:	19 944 words for HMM training, 19 934 words for test
Types of FST (Finite-State Transducers)	cf. table 1
*	Multiple, i.e. ambiguous tagging results: Only first result retained
97.06 99.98	Tagging accuracy of 97.06 %, and agreement of FST with HMM tagging results of 99.98 %
———	Transducer could not be computed, for reasons of size.

Table 2: Tagging accuracy and agreement of the FST tagging results with those of the underlying HMM, for tag sets of different sizes

Table 2 shows the tagging accuracy and the agreement of the tagging results with the results of the underlying HMM for different FSTs and tag sets of different sizes.

To get results that are almost equivalent to those of an HMM, a b-type FST needs at least a look-back of $\beta = 2$ and a look-ahead of $\alpha = 1$ or vice versa. For reasons of size, this kind of FST could only be computed for tag sets with 36 tags or less. A b-type FST with $\beta = 3$ and $\alpha = 1$ could only be computed for the tag set with 9 tags. This FST gave exactly the same tagging results as the underlying HMM.

Table 3 illustrates which of the b-type FSTs are sequential, i.e. always produce exactly one tagging result, and which of the FSTs are non-sequential.

For all tag sets, the FSTs with no look-back ($\beta = 0$) and/or no look-ahead ($\alpha = 0$) behaved sequentially. Here 100 % of the tagged sentences had only one result. Most of the other FSTs ($\beta \cdot \alpha > 0$) behaved non-sequentially. For example, in the case of 27 tags with $\beta = 1$ and $\alpha = 1$, 90.08 % of the tagged sentences had one result, 9.46 % had two results, 0.23 % had tree results, etc.

Non-sequentiality decreases with growing look-back and look-ahead, $\beta + \alpha$, and should completely disappear with sufficiently large $\beta + \alpha$. Such b-type FSTs can, however, only be computed for small tag sets. We could compute this kind of FST only for the case of 9 tags with $\beta = 3$ and $\alpha = 1$.

The set of alternative tag sequences for a sentence, produced by a b-type FST with $\beta \cdot \alpha > 0$, always contains the tag sequence that corresponds with the result of the underlying HMM.

Transducer	Sentences with n tagging results (in %)					
	n= 1	n= 2	n= 3	n= 4	5-8	9-16
74 tags, 297 classes (original tag set)						
b-FST ($\beta\cdot\alpha=0$)	100					
b-FST (β=1,α=1)	75.14	20.18	0.34	3.42	0.80	0.11
b-FST (β=2,α=1)	FST was not computable					
45 tags, 214 classes (reduced tag set)						
b-FST ($\beta\cdot\alpha=0$)	100					
b-FST (β=1,α=1)	75.71	19.73	0.68	3.19	0.68	
b-FST (β=2,α=1)	FST was not computable					
36 tags, 181 classes (reduced tag set)						
b-FST ($\beta\cdot\alpha=0$)	100					
b-FST (β=1,α=1)	78.56	17.90	0.34	2.85	0.34	
b-FST (β=2,α=1)	99.77	0.23				
27 tags, 119 classes (reduced tag set)						
b-FST ($\beta\cdot\alpha=0$)	100					
b-FST (β=1,α=1)	90.08	9.46	0.23	0.11	0.11	
b-FST (β=2,α=1)	99.77	0.23				
18 tags, 97 classes (reduced tag set)						
b-FST ($\beta\cdot\alpha=0$)	100					
b-FST (β=1,α=1)	93.04	6.84		0.11		
b-FST (β=2,α=1)	99.89	0.11				
9 tags, 67 classes (reduced tag set)						
b-FST ($\beta\cdot\alpha=0$)	100					
b-FST (β=1,α=1)	86.66	12.43		0.91		
b-FST (β=2,α=1)	99.77	0.23				
b-FST (β=3,α=1)	100					

Language:	English
Test corpus:	19 934 words, 877 sentences
Types of FST (Finite-State Transducers)	cf. table 1

Table 3: Percentage of sentences with a particular number of tagging results

5 Conclusion and Future Research

The algorithm presented in this paper describes the construction of a finite-state transducer (FST) that approximates the behaviour of a Hidden Markov Model (HMM) in part-of-speech tagging.

The algorithm, called b-type approximation, uses look-back and look-ahead of freely selectable length.

The size of the FSTs grows with both the size of the tag set and the length of the look-back plus look-ahead. Therefore, to keep the FST at a computable size, an increase in the length of the look-back or look-ahead, requires a reduction of the number of tags. In the case of small tag sets (e.g. 36 tags), the look-back and look-ahead can be sufficiently large to obtain an FST that is almost equivalent to the original HMM.

In some tests s-type FSTs (Kempe, 1997) and b-type FSTs reached equal tagging accuracy. In these cases s-type FSTs are smaller because they encode the most frequent ambiguity class sequences of a training corpus very accurately and all other sequences less accurately. b-Type FSTs encode all sequences with the same accuracy. Therefore, a b-type FST can reach equivalence with the original HMM, but an s-type FST cannot.

The algorithms of both conversion and tagging are fully implemented.

The main advantage of transforming an HMM is that the resulting FST can be handled by finite state calculus[9] and thus be directly composed with other FSTs.

The tagging speed of the FSTs is up to six times higher than the speed of the original HMM.

Future research will include the composition of HMM transducers with, among others:

- FSTs that encode correction rules for the most frequent tagging errors in order to significantly improve tagging accuracy (above the accuracy of the underlying HMM). These rules can either be extracted automatically from a corpus (Brill, 1992) or written manually (Chanod and Tapanainen, 1995).

- FSTs for light parsing, phrase extraction and other text analysis (Aït-Mokhtar and Chanod, 1997).

An HMM transducer can be composed with one or more of these FSTs in order to perform complex text analysis by a single FST.

ANNEX: Regular Expression Operators

Below, `a` and `b` designate symbols, `A` and `B` designate languages, and `R` and `Q` designate relations between two languages. More details on the following operators and pointers to finite-state literature can be found in
`http://www.xrce.xerox.com/research/mltt/fst`

`~A`	*Complement* (negation). Set of all strings except those from the language `A`.
`\a`	*Term complement.* Any symbol other than `a`.
`A*`	*Kleene star.* Language `A` zero or more times concatenated with itself.
`A^n`	*A n times.* Language `A` n times concatenated with itself.
`a -> b`	*Replace.* Relation where every `a` on the upper side gets mapped to a `b` on the lower side.

[9] A large library of finite-state functions is available at Xerox.

`a:b`	*Symbol pair* with `a` on the upper and `b` on the lower side.
`R.i`	*Inverse* relation where both sides are exchanged with respect to `R`.
`A B`	*Concatenation* of all strings of `A` with all strings of `B`.
`R .o. Q`	*Composition* of the relations `R` and `Q`.
`0` *or* `[]`	*Empty string* (epsilon).
`?`	*Any symbol* in the known alphabet and its extensions

Acknowledgements

I am grateful to all colleagues who helped me, particularly to Lauri Karttunen (XRCE Grenoble) for extensive discussion, and to Julian Kupiec (Xerox PARC) for sending me information on his own related work. Many thanks to Irene Maxwell for correcting various versions of the paper.

References

Aït-Mokhtar, Salah and Chanod, Jean-Pierre (1997). Incremental Finite-State Parsing. In the *Proceedings of the 5th Conference of Applied Natural Language Processing (ANLP).* ACL, pp. 72-79. Washington, DC, USA.

Bahl, Lalit R. and Mercer, Robert L. (1976). Part of Speech Assignment by a Statistical Decision Algorithm. In *IEEE international Symposium on Information Theory*. pp. 88-89. Ronneby.

Brill, Eric (1992). A Simple Rule-Based Part-of-Speech Tagger. In the *Proceedings of the 3rd conference on Applied Natural Language Processing*, pp. 152-155. Trento, Italy.

Chanod, Jean-Pierre and Tapanainen, Pasi (1995). Tagging French - Comparing a Statistical and a Constraint Based Method. In the *Proceedings of the 7th conference of the EACL*, pp. 149-156. ACL. Dublin, Ireland. `cmp-lg/9503003`

Church, Kenneth W. (1988). A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. In *Proceedings of the 2nd Conference on Applied Natural Language Processing*. ACL, pp. 136-143.

Kaplan, Ronald M. and Kay, Martin (1994). Regular Models of Phonological Rule Systems. In *Computational Linguistics*. 20:3, pp. 331-378.

Karttunen, Lauri (1995). The Replace Operator. In the *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*. Cambridge, MA, USA. `cmp-lg/9504032`

Kempe, André and Karttunen, Lauri (1996). Parallel Replacement in Finite State Calculus. In the *Proceedings of the 16th International Conference on Computational Linguistics*, pp. 622-627. Copenhagen, Denmark. `cmp-lg/9607007`

Kempe, André (1997). Finite State Transducers Approximating Hidden Markov Models. In the *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pp. 460-467. Madrid, Spain. `cmp-lg/9707006`

Rabiner, Lawrence R. (1990). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In *Readings in Speech Recognition* (eds. A. Waibel, K.F. Lee). Morgan Kaufmann Publishers, Inc. San Mateo, CA., USA.

Roche, Emmanuel and Schabes, Yves (1995). Deterministic Part-of-Speech Tagging with Finite-State Transducers. In *Computational Linguistics*. Vol. 21, No. 2, pp. 227-253.

Viterbi, A.J. (1967). Error Bounds for Convolutional Codes and an Asymptotical Optimal Decoding Algorithm. In *Proceedings of IEEE*, vol. 61, pp. 268-278.