550, rue Sherbrooke Ouest, bureau 100
Montréal (Québec) H3A 1B9
Tél. : (514) 840-1234; Téléc. : (514) 840-1244
888, rue St-Jean, bureau 555
Québec (Québec) G1R 5H6
Tél. : (418) 648-8080; Téléc. : (418) 648-8141
http://www.crim.ca

CRIM - Documentation/Communications

E-Inclusion Core Speech

Direct Generation of States to Phones FST

Technical Report

CRIM-06/05-03

Pierre Ouellet
Senior Research Agent
Speech Recognition Group

May 2006

Collection scientifique et technique

ISBN-13: 978-2-89522-072-5
ISBN-10: 2-89522-072-7

Contents

1 Introduction

For many years at CRIM, researchers have been using Weighted Finite-State Transducers (abbreviated as WFST) for representing the statistical information used in Automatic Speech Recognition (ASR). WFST can be combined and optimized to produce a fully-integrated constraint network with full cross-word contexts. Such a network can then be used for guiding the ASR Viterbi search using relatively simple software, thanks to the uniform representation made possible by WFST. The construction of WFST must be carefully engineered: even though there may be multiple theoretically equivalent ways of producing a WFST, some methods are in pratice unworkable because they require too much memory or too much CPU time. Any method which improves the process (by requiring less memory, producing smaller results, etc.) is interesting because in the end it may allow for larger statistical models to be used for ASR. In [1], a promising method is described which has a definite benefit and a potential one for the CRIM ASR system:

1. It makes more complex and accurate acoustic modeling possible.
2. It may reduce the size of WFST and/or the amount of computer resources to produce them.

Currently, we cannot evaluate (1) since we do not have the tools for producing more complex acoustic models. So for the moment, we will focus on (2). This report has two main parts. Sections up to Section 5 contain an introduction to the necessary ASR concepts and to the WFST construction process. Section 6 covers the implementation details of the method.

2 Context-dependent acoustic phonetic modeling

2.1 Introduction

Typical Large Vocabulary Continuous Speech Recognition (LVCSR) systems use context-dependent (CD) phones as basic units for acoustic modeling, in an attempt to model co-articulation phenomena. Each phone in the language of interest may be represented by a number of acoustic models, each of which may be used in a specific set of contexts (preceding and following phones). Some questions arise when using CD phone models:

1. How large should the context be? A larger context leads to more detailed modeling and a greater potential for recognition accuracy, but requires more computing resources.
2. Which variants should be modeled? A sufficiently large number of examples must be available for each variant for the statistics to be significant.

In the CRIM LVCSR setup, as with many existing setups, the answer to question 1 is currently "one phone on the left ($k_l = 1$) and one phone on the right ($k_r = 1$)"; the corresponding acoustic models are called Context-Dependent Triphones.

2.2 Compatibility of models in sequences

The main computational difficulty in handling CD models for ASR is their proper sequencing when evaluating the likelihood of each phone sequence (which depends on the word sequence under consideration). When context-independent phones models are used, any phone model may follow any

other phone model (within the constraints on word sequences). When context-dependent phones are used, only models with compatible contexts may follow one another. For example, if a model for phone a may be followed by a model for phone b (because its set of possible contexts allows it), then the model for phone b must be one which allows a model for phone a to precede it. More formally, if we define a logical model m for phone p by the triple (l, p, r), where l is the phone allowed on the left of m, and r is the phone allowed on the right of m, a model $m' = (l', p', r')$ for phone p' may follow m if and only if $p = l'$ and $p' = r$. A more visual guide is the following:

Model	$i-1$	i	$i+1$	$i+2$
(l, p, r)	l	p	r	$\ldots$
(l', p', r')	$\ldots$	l'	p'	r'
Constraint	none	$=$	$=$	none

To determine whether model $m = (l, p, r)$, representing the phone p at position i within the phone sequence under consideration may be followed by model $m' = (l', p', r')$ representing phone p' at position $i + 1$, shift m' by one position to the right and superimpose it with m. If all positions marked "=" have the same phones, then (and only then) m may be followed by m'. The case for quinphones is similar:

Model	$i-2$	$i-1$	i	$i+1$	$i+2$	$i+3$
(l_2, l_1, p, r_1, r_2)	l_2	l_1	p	r_1	r_2	$\ldots$
$(l'_2, l'_1, p', r'_1, r'_2)$	$\ldots$	l'_2	l'_1	p'	r'_1	r'_2
Constraint	none	$=$	$=$	$=$	$=$	none

In general, any two models $m = (x_1, x_2, \ldots, x_k)$ and $m' = (y_1, y_2, \ldots, y_k)$ may be considered in the sequence (m, m') if and only if $x_{i+1} = y_i, 1 \leq i < k$. Determining whether sequences of three or more models are compatible requires determining whether all pairs of consecutive models are compatible.

2.3 Practical limitations

In practice, it is not always possible to have distinct models for all the possible contexts in which a phone may be found:

1. Recordings (audio examples) may not be available for all contexts. In practice, this will be the case.

2. The number of logical models may be too large to handle explicitly if large contexts are used. (See table 1.)

The usual solution to this is to group, or *cluster*, certain phonetic contexts and have each group share a common physical model. (A *logical* model corresponds to a specific phonetic context; a *physical* model may correspond to one or more logical models and thus may represent multiple phonetic contexts.) The method we use at CRIM is based on acoustic-phonetic *decision trees* (DT), which we describe in more detail below.

	Context size:	$k_l = k_r = 1$	$k_l = k_r = 2$
Description	N	$N(N+1)^2$	$N(N^2+N+1)^2$
Small	30	28 830	26 002 830
French	37	53 428	73 247 013
English	43	83 248	154 088 307
Large	50	130 050	325 380 050
Huge	60	223 260	804 175 260

Table 1: Number of contexts as a function of the number of phones and the context size. N **is the number of phones; the column** $k_l = k_r = 1$ **corresponds to triphones, and the column** $k_l = k_r = 2$ **corresponds to quinphones.**

3 Acoustic-phonetic decision trees

The widely used method of phonetic decision trees [2] is a hybrid:

1. It uses a rudimentary linguistic "expert system", in the form of questions regarding contexts. For example: "*Is the preceding phone a vowel?*".

2. It uses a data-driven method to determine the relevance of each question for specific contexts.

It is a top-down method which starts from general models and specializes them progressively.

3.1 Outline of the algorithm

Consider a specific phone p and a set of questions Q; build a single tree node, the root, whose model will be a Context-Independent phone; then perform the following steps as long as there are un-processed nodes:

1. Determine whether the node has enough examples in the training data set; if not, this node will be a leaf.

2. For each $q \in Q$, split $node$ into $node_n$ and $node_y$, and associate with $node_n$ all logical CD triphones for which the answer to the question is "no" and associate with $node_y$ all logical triphones for which the answer to the question is "yes". Compute the increase in the likelihood of the training data which comes from this more detailed modeling.

3. Select the question whose corresponding node split yields the largest increase in likelihood.

4. If the increase in likelihood exceeds a certain pre-determined threshold, add $node_n$ and $node_y$ as new roots to be processed, starting from step 1.

At the end of this process, the leaves will represent sets of logical models for which the allowed contexts depend on all the questions that were asked in the construction of the decision tree. The models used in practice, Hidden Markov Models (HMM), use more than one state for better modeling of the transition between the pronunciation of consecutive phones. In this case, a separate

decision tree is built for each state position. Note that in step 2, a naive implementation keeps a list of all possible contexts, whose number is given in table 1. In order to handle quinphones or models with larger contexts, a less naive method is needed.

4 Finite-State Machines

4.1 Definition

A Finite-State Machine, or FSM, is a graph-like structure which has a set of states and a set of arcs connecting those states. Arcs bear labels which contain an input symbol and possibly additional information depending on the FSM type. In particular, arcs may bear output symbols and/or weights. This table shows the correspondence:

Acronym	Name	Output symbols	Weights
FSA	Finite-State Automaton	N	N
WFSA	Weighted Finite-State Automaton	N	Y
FST	Finite-State Transducer	Y	N
WFST	Weighted Finite-State Transducer	Y	Y

A (W)FSA is equivalent in practice to a (W)FST where all arcs have identical input and output symbols. A path is a sequence of n states and $n-1$ arcs; a *valid path* is a path where the first state is an initial state of the FSM and the last one is a final state (or accepting state). The input label of a path is the concatenation of the individual input arc symbols along the path; the output label of a path is the concatenation of the individual output arc symbols along the path; the cost of a path is the sum of the weights of the individual arcs on the path.

4.2 FSM representations for ASR

4.2.1 The language model

The WFSA G representing the language model contains one valid path for every possible word sequence, where every word belongs to a pre-determined vocabulary. The cost of a path π is equal to $-\ln(p_{LM}(w(\pi)))$, where $w(\pi)$ is the word sequence (label) on π and $p_{LM}(w(\pi))$ is the probability of this word sequence according to the language model.

4.2.2 The pronunciation dictionary

The FST D representing the pronunciation dictionary (or lexicon) contains one valid path for each word/pronunciation pair in the dictionary. Also, it allows for any number of words. The input symbols represent phones, and output symbols represent words.

4.2.3 The context-dependency constraints

The FST C maps every sequence of compatible context-dependent phone models to the corresponding sequence of context-independent phones. This is produced in two steps. In the first step, which depends only on the set of phones used, an FST C_L representing the allowable sequences

of logical context-dependent models is built. In the second step, which depends on the specific acoustic models (AM) used, the logical models are replaced by the corresponding physical models as determined by the AM, and the result is optimized for size by taking advantage of the redundancy produced. The number of states for C_L in the case of triphones is given by the formula $(N+1)^2 = O(N^2)$, and the number of arcs is given by the formula $N(N^2+2N+2) = O(N^3)$, where N is the number of phones. For contexts of size k, the number of states would be $O(N^{k-1})$ and the number of arcs would be $O(N^k)$ (the number of contexts is also $O(N^k)$).

4.2.4 HMM state sequences

The FST H representing valid sequences of HMM states, that is, sequences of HMM states which form physical models, depends heavily on the specific acoustic models used. The largest possible size for H occurs when all logical models are distinct, i.e., when each physical model represents exactly one logical model.

5 Production of the recognition network

The following subsections describe the overall process by which the integrated recognition network is produced. Additional operations are applied to the result of intermediate steps to reduce their redundancy and/or size, and to make the end result more efficient for speech recognition. Moreover, some markers (auxiliary symbols) have to be added in order for these optimizations to be applicable. We do not give details about those operations here, as this is just an overview.

5.1 Process

The construction proceeds right-to-left, with D and G being combined to produce DG, which is then combined with C to produce CDG, and finally this is combined with H to produce $HCDG$.

5.2 Size considerations

Of all four FSTs H, C, D, and G, the size of G and the size of H are the ones that have a significant effect on the size of $HCDG$. In practice, most of the times when DG can be built without exhausting the available memory, so can CDG. However, even though $HCDG$ can often be built, the optimizations cannot be applied, which limit the use of the result (for example, real-time recognition may not be possible). Note that the size of the vocabulary (and thus the size of D) has a minimal impact on the size of the final result.

5.3 An earlier alternative method

In the early days of FSTs at CRIM, the construction process for $HCDG$ was somewhat different: DG was produced as above, but H and C were combined together into HC, which was then optimized, and then this would be combined to DG without further optimizations. This method proved necessary given the limited amount of RAM at that time. However, when we later could compare both methods due to our having more RAM available, it turned out that this alternative method always produced significantly larger final results (even though the intermediate results were more manageable).

5.4 A revival?

In [1], a method is described which is similar to the alternative method described above, except that it produces the equivalent of HC without constructing H and C explicitly. The advantage of this method is that the computer resources needed depend on the actual (physical) size of the acoustic models, and not on their theoretical size (see table 1). The remainder of this paper describes CRIM's implementation of this method, as well as some experiments.

6 Generating the HC FST

6.1 Introduction

The HC FST maps any allowable sequence of HMM states to its corresponding phone sequence. Equivalently, the inverse of HC maps any sequence of phones to a correct sequence of HMM states, where "correct" means that the context of each phone in the phone sequence is taken into account. Here is a recap of the limitations of the traditional method outlined above:

1. It requires an explicit list of models, which has size $O(N^k)$ for a context of size k.
2. It generally works fine for triphones, but is not practical for larger contexts.
3. It does not take advantage of the sharing inherent in model sets produced using a decision tree.

Constructing HC requires that we determine which sequences of HMM states are allowable. In CRIM's "traditional" setting, this information is obtained from the list which maps logical models to physical models, and from the correspondence between physical models and the sequence of HMM states of which they are comprised (these are by-products of the acoustic model training process). The overall process requires these two steps:

1. Generating within-phone sequences (physical models): enumerate sequences of states from a given phone which have compatible contexts.
2. Generating between-phone sequences: enumerate all valid ordered pairs of physical models which can be considered.

6.2 Generating bitmaps from a decision tree

In order to produce within-phones sequences from a decision tree, we

1. Generate a bitmap describing in a compact manner the contexts for each decision tree leaf node.
2. Use the leaf node bitmaps to generate the HC FST.

Given a set of phones P, $N = |P|$, and a context width $k = k_l + 1 + k_r$ (k_l is the size of the left context, and k_r is the size of the right context), bitmaps will be bit matrices of dimensions $(N + 1) \times k$, with row 0 corresponding to a "null phone" which is useful for treating the case of

missing contexts (models in leftmost or rightmost position; see 6.2.3 below). Column k_l always contains exactly one 1, the row of which determines the phone to which the state under consideration belongs (the *reference phone*). Column(s) 0 to $k_l - 1$ (inclusive) describe the permissible left contexts whereas columns $k_l + 1$ to $k - 1$ (inclusive) describe the permissible right contexts. Example for triphones ($k_l = k_r = 1$, so $k = 3$):

Bitmap column	0	1	2
Relative position	-1	0	1
Context	left	reference phone	right

Example for quinphones ($k_l = k_r = 2$, so $k = 5$):

Bitmap column	0	1	2	3	4
Relative position	-2	-1	0	1	2
Context	left of left	left	reference phone	right	right of right

6.2.1 Basic definitions

6.2.1.1 Bitmap operations The basic bitmap operations are union, represented as $\vee$, and intersection, represented as $\wedge$.

6.2.1.2 Canonical empty bitmaps A canonical *empty* bitmap for phone p_i, which does not allow any phone in any context, consists of all zeroes except that $b_{ik_l} = 1$. While it is useless by itself (since it represents an empty set of allowable contexts), it is used as the neutral element (as an initialization value) for union operations. Example where $i = 1$ and thus $p_i = a$:

phone	number	0	1	2
ϵ	0	0	0	0
a	1	0	1	0
b	2	0	0	0
c	3	0	0	0

6.2.1.3 Empty bitmaps An *empty* bitmap is a bitmap which represents an empty set of allowable contexts, but which is not necessarily canonical. It is characterized by the presence of at least one all-zero column, indicating that no phone may be found at that specific position. Thus, determining whether a bitmap is empty involves determining whether any column contains all zeros. We will loosely notate an empty bitmap by the symbol $\emptyset$.

6.2.1.4 Full bitmaps A *full* bitmap describes a subset of logical models where all phones are allowed at all positions other than the reference phone. It is used as the neutral element of intersection operations. The elements $b_{ij}, 0 \leq i, i' \leq N, 0 \leq j < k$ of a full bitmap for a phone $p_i \in P$ satisfy

$$\begin{aligned} 0 \leq j < k &\Rightarrow b_{ij} = 1, \\ i' \neq i &\Rightarrow b_{i'k_l} = 0, \\ i' \neq i, j \neq k_l &\Rightarrow b_{i'j} = 1, \end{aligned}$$

Example of a full bitmap where $i = 1$, and thus $p_i = a$:

phone	number	0	1	2
ϵ	0	1	0	1
a	1	1	1	1
b	2	1	0	1
c	3	1	0	1

6.2.1.5 Out of bounds definitions

For convenience, we define $b_{ij} = 0$ if $i < 0$ or $i > N$ and $b_{ij} = 1$ for $0 \leq i \leq N, j < 0 \vee j > k$. This can be interpreted as saying that phones not in P are not allowed in any context; also, phones which are farther from the reference phone than what the bitmap describes (i.e., outside the context) are implicitly allowed, and this makes it explicit. In particular, one can produce, for example, quinphone bitmaps from triphone bitmaps by prepending and appending a 1 to each line of the bitmap.

6.2.2 Question bitmaps

Question bitmaps are used to refine the sets of permissible contexts as we descend into the decision tree. We split questions into two types: elementary questions and compound questions. An elementary question is of the form "Is the phone at position $k_j, 0 \leq k_j \leq k-1$ phone p_i?". A compound question is of the form "Is the phone at position k phone p_{i_1}, or p_{i_2}, or ...?"
Notes:

1. Since questions may apply to any phone, we must allow any reference phone at position k_l, that is, we set $b_{ik_l} = 1, 0 \leq i \leq N$ for all question bitmaps.

2. Also, we need separate bitmaps for the "no" answer and for the "yes" answer.

6.2.2.1 Generating the bitmaps for an elementary question

1. Answering "yes" to an elementary question q means that a specific phone is at the position asked, and thus that all other phones may *not* be found at that position. Also, the answer has no bearing on other positions. Therefore, the corresponding bitmap has all 1s in non-relevant positions, and exactly one 1 in the position asked, that is, the "yes" bitmap for an elementary question q has $b_{ik'}(q, y) = 1, k' \neq k$, $b_{ik}(q, y) = 1$, and $b_{i'k}(q, y) = 0, i' \neq i$. Example of a "yes" question bitmap for question "Is the right phone c?":

phone	number	0	1	2
ϵ	0	1	1	0
a	1	1	1	0
b	2	1	1	0
c	3	1	1	1

2. Answering "no" to an elementary question q only means that a specific phone may *not* be found at the position asked, and thus that all other phones may still be found at that position. As in the case for the "yes" question, the answer has no bearing on other positions. Therefore, the corresponding bitmap has 1s everywhere except at the entry regarding the question asked, that is, the "no" bitmap for an elementary question satisfies $b_{ik}(q, n) = 0$ and $b_{i'k'}(q, n) = 1, i' \neq i \vee k' \neq k$. Example of a "no" question bitmap for question "Is the right phone c?" (same question as above):

phone	number	0	1	2
ϵ	0	1	1	1
a	1	1	1	1
b	2	1	1	1
c	3	1	1	0

Since the question in the example could be applied to any phone's decision tree, column 1 of both "yes" and "no" bitmaps contains all 1s; since the question concerns the right context, its corresponding answer does not restrict which phones may be on the left context, and thus column 0 of both "yes" and "no" bitmaps contains all 1s.

6.2.2.2 Generating the bitmaps for a compound question

Let $Q = q_1 \vee q_2 \vee \ldots \vee q_m$ be a compound question consisting of the union of m elementary questions ($m \geq 1$):

1. The "yes" bitmap for a compound question is the logical *or* of the "yes" bitmaps of its constituent questions, i.e., $B(Q, y) = B(q_1, y) \vee B(q_2, y) \vee \ldots \vee B(q_m, y)$.

2. The "no" bitmap for a compound question is the logical *and* of the "no" bitmaps of its constituent questions, i.e., $B(Q, n) = B(q_1, n) \wedge B(q_2, n) \wedge \ldots \wedge B(q_m, n)$. To understand why, consider the example question "Is the right context a or is the right context b?": if we answer "no", then we are saying "The right context is not a *and* the right context is not b."

6.2.3 Handling leftmost and rightmost contexts

Since utterances have a beginning and an end, models for phones uttered at these boundaries have either no left or no right context. There are two main ways to handle this case:

1. Require that there be some pre-determined context-independent model (e.g., a silence model) at both ends of any utterance.

2. View an utterance as having a number of *null phones* before its beginning and another number of null phones after its end. The problem is now reduced to considering an empty context as the "presence" of a null phone.

The former works, but is not general. The latter handles the problem correctly in all situations, which is why we introduced the null phone ϵ above. Indeed, if a question asks "Is the right context phone a" and the answer is "yes", then the right context may not be empty ($\neq \epsilon$); moreover, if the answer is "no", then the right context may be empty.

6.2.4 The bitmaps of a decision tree root node

The "yes" and "no" bitmaps of a DT root are both full bitmaps as described in 6.2.1.4.

6.2.5 Splitting a node depending on a question

The procedure for computing node bitmaps proceeds top-down from the root to the leaves. Let T be a non-leaf node split according to question Q; let T_y be the yes child and T_n be the no child. We have:

$$\begin{aligned} B(T_y) &= B(T) \wedge B(Q, y), \\ B(T_n) &= B(T) \wedge B(Q, n). \end{aligned}$$

6.2.6 Pitfalls

The construction process for decision trees may include the merging of leaf nodes. That is, a decision tree may in fact be a Directed Acyclic Graph (DAG). Care must be taken when constructing bitmaps from such a DAG. It would seem logical that the bitmap for two merged DT leaves should be the union of their respective bitmaps; however, this is incorrect as it leads to an excessive list of physical models (allowable sequences of states). The correct way to handle merged leaves is to keep them distinct until the HC FST arcs are output.

6.3 Generation of HC from bitmaps

The process for generating the HC FST from the bitmaps produced using the method described above is outlined in [1]. Here, we attempt to give enough details for the reader to understand our implementation.

6.3.1 Generation of physical models

This is performed for each phone, one phone at a time:

1. Group all states (DT leaf nodes) per HMM state position into sets $pos_1, pos_2, \ldots, pos_H$.
2. Set $T_1 = pos_1$.
3. For $\{i : 2 \leq i \leq H\}$ in sequence (if $H = 1$, we are done):
   (a) Let T_i be the initially empty list of allowable sequences of states of length i.
   (b) Consider all ordered state pairs $\{(s_{j_1}, s_{j_2}) : s_{j_1} \in T_{i-1} \wedge s_{j_2} \in pos_i\}$; if a pair is compatible, that is, if $B(s_{j_1}) \wedge B(s_{j_2}) \neq \emptyset$, add it to list T_i.
4. The set of physical models is $M = T_H$; the bitmap of each physical model is the $\wedge$ of its constituent states.

6.3.2 Determining allowable sequences of physical models

Now that we know which sequences of HMM states form valid physical models, we need to determine which ordered pairs (m_{i_1}, m_{i_2}) of physical models may follow each other. To this end, we express the concepts stated in 2.2 by using our bitmap framework. We define the *bitmap shift* operation as follows: if B is a bitmap with elements $b_{i,j}$, the shifted bitmap $B' = shift(B)$ has elements $b'_{i,j} = b_{i,j+1}$ for $j < k$, and $b'_{i,k} = 1$ otherwise. Example:

$$shift\left(\begin{array}{|c|c|c|}\hline b_{00} & b_{01} & b_{02} \\ \hline b_{10} & b_{11} & b_{12} \\ \hline b_{20} & b_{21} & b_{22} \\ \hline b_{30} & b_{31} & b_{32} \\ \hline\end{array}\right) = \begin{array}{|c|c|c|}\hline b_{01} & b_{02} & 1 \\ \hline b_{11} & b_{12} & 1 \\ \hline b_{21} & b_{22} & 1 \\ \hline b_{31} & b_{32} & 1 \\ \hline\end{array}$$

Then, model m_1 may be followed by m_2 if and only if $shift(B(m_1)) \wedge B(m_2) \neq \emptyset$. Algorithm 1 describes a naive way of producing the set R of allowable sequences of two physical models. This will examine $|M|^2$ pairs of models, most of which will not be compatible. A more efficient algorithm is given in Algorithm 2; lines 3 and 4 organize the models so that only pairs of models which have a chance of matching are considered. In practice, this may lead to an order of magnitude gain in speed over the naive method.

Algorithm 1 Naive generation of compatible physical model pairs

1: $R = \emptyset$
2: **for all** $\{(m_1, m_2) : m_1 \in M \wedge m_2 \in M\}$ **do**
3: **if** $shift(B(m_1)) \wedge B(m_2) \neq \emptyset$ **then**
4: $R \leftarrow R \cup \{(m_1, m_2)\}$
5: **end if**
6: **end for**

Algorithm 2 Efficient generation of compatible physical model pairs

1: $R = \emptyset$
2: **for all** $\{i : p_i \in P\}$ **do**
3: $M_i = \{m : m \in M \wedge b_{i,k_l}(m) = 1\}$ ▷ All models for phone i ◁
4: $L_i = \{m : m \in M \wedge b_{i,k_l-1}(m) = 1\}$ ▷ All models which allow phone i on their left ◁
5: **for all** $(m_1, m_2) \in M_i \times L_i$ **do**
6: **if** $shift(B(m_1)) \wedge B(m_2) \neq \emptyset$ **then**
7: $R \leftarrow R \cup \{(m_1, m_2)\}$
8: **end if**
9: **end for**
10: **end for**

6.4 Generating the HC FST

At this point we have:

1. M, the set of physical models.
2. R, the set of compatible ordered pairs of physical models.

Algorithm 3 describes the process for generating the HC FST given what we have computed so far.

7 Evaluation of the method

In [1], the authors state that the end result of WFST construction ($HCDG$) obtained by using the bitmap method is 75%-80% smaller than obtained with their previous method. However, we could not duplicate that result, which may indicate that their original method was highly inefficient. In our setup, the final sizes of the two $HCDG$ FSTs is similar. An important step, not mentioned in [1], is that it is the inverse of HC that should be determinized, not HC itself. That is, where the article mentions $det(\widetilde{HC})$, it should mention $\widetilde{T}$, where $T = (det((HC)^{-1}))^{-1}$. Omitting this step causes a size explosion in the composition of HC and DG, even for small DG.

8 Conclusion and Future Work

In this report, we have described the implementation of a method for producing weighted finite-state transducers representing the search network for automatic speech recognition. The method is suitable for modeling large phonetic contexts, which we intend to use eventually. We have not succeeded in reducing the memory requirements for composition, but we expect to do so in the near future. This will allow us to push the limits of our models and to use ever larger acoustic and language models.

References

[1] M. Schuster and T. Hori, "Efficient generation of high-order context-dependent weighted finite-state transducers for speech recognition," in *Proceedings of ICASSP 2005*, 2005, vol. I, pp. 201–204.

[2] S.J. Young, J.J Odell, and P.C. Woodland, "Tree-based state tying for high accuracy acoustic modelling," in *Proceedings of HLT 1994*, 1994.

Algorithm 3 Algorithm for generating the HC FST

Let q_0 be the initial state of HC
Let q_F be the (only) final state of HC
▷ The set of states of HC ◁
$Q \leftarrow \{q_0, q_F\}$
▷ The set of final states of HC ◁
$F \leftarrow \{q_F\}$
▷ The set of arcs of HC ◁
$A \leftarrow \emptyset$
▷ Generate within-phone-arcs ◁
for all $m \in M$ **do**
    Let n be the number of states of m
    for $i = 1$ to n **do**
        Let s_i be the name of state i of model m
        $Q \leftarrow Q \cup \{q_{i-1}(m), q_i(m)\}$
        $A \leftarrow A \cup \{(q_{i-1}(m), q_i(m), s_i, \epsilon)\}$
    end for
end for
▷ Generate between-phone arcs ◁
for all $(m_i, m_j) \in R$ **do**
    Let n be the number of states of m_i
    Let p be the reference phone of m_j
    $A \leftarrow A \cup \{(q_n(m_i), q_0(m_j), \epsilon, p)\}$
end for
▷ Generate arcs from initial state ◁
for all $m \in M$ **do**
    Let p be the reference phone of m
    if $b_{0,j} = 1, 0 \leq j < k_l$ **then**
        $A \leftarrow A \cup \{q_0, q_0(m), \epsilon, p)\}$
    end if
end for
▷ Generate arcs to final state ◁
for all $m \in M$ **do**
    if $b_{0,j} = 1, k_l < j \leq k$ **then**
        $A \leftarrow A \cup \{q_n(m), q_F, \epsilon, \epsilon)\}$
    end if
end for