TIME-FIRST SEARCH FOR LARGE VOCABULARY SPEECH RECOGNITION

Tony Robinson[0 1] *and James Christie*[1]

SoftSound Limited[0] Cambridge University[1]
ajr@softsound.com {ajr,jdmc2}@eng.cam.ac.uk

ABSTRACT

This paper describes a new search technique for large vocabulary speech recognition based on a stack decoder. Considerable memory savings are achieved with the combination of a tree based lexicon and a new search technique. The search proceeds time-first, that is partial path hypotheses are extended into the future in the inner loop and a tree walk over the lexicon is performed as an outer loop. Partial word hypotheses are grouped based on language model state. The stack maintains information about groups of hypotheses and whole groups are extended by one word to form new stack entries.

An implementation is described of a one-pass decoder employing a 65,000 word lexicon and a disk-based trigram language model. Real time operation is achieved with a small search error, a search space of about 5 Mbyte and a total memory usage of about 35 Mbyte.

1. INTRODUCTION

Search is an interesting problem in the field of large vocabulary speech recognition. Typically the acoustic vectors corresponding to an utterance may be coded in a few thousand bytes and the lexical output in a few hundred bytes. Yet in between we often need tens or even hundreds of megabytes of memory to perform the search for the maximum likelihood word sequence. The total amount of memory used is often much larger than this as the acoustic and language models may be of considerable size.

The memory required for the acoustic and language models need not be large. For example, the techniques of vector quantisation and recurrent neural networks [7] provide compact acoustic models. Similarly a class based language model may be used or most of the language model may be based on disk and paged in when needed [5]. Thus this paper addresses the remaining source of memory usage, that is the search space.

The paper is organised as follows. Section 2 presents the time-first algorithm for the case of isolated word recognition. Section 3 then extends this to large vocabulary continuous speech recognition. A particular implementation is then described in section 4 and finally the paper concludes with a discussion of comparable search techniques and related areas.

2. ISOLATED WORD RECOGNITION

This section describes a reordered search strategy for large vocabulary isolated word recognition with hidden Markov models[2]. A word model is composed of a sequence of phone models according to a pronunciation dictionary. An example is given in table 1 which is taken from BEEP[3]. Figure 1 shows the hidden Markov model for a single word, `ABBOT`. It is composed of the phones `ae`, `b ax` and `t` and for simplicity single state phone models have been used[4]. As is conventional, $O(t)$ represents the observation at time t, $b_i(O(t))$ the likelihood of observing $O(t)$ given that it came from state i and a_{ij} the probability of transition from state i to state j. The dynamic programming search for the most likely state

A	ah
A	ey
A'S	ey z
ABANDON	ax b ae n d ax n
ABANDONED	ax b ae n d ax n d
ABANDONING	ax b ae n d ax n ih ng
ABANDONMENT	ax b ae n d ax n m ax n t
ABBEY	ae b iy
ABBOT	ae b ax t

Table 1: Word pronunciations

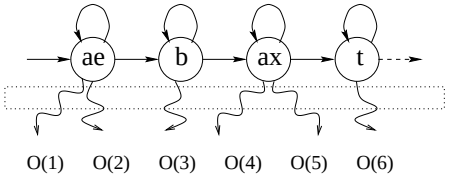

Figure 1: A simple hidden Markov model

sequence can be expressed as finding the maximum sum of log emission and log transition probabilities. The initialisation condition specifies that only the first state is valid at the start:

$$\phi_i(0) = \begin{cases} 0 & i = 0 \\ -\infty & i > 0 \end{cases} \quad (1)$$

and at every subsequent (j, t) the most probable path is chosen:

$$\phi_j(t) = \max_i \left(\phi_i(t-1) + \log a_{ij}\right) + \log b_j(O(t)) \quad (2)$$

Conventionally the search for the most probable state sequence proceeds time synchronously. This is illustrated in figure 2 where the values of $\phi_j(t)$ are computed in the numbered sequence[5]. The global best path is determined by tracing back the best transitions from the end to the start (shown in bold). This process is repeated for each word in the vocabulary and the most probable match is chosen as the recognised word. However, instead of performing the search as:

[0] SoftSound Ltd., PO Box 802, St Albans, AL3 4BF, United Kingdom.

[1] Cambridge University Engineering Department, Trumpington Street, Cambridge, CB2 1PZ, United Kingdom.

[2] It is also applicable to other dynamic programming based searches

[3] ftp://svr-ftp.eng.cam.ac.uk/pub/comp.speech/dictionaries/BEEP.tar.gz

[4] Three state, left to right, context-dependent hidden Markov models are currently the most popular

[5] In this example the points labelled 2, 3, 4, 7, 8 and 12 are not accessible and therefore may be explicitly excluded from the search. Similarly, points 13, 17, 18, 21, 22 and 23 can never be on the best path and may also be excluded.

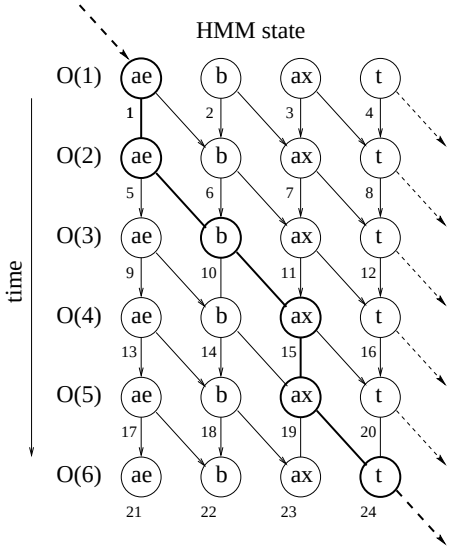

Figure 2: The standard dynamic programming search

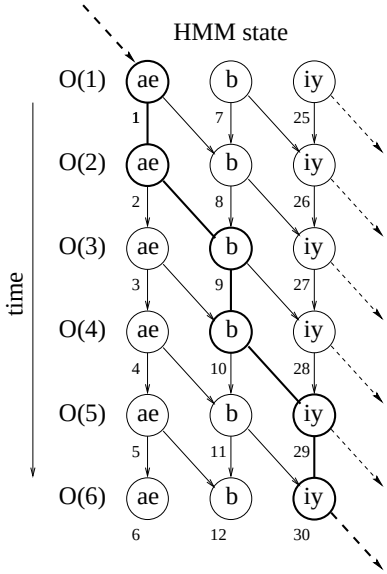

Figure 4: Sharing prefix computations in the reordered search

for t = 1 to T
    for j = 1 to N
        $\phi_j(t) = \max_i (\phi_i(t-1) + \log a_{ij}) + \log b_j(O(t))$

for "left-to-right" hidden Markov models it may equally well be carried out as:

for j = 1 to N
    for t = 1 to T
        $\phi_j(t) = \max_{i \leq j} (\phi_i(t-1) + \log a_{ij}) + \log b_j(O(t))$

as illustrated in figure 3. This is advantageous if the next item to

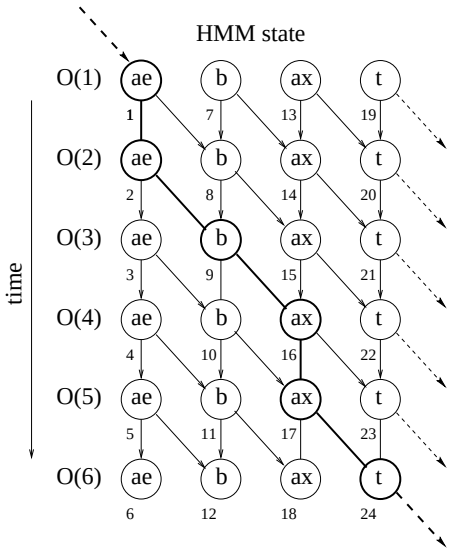

Figure 3: The reordered search

be searched shares a common prefix, such as the word ABBEY, shown in figure 4. Computation numbers 1–12 can be retained from the previous word, 13–24 are discarded and 25–30 added. In general we may search a complete pronunciation tree in this way. The tree for table 1 along with the computation order is shown in figure 5. Tree structures have several advantages. They represent the lexicon more compactly than the simple linear structure whilst maintaining a unique path for each word. When searching a tree structure the shared prefixes results in fewer computations. Also, when pruning is employed, large areas of the search space can be disregarded with one decision.

However, with the reordered search strategy there is an additional advantage, that is the memory required to store the values of $\phi_i(t)$ scales as T by the longest word in the lexicon as the memory can be reused as a stack. In the case of table 1 and figure 5 this requires a memory structure of T x 11 locations to search the word ABANDONMENT with all other searches reusing the lower memory locations.

The 65,000 word speech recognition system detailed in the next section contains an average of 1.08 pronunciations per word and has 136963 nodes in the pronunciation tree. The maximum tree depth is 20 nodes and the average word length is 26 observations. Assuming no pruning and that the details of the best path are not required, the time-synchronous algorithm uses 136963 storage locations and the new search algorithm requires 520 (20 x 26) storage locations[6].

3. CONTINUOUS SPEECH RECOGNITION

Continuous speech recognition in a stack decoder framework involves growing a tree of word hypotheses. In a simple implementation each leaf in the tree corresponds to one element on the stack. The stack may be ordered by time or by expected total path probability. Processing consists of popping the top item from the stack, extending it by each word in the lexicon and inserting the extended hypotheses into the stack. In addition, the finite state property of N-gram language models may be exploited by only maintaining the most probable stack item for each unique language model state. The list of word hypotheses to be processed may be kept on a global stack as in the implementation of Paul [4] or on one stack per time slot as in the NOWAY decoder of Renals [6].

When employing the time-first search strategy it is advantageous to group together, as one stack item, a sequence of word

[6]Both algorithms require the storage of the pronunciation tree, however this may be kept in read-only memory

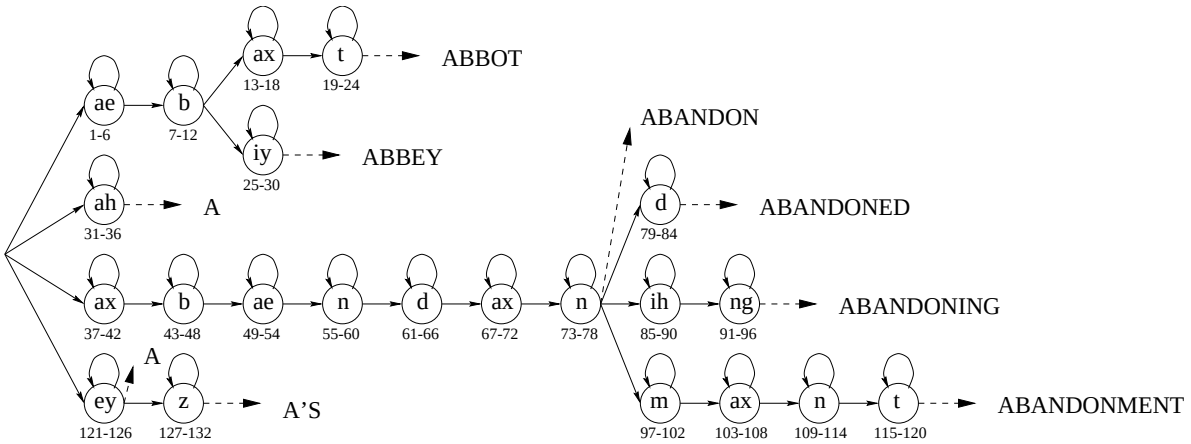

Figure 5: A tree structured lexicon showing the order of the computation steps

hypotheses where each hypothesis has associated with it the accumulated log probability and its location as a leaf in the tree of word hypotheses. Each stack item may be extended by all words in the lexicon by the method of section 2. For example, consider the joint recognition of the word ABBOT followed by the word ABBEY as in figure 6. The left hand side is the last column of figure 3 and shows the result of computation steps 22–24 which would appear as a stack item. The right hand side represents new computation, numbered as steps 133–144 and 157-162. Here it can be seen that there are three possible exit points from the word ABBOT which form entry points into the word ABBEY and that a data structure which groups together this information will result in more efficient search compared with treating each path independently. Items held

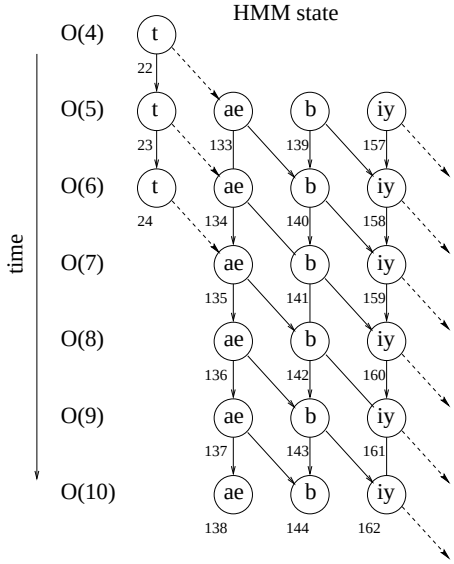

Figure 6: Joint recognition of two words

on the stack now consist of:

1. the language model state (i.e. the last $N-1$ words)
2. the range possible end times for this language model state
3. for each end time:
   (a) the accumulated log probability
   (b) the location in the word hypothesis tree

These items are assembled into a priority queue organised as a heap partially ordered on time. Additionally a separate hash table of language model states is maintained so that any item in the heap may be rapidly indexed by its language model state. The history pointer indicates the leaf of the tree representing all active partial paths.

The heap is initialised by inserting the NULL partial path representing the start of the sentence. Each processing slot consists of popping the item on the top of the heap and extended by one word to give one or more new word histories. The hash table is consulted to decide whether these new paths should be merged with an item already existing on the heap or whether a new item should be added to the heap. Processing terminates where there are no items left on the heap.

```
while heap is not empty
  pop the top item from the heap
  expand the set of hypotheses (section 2)
  for each extension
    if not seen before then
      add to heap
    else
      update appropriate item with these paths
```

A conventional beam search can be used to prune the search space. A record of the highest path probability to every frame is maintained and the search is pruned if the current hypothesis is less likely than a fixed fraction of the highest path probability. An online garbage model is used to control the beam and so to limit the growth of novel path extensions.

Unigram smearing is employed in order to smooth the sharp discontinuity of the language model probabilities which would otherwise occur at the end of the tree.

4. INITIAL IMPLEMENTATION

To verify the principles of the previous sections a large vocabulary system was built for British English. This used the four recurrent neural network acoustic models and the test data from the SQALE evaluation [8] and a 65,000 word language model based on ARPA CSR language model texts and the British National Corpus. The language model contained 11,166,722 bigrams and 13,368,797 trigrams. Context independent phone models are used with a repeated state to enforce a minimum phone duration.

The evaluation was performed on a UltraSparc2 running at

300MHz[7]. A disk-based language model was implemented along the lines of [5]. This component read in complete bigram and trigram tables on first query and cached them according to a simple age based replacement policy. Pruning thresholds were adjusted to obtain real-time performance. Approximate CPU and memory usage is given in table 2

Component	Memory usage	CPU usage
Acoustic models	5M	25%
Lexicon and unigram	5M	-
Pronunciation tree	5M	-
Search	5M	30%
LM access	15M	45%

Table 2: Memory and CPU breakdown

The real-time system had a word error rate of 17.7% compared with the case when it was believed that there were no search errors which had an error rate of 16.8%. The difference is a 5% increase in the number of errors and is of the same order as implementation differences in decoding algorithms[8]. For completely memory resident operation the language model required about 190 Mbyte. The NOWAY process size was about 370 Mbyte and the new decoder with a memory based language model ran in about 200 Mbyte.

All figures in this section are approximate and are provided only to indicate that significant memory savings are possible. Further investigation into pruning thresholds, least-upper bound estimators, language-model smearing techniques, LM cache expiry strategies and standard code optimisations are likely to have a significant effect.

5. RELATED WORK AND CONCLUSIONS

The time-synchronous Viterbi decoder with a tree based lexicon provides the basis for most current large vocabulary decoders. However, if a trigram (or longer N-gram) is to be used as the language model then certain modifications need to be made to allow reasonable execution time and memory usage. The decoder of Odell [3] uses dynamic memory allocation. The decoders from BBN [2] and CMU [5] make a first pass approximation to the trigram to make the state space usage equivalent to a bigram (this approximation is fixed in a subsequent pass). Other similar systems resort to employing a bigram language model on the first pass and rescoring the generated lattice with the more detailed language model on a subsequent pass.

The other main field of decoders are those based on a stack. There may be one stack in total (as in this paper) or one stack for every time frame. Stacks may be ordered, partially ordered or unordered. Notable systems are the IBM stack decoder [1], Doug Paul's stack decoder [4] and NOWAY written by Steve Renals [6]. The system that has been described here differs from all these other implementations in that a stack item consists of a group of partial paths, all of which have the same language model state. This results in more efficient implementation as fragmentation of language model unique paths occurs much less frequently.

The organisation of a decoder that employs a single stack where each item corresponds to a time range, has a unique language model state and is propagated time-first has many advantages:

- The memory usage for search is very small
- Search is fast – real time performance may be achieved
- Like most stack-decoders continuous operation is relatively easy to implement
- The search strategy is cooperative with a standard CPU memory cache in that many operations fall into a small physical memory range
- The search strategy is cooperative with language model access patterns as the aim is that a particular N-gram is asked for only once
- More sophisticated acoustic models such as segmental models are naturally incorporated as the previous state accumulated log probabilities are readily to hand
- Pruning and memory allocation share the same strategy and so work together

In conclusion, the new algorithm allows real-time large vocabulary search to be performed with a modest (5%) increase in the number of errors and requires very little operational memory.

[7]The operating system was Solaris 2.5.1 with 768M of RAM, 2M L2 cache and a 16k/16k L1 cache. The disk drive is a Seagate Barracuda ST34371W with a cache of 512K and a seek time of 8.8ms

[8]The baseline NOWAY error rate using the same acoustic models was 17.5% and the "demo" mode was 18.1%. The differences have been traced to minor implementation details such as the interword-pause and are not thought to be significant

6. ACKNOWLEDGEMENTS

This work was supported in part by the ESPRIT Long Term Research Project (23495) THISL[9] and a grant from Hewlett Packard Laboratories, Bristol. Steve Renals of Sheffield University has patiently explained his NOWAY decoder over the years. Philip Clarkson provided the language model toolkit used and Gary Cook assisted with the processing of the acoustic data, both from Cambridge University.

7. REFERENCES

[1] L. R. Bahl and F. Jelinek. Apparatus and method for determining a likely word sequence from labels generated by an acoustic processor. US Patent 4,748,670, May 1988.

[2] Long Nguyen and Richard Schwartz. Efficient 2-pass n-best decoder. In *Proceedings of the DARPA Speech Recognition Workshop*, pages 100–103, February 1997.

[3] Julian James Odell. *The Use of Context in Large Vocabulary Speech Recognition*. PhD thesis, Cambridge University Engineering Department, March 1995.

[4] D. B. Paul. An efficient A* stack decoder algorithm for continuous speech recognition with a stochastic language model. In *Proc. ICASSP*, volume 1, pages 25–28, San Francisco, 1992.

[5] Mosur K. Ravishankar. *Efficient Algorithms for Speech Recognition*. PhD thesis, Carnegie Mellon University, 1996. Also technicat report CMU-CS-96-143.

[6] Steve Renals and Mike Hochberg. Decoder technology for connectionist large vocabulary speech recognition. Technical Report CUED/F-INFENG/TR.186, Cambridge University Engineering Department, 1994.

[7] Tony Robinson, Mike Hochberg, and Steve Renals. The use of recurrent networks in continuous speech recognition. In Chin-Hui Lee and Frank K. Soong, editors, *Advanced Topics in Automatic Speech and Speaker Recognition*, chapter 7. Kluwer Academic Publishers, 1996.

[8] S. J. Young, M. Adda-Dekker, X. Aubert, C. Dugast, J.-L. Gauvain, D. J. Kershaw, L. Lamel, D. A. Leeuwen, D. Pye, A. J. Robinson, H. J. M. Steeneken, and P. C. Woodland. Multilingual large vocabulary speech recognition: the European SQALE project. *Computer Speech and Language*, 11:73–89, 1997.

[9]http://www.dcs.shef.ac.uk/research/groups/spandh/projects/thisl.html