6th International Conference on Spoken Language Processing (ICSLP 2000)
Beijing, China
October 16-20, 2000

REDUCED TRACEBACK MATRIX STORAGE FOR SMALL FOOTPRINT MODEL ALIGNMENT

Jeff Meunier

Human Interface Laboratory, Motorola Labs
1301 E. Algonquin Rd., Schaumburg, IL 60196, USA
meunier@labs.mot.com

ABSTRACT

This paper describes two alternative techniques for storage of path traceback information during model alignment. By taking advantage of the restricted state transitions in the standard left-right hidden Markov model (HMM), traceback information can be stored as a set of state dwell counts. Compared to the traditional method of storing intermediate path information for every frame, these in-place techniques can provide significant memory savings in a small footprint embedded system. The first technique organizes the dwell counts in a "triangular array" and specifies a set of update operations performed during alignment. This direct storage technique allows for easy access of the path information once alignment is complete. The second technique encodes the dwell counts in base-*D* numeric values. By doing this, the array update procedure reduces to a single calculation and does not require multiple memory copies. Both techniques can be done in-place and require around $N^2/2$ memory words instead of the NT memory words required for conventional storage (where N is the number of HMM states and T is the maximum number of frames in a modeled utterance).

1. INTRODUCTION

When performing Viterbi alignment for speech recognition, some procedures require full frame-to-state alignment information and not just scores. For instance, during speaker dependent enrollment the system uses the user's input (typically one or more spoken samples of a new utterance) to derive a representative hidden Markov model (HMM). HMM parameters are calculated by assigning portions of the speech utterance to individual HMM states via Viterbi alignment and using this assignment to update the HMM state parameters. To do this a full frame-to-state alignment is needed, consisting of the best path through the model at every frame.

Since the best path can not be identified until alignment is complete, intermediate path information is usually saved by recording the best path into every state at every frame. Since this array of path information, or traceback matrix, has a storage word for every frame-state combination, this technique can require large amounts of memory. Specifically, for a model with N states and a maximum utterance length of T frames, the traceback matrix will occupy NT memory words. In a case where a 20-state whole word model is used for a 3 second long utterance, 6000 words of memory would be necessary to store the traditional traceback matrix.

In order to implement such alignment algorithms on a device where very little RAM is available, a technique is needed for storing traceback information that minimizes memory usage. To accomplish this, the work presented here uses information about the HMM structure to reduce the amount of traceback information needed. Due to the restrictions often placed on HMM state transitions, a more efficient method of storing traceback path information is possible.

2. REDUCED TRACEBACK MATRIX STORAGE ALGORITHMS

HMM state transitions are typically restricted to being left-right in nature, where the allowable paths into a given state come only from that state or preceding states. Such an HMM, sometimes referred to as a Bakis model [1][2], is shown in Figure 1.

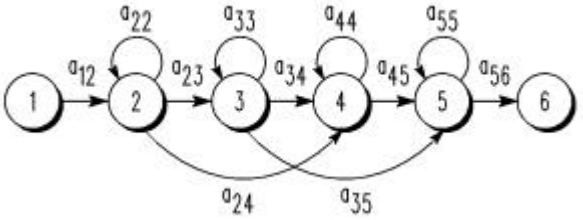

Figure 1: Left-right, or Bakis, hidden Markov model

This simplification from the case of the ergodic HMM, where any state can precede any given state, dramatically reduces the number of possible paths through the model. Figure 2 illustrates this. Paths into state 5 at frame 5 (point A) are limited to coming from state 5 (B), state 4 (C), or state 3 (D) for this left-right model that allows single skips.

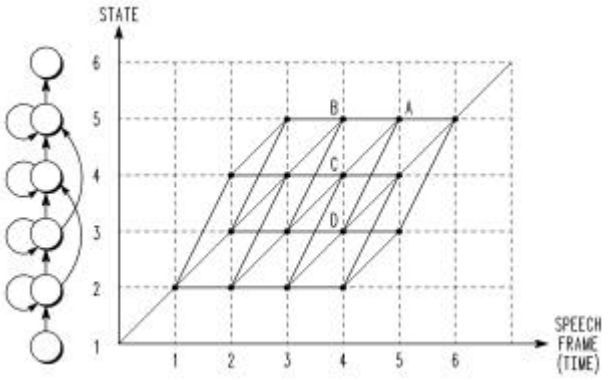

Figure 2: Traceback lattice for a left-right HMM

With this simplified lattice structure, Lou [3] observes that a path through the model to a given state can be uniquely described by the number of frames spent in each state along the path. For example, it would be sufficient to record a path to state five at frame 5 as follows: one frame in state one, one frame in state two, zero frames in state three (if it was skipped), 3 frames in state four, and one frame in state 5. By storing the best path into each state in this fashion, and by defining a set of update procedures for maintaining this information during model alignment, the storage needed for traceback information is greatly reduced. Two techniques based on this storage method are described below. The first technique uses simple storage of the state dwell counts in an optimized array to store the path information. The second technique uses base-*D* numeric values to record the path information, which simplifies the update procedure.

2.1. Storage Using Dwell Counts

The first technique, as Lou suggests in [1], uses an array of state "dwell counts" to efficiently maintain traceback information. The path back from each state *n* is stored by updating a set of *n* dwell counts, enumerating how many frames were spent in each of the states *1* through *n*. A skipped state can be represented by zero frames assigned to that state. To store and update this information efficiently the notion of a "triangular array" is used, as illustrated in Figure 3.

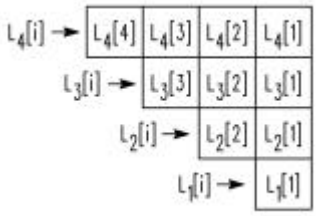

Figure 3: Triangular array for storage of traceback information as dwell counts.

Only *n* dwell counts are necessary to define the path to state *n*. For example, the path to state three is recorded in the array row $L_3[]$ by storing the dwell count for state 3 in $L_3[3]$, the count for state 2 in $L_3[2]$, and the count for state 1 in $L_3[1]$. For an HMM with *N* states, this requires an array of size *N(N+1)/2*. Since *N* can be up to an order of magnitude smaller than the number of frames *T* in an utterance to be modeled, this represents a significant savings over storing a preceding state number for each frame and state in a size *NT* conventional array.

Additional benefit is realized using this storage format by updating the path information in-place. As alignment progresses with new incoming frames, only the most recent traceback hypotheses are of interest. Traceback path hypotheses at previous frames are not important, and therefore do not need to be stored. In-place update of the array is possible, as long as information needed from the previous iteration is retained in an appropriate manner. By beginning the search with the last state first, path information for the preceding states is available from the rows below that have not yet been updated. When the path for the state being evaluated needs to inherit the best path back from a previous state, that information can be copied from the corresponding row below.

To perform these in-place operations and update the traceback array during model alignment, a set of update procedures is needed. Only two core operations are necessary to accomplish this: one simple operation to handle self-loops, and a second generalized operation to handle state transitions. These update procedures are illustrated in Figure 4. The examples show possible updates of the traceback array considering paths into state 4 at frame 5. The path information back from state 4 is stored in the top row of the array. For each transition type, the array on the left shows the contents before the update at frame 4, and the array on the right shows the contents after the update at frame 5.

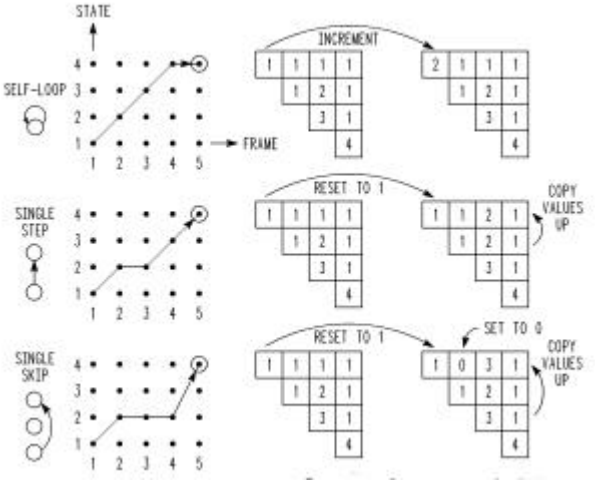

Figure 4: Update procedures for traceback information stored as dwell counts.

When the self-loop transition is chosen as optimal, the update procedure consists simply of incrementing the counter for the current state. Since the rest of the path back remains unchanged, no memory moves are necessary. When the single step transition is chosen, two operations need to be performed. First, the dwell counter for the current state is reset to 1, to indicate that only the current frame is aligned with that state. Then, the rest of the path back is "inherited" from the preceding state by copying the values from the corresponding row. In the example, since the best bath is through state 3, the path information from state 3 is copied up into the row for state 4. When the single skip transition is chosen, the procedure is nearly identical. In addition to resetting the current state dwell count to 1, the skipped state's dwell count is set to zero to indicate that no frames were aligned with that state. When inheriting the rest of the path from the preceding state, it is copied from two states back to reflect the skip. It can be seen that this generic procedure conveniently extends to any number of skipped states, although skipping more than one state is not typically allowed. After performing the appropriate update for the path back from state 4, the procedure would continue with the updates of the rows for states 3, 2, and 1, at each point using the path

information in the rows below when necessary to inherit paths back from preceding states.

Once alignment of an utterance is complete and the traceback array is fully updated with the optimal path information as of the last frame, recovery of frame-to-state alignment information is straightforward. Each memory word $L_n[i]$ contains the dwell time spent in a state, and the sum of these dwell times across a row is equal to the number of frames in the utterance. Generating a frame-to-state alignment is a simple matter of reading out the dwell counts and tagging the correct number of frames with each state number. Sometimes state dwell count information is explicitly used during model training to update parameters for state duration penalties or constraints. Since this is the storage format used by the traceback array, the information is readily available and does not need to be recorded or generated separately.

In these examples, discussion has involved memory access and memory copies from this "triangular array" of traceback information. Needless to say, in an actual hardware implementation memory is not organized like this. The triangular structure would be unrolled and mapped onto a linear array of memory words. Due the uniform structure of the array, it is relatively simple to access the linearized array in this triangular fashion with intelligent use of pointer arithmetic.

Nonetheless, it might be beneficial in some systems to eliminate the memory move procedure and simplify the updates to computational operations. By collapsing each row of individual dwell counts into a single value, the technique of the next section accomplishes this.

2.2. Storage Using Encoded Values

Looking at the memory array of Figure 3, multiple memory words are used in a row to store a single traceback path. In order to reduce the pointer arithmetic and memory copies needed to update the array, a second technique is introduced. This second approach uses a single encoded value to store a traceback path. Illustrated in Figure 5, each row of memory words is reduced to a single memory word.

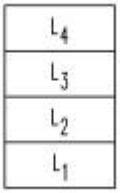

Figure 5: Array for storage of traceback information as encoded values.

Each memory word L_n stores the path information back from state *n* in a base-*D* number at least *n* base-*D* "digits" long. This is best illustrated in the examples of Figure 6. For simplicity of illustration the base *D* chosen is 10, however in a real implementation another value would likely be chosen.

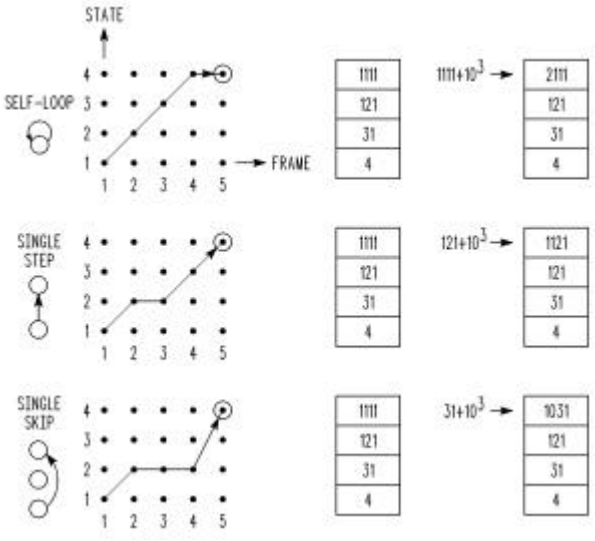

Figure 6: Update procedures for traceback information stored as encoded values.

The examples use the same paths that were used in Figure 4, so it should be apparent how the multiple memory words have been consolidated into one encoded value. For example, if "121" is given as the value representing the path back from state 3, the path is: one frame in state 3, 2 frames in state 2, and 1 frame in state 1. Since each digit in the encoded value represents the number of frames spent in a state, the base *D* that is chosen sets the upper bound on the number of frames that can be assigned to a single state. In the example of Figure 6, since *D* is equal to ten, no more than nine frames can be assigned to any given state. Therefore, there is a tradeoff between the size of the memory word needed to store the encoded value and the maximum allowable state dwell time that can be accommodated.

By representing the path information with a single encoded value, the update operations can be performed via a simple calculation instead of memory copies. Furthermore, the path updates for all possible cases are performed with the same generalized operation. When updating the path to state *n*, if the optimal path into state *n* is from state *m* (where $m \leq n$) the encoded path L_n is updated as

$$L_n = L_m + D^{n-1}$$

where *D* is the base chosen for the encoded value. As before, the update procedure is performed working down starting with the last state L_N, so path information for preceding states can be inherited.

Comparing this operation with the updates of the previous technique, it can be seen that the same path information is being stored and used, just in a different format. In the examples of Figure 6, the possible updates for the path into state 4 are being considered and the value of L_4 is being updated. In each case the value $D^{n-1} = 10^3$, representing a frame spent in frame 4, is being added to the value corresponding to the preceding state selected. For the self-loop the corresponding value is L_4 (1111), for the single step the value is L_3 (121), and for the single skip

the value is L_2 (31), as shown in the figure. As before, the technique used here can handle any number of skipped states. Once the update for state four is complete, the process would continue with states 3, 2, and 1.

Once alignment of an utterance is complete and the traceback array is fully updated with the optimal path information as of the last frame, a small amount of processing is necessary to retrieve frame-to-state alignment information. Since the dwell counts are stored as base-D digits, individual counts may not be directly available. In this case, it may be necessary to iterate through the value while recording the frame-to-state assignments, subtracting off powers of D^{n-1} and checking the result at each step. However, if D has been chosen to allow convenient access (e.g., a power of 2), the individual dwell counts may be accessible using a simple bit mask. As before, since we are storing dwell counts, they can also be used to update parameters for state duration constraints or penalties in the HMM.

Use of this encoded value technique would be most practical in systems where the maximum dwell D and the number of states N are small, making the number of bytes required to store each base-D value L_n manageable and convenient. As mentioned previously, the base D chosen will impose an upper limit on the number frames that can be assigned to any state. Furthermore, a base D can be chosen (e.g., again by choosing a power of 2) that will allow easy generation of the exponential D^{n-1}. Other choices might require calculating the exponential on the fly or storing pre-computed values in a lookup table. Unless an extremely small or extremely large value of D is chosen, the memory required for array storage in this format is similar to that required for the first technique. However, the nature of the uniform, calculation-based update might make this procedure an attractive alternative in some small-footprint alignment applications.

3. CONCLUSION

In embedded speech recognition systems where there is sensitivity to algorithm footprint, methods for reducing memory usage continue to be important. Often, the maintenance of traceback path information during model alignment can be memory intensive. By taking advantage of the constraints in the left-right HMM, a path through the model can be completely specified by a set of state dwell counts. This paper has described two methods of reducing traceback array size that use this principle. The first technique stores the state dwell counts as individual memory words, and proposes a triangular memory structure for efficient storage of the array. By defining two standard operations for updating the array during model alignment, the information can be easily updated in-place using a small number of memory move operations. When alignment is complete, the dwell counts can be directly accessed to determine frame-to-state alignment. The second technique stores the same information using a base-D numeric value. Not only does this allow in-place array updating during alignment, but it also reduces the update to a single calculation that does not require memory copies. The value of D that is chosen does sets an upper limit on the maximum state duration that can be supported, which must be considered during implementation. The value of D chosen can also simplify the update calculations and simplify access to individual dwell counts when alignment is complete. While the encoded value technique may be more appropriate in some cases than in others, the uniform and calculation-based update procedure lends itself to efficient implementation. With both techniques described, the memory required to store the traceback information is greatly reduced compared to traditional techniques.

4. REFERENCES

1. L. Rabiner and B. H. Juang, *Fundamentals of Speech Recognition*, Englewood Cliffs, NJ: Prentice Hall, 1993.
2. R. Bakis, "Continuous speech word recognition via centisecond acoustic states," in *Proc. ASA Meeting*, Washington D.C., April 1976.
3. H. Lou, "Implementing the Viterbi Algorithm: Fundamentals and real-time issues for processor designers," *IEEE Signal Processing Magazine*, vol.12 no. 5, Sept. 1995, pp. 42-52.