Some Results on Search Complexity vs Accuracy

Mosur K. Ravishankar

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
$rkm@cs.cmu.edu$

ABSTRACT

This paper presents three different techniques applied in or developed during the 1996 Hub-4 broadcast news transcription task. First, an efficient *shortest path graph search* algorithm is applied to the word lattice created by Viterbi search, producing a globally optimum result. This reduces the word error rate by about 3-10% (relative), depending on the test set. The execution time is at or close to real time for most utterances. Second, a *segmented N-best list* generation algorithm is described for producing compact N-best lists for very long utterances. Finally, a *temporal smoothing* technique is compared to deleted interpolation. On one test set, temporal smoothing reduces the error rate by 3% for an 8% increase in search cost, while the latter improves by 6% for a 50% increase in search cost.

1. Introduction

In this paper we describe the results of a number of search experiments on the 1996 Hub-4 development and evaluation test sets. We have also attempted to document issues that arose during that course and their various resolutions. Some of the techniques were only carried out on the development set and not used in the evaluation.

In what follows, three different techniques are presented and evaluated. The shortest path graph search algorithm is described, along with experimental results, in Section 2. The N-best list generation problem and a new segmented N-best list algorithm are presented in Section 3. Finally, a temporal smoothing technique that overcomes noise by interpolating HMM state scores from neighboring frames is described in Section 4.

2. Shortest Path Search

Most current speech recognition systems, including CMU's, implement multiple search passes. The first pass is usually a full search that produces a word lattice that then constrains the search space for succeeding passes ([7, 6, 8, 9]).

The CMU evaluation decoder ([1]) is configured as follows. An initial Viterbi beam search produces a single recognition hypothesis as well as a word lattice that includes word segmentations and acoustic scores. The *shortest path graph search* produces another single, globally optimal hypothesis (using a trigram grammar) from the word lattice. Finally, N-best lists are generated from the word lattice.

In broadcast speech, the presence of background or channel noise, spontaneous speaking styles, the unconstrained duration of utterances, all tend to aggravate N-best list generation. There are a vary large number of closely competing hypotheses. Therefore, the N-best list may not actually be the best N hypotheses. The shortest path graph search enables us to find the globally optimum word sequence in the word lattice. This hypothesis is also injected into the N-best list.

The final recognition results after rescoring the N-best list are presented in [1]. In this section we evaluate the shortest path search algorithm. It is interesting since it is compact and easy to implement, computationally efficient, and produces results fairly close to N-best rescoring, especially on clean speech. (It has previously been used in the CMU Sphinx-II real-time decoder as a second, rescoring step.)

2.1. Problem Formulation

The word lattice produced by the Viterbi beam search is transformed into a Directed Acyclic Graph (DAG) as follows: A node in the graph represents an instance of a word w with a particular begin time t. We denote such a node by (w, t). There can be a range of end-times for this node. We create an edge from a node (w_i, t_i) to node (w_j, t_j) *iff* $t_j - 1$ is one of the end times of (w_i, t_i).

The cost associated with an edge from a node (w_i, t_i) to (w_j, t_j) is formed from two components: an acoustic score or log-likelihood and a language model log-probability. The former is the acoustic score for w_i from time t_i to $t_j - 1$, with w_j being the phonetic right context. For the language model log-probability, let us first consider the case of a simple bigram grammar. The log-probability is simply $log(P(w_j|w_i))$. The total edge cost is the weighted sum of the two[1]. It is independent of other edges in the DAG.

The DAG has distinguished *start* and *end* nodes corresponding to the sentence initial and sentence final silence words, respectively. We compute the best (least cost) path through the DAG from the *start* node to the *end* node using any of the standard shortest-path graph algorithms. This gives the single best recognition hypothesis for the utterance, given the word lattice.

Longer distance language models can be handled, in principle, by replicating nodes and additional edges as necessary. This is still practical with trigram grammars, but with 4-gram or higher-order grammars the growth in the DAG size can become unmanageable. (With a trigram model, in fact, it is not actually necessary to physically expand the DAG. One can reformulate the problem in terms of an edge cost given the predecessor edge, and applying language model probabilities dynamically.)

The best path search has a number of advantages:

[1] Usually, the language model log-probability is weighted by a *language weight* whose optimum value is determined empirically.

- The result is globally optimum.
- The algorithm is implemented efficiently and compactly.
- There is no pruning, unlike in the case of beam search or A* search algorithms. This is very useful in empirically optimizing parameters such as language weight and word insertion penalty. When any such parameter is varied in a search algorithm that employs pruning, the *effective* pruning threshold, and hence the active search subspace, is changed significantly. Thus, comparisons of results with different parameter values can be misleading. Since the shortest path algorithm searches the entire DAG with no pruning, the optimum parameter values can be determined exactly.

The practical applicability of this algorithm is occasionally limited, however, by the size of the DAG. Noisy speech can result in very large DAGs. A second factor is the presence of *noise word* models (such as silence and filled pauses) in the Sphinx-3 system. Noise words are inserted in a manner transparent to the n-gram language model. In creating the DAG, additional links must be created to bypass noise word nodes, increasing the size of the DAG significantly. Our implementation aborts a DAG search if pre-specified computational or memory limits are exceeded.

2.2. Experimental Results

As described in [1], the CMU evaluation run consisted of two phases. In phase-1, the test set was decoded using conventional Viterbi beam search, followed by the best path DAG search algorithm. The result of the latter was used for unsupervised MLLR adaptation.

In phase-2, the adapted acoustic models were used for decoding. In addition, the context-dependent (CD) state output probabilities were interpolated with corresponding context-independent (CI) ones to produce smoother models ([4]). Once again, a complete Viterbi beam search was followed by DAG search. An N-best rescoring process was the final step; the best path search results were added to the N-best lists before the rescoring. (Also, the DAG search and N-best rescoring steps used different trigram language models and weights, which were optimized independently.)

We present the word error rates (WER) of each step in each phase of the evaluation run. In particular, we focus on the performance of the best path search relative to the other steps. Tables 1 and 2 show the word error rates of individual search steps in the partitioned and unpartitioned evaluations, respectively.

	Tot	*F0*	*F1*	*F2*	*F3*	*F4*	*F5*	*FX*
Vit	38.8	28.8	33.8	44.9	45.1	45.6	45.2	64.6
Dag	37.3	27.2	32.4	43.2	43.3	45.7	45.8	61.8

Phase-1 (Before Adaptation)

	Tot	*F0*	*F1*	*F2*	*F3*	*F4*	*F5*	*FX*
Vit	36.3	27.2	33.2	40.4	37.7	44.1	37.1	58.5
Dag	35.5	26.1	32.3	39.7	37.3	43.9	38.1	57.8
Nbs	34.9	25.8	32.1	38.6	36.6	43.7	36.5	55.8

Phase-2 (After Adaptation)
(Vit: Viterbi; Dag: Best path; Nbs: N-best search)

Table 1: WER of individual steps in Hub-4 Partitioned Evaluation.

	Tot	*F0*	*F1*	*F2*	*F3*	*F4*	*F5*	*FX*
Vit	39.1	27.2	34.4	45.8	50.0	45.6	40.8	66.7
Dag	37.8	26.0	33.5	44.7	48.4	45.0	40.8	62.9

Phase-1 (Before Adaptation)

	Tot	*F0*	*F1*	*F2*	*F3*	*F4*	*F5*	*FX*
Vit	37.3	25.7	34.0	42.6	49.5	43.0	37.5	60.9
Dag	36.5	24.8	33.7	39.8	48.8	42.5	38.8	60.3
Nbs	35.9	24.7	33.1	39.1	48.4	42.1	35.5	58.3

Phase-2 (After Adaptation)
(Vit: Viterbi; Dag: Best path; Nbs: N-best search)

Table 2: WER of individual steps in Hub-4 Unpartitioned Evaluation.

Overall, we see an improvement in WER as a result of the best path search. However, the improvement varies with the test condition. It is most prominent in the clean, or F0 condition, where it comes closest to N-best rescoring. The performance improvement is more variable in the noisier conditions.

We now look at the execution speed of the best path DAG search algorithm. Figure 1 shows the distribution of the normalized execution time (CPU time/utterance duration) for each utterance[2]. It is a combined graph for both phases of both the partitioned and unpartitioned evaluations.

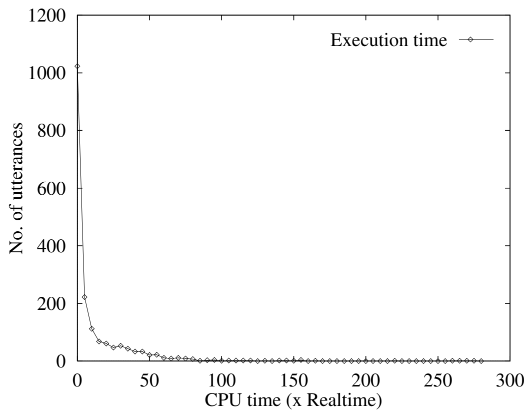

Figure 1: Best Path Search Execution Time Distribution

Most utterances are processed at or close to real time. The utterances with very long execution times are typically noisy, or badly articulated speech. The word lattice is very large in these cases to begin with.

3. N-best List Generation

The CMU evaluation system included N-best list generation ([7, 6]) for rescoring with higher-order N-gram language models. The unconstrained length of utterances in the broadcast news test implies that N-best lists must be extremely long in order to improve the chances of the correct transcription being included in the list. In short, the list size must grow exponentially with utterance length to

[2] The experiments were run on high end DEC Alpha and Pentium Pro workstations.

maintain a constant rate. We considered a number of options to deal with this situation:

1. Using a front-end segmenter ([5]) to break long utterances into shorter segments.
2. Decoding complete utterances in the Viterbi pass, but using silence information obtained thereby to identify subsegment breaks during N-best generation.
3. Generating *segmented N-best lists*; segment breaks are based on *articulation points* (borrowing a term from graph theory) in the word lattice. (See Section 3.1 below.)

In all cases, the shorter the length of the segments, the better the quality of the N-best lists. However, segment breaks also cause breaks in linguistic context, and occasionally within a word. The loss of relevant context information can introduce additional errors when N-best lists for each segment are rescored independently. This problem becomes more acute as the average segment size shrinks. The obvious solution is to rescore N-best lists in the context of neighboring ones. But because of time and resource constraints we were not able to experiment with such possibilities before the evaluation deadline.

Ultimately, because of time constraints, we settled on the front-end segmentation as the simplest scheme. A front-end segmenter is more prone to errors, compared to HMM-based models that can detect silences very accurately. To minimize errors introduced by the segmenter and by linguistic context breaks, we settled on a relatively long average segment duration of about 30 sec.

Though it was not used in the evaluations, we did implement the segmented N-best list generation algorithm, which has some similarities to the IBM envelope search ([11]). It can produce compact, low error N-best lists with efficiency and is described below.

3.1. Segmented N-best Lists

This algorithm first constructs a DAG from the word lattice produced by the Viterbi search, as described in Section 2. The DAG provides candidates for paths to be explored during the N-best search. The result of the search is a set of segmentations of the utterance, and separate N-best lists within each segment. (In principle, segments can be as short as one word.) This allows the generation of compact yet sufficiently rich N-best lists. Furthermore, the segmentation leads to greater efficiency in terms of processor and memory usage.

Like all N-best algorithms, this one works by maintaining a number of partial paths and repeatedly expanding the best one by one word. In this case, several lists of partial hypotheses are maintained, one list for each possible end time in the DAG. Each list is sorted in descending order of the partial path score. The lists are initialized with the utterance initial silence and all words that can succeed it (as determined by the DAG).

The algorithm proceeds in a time-synchronous manner, repeatedly expanding the best partial path in the earliest end-time list that is non-empty. However, whenever the algorithm moves in time to a new non-empty list, it first checks if that is also the last non-empty list. If so, it marks an *articulation point*, and hence a segment boundary. That is, all complete paths must pass through one of the partial paths ending in this list. A segment N-best list is generated at this point and the list compacted to include only the unique grammar states. Since all future extension to partial paths must proceed from one of these, there is no need to maintain duplicate grammar states. The complete algorithm follows:

```
/* plist[t] = partial path list with end time=t */
for (t = 1 to T) { /* T is utterance length */
  /* Look for articulation point at t */
  if (plist[t+1]..plist[T] are all empty) {
    output plist[t]; /* Nbest list for segment */

    compact plist[t]; /* Retain only unique grammar
    states; e.g. last two words with trigram LM */
    /* Start of new segment */
  }

  /* Expand paths in plist[t]; usual N-best step */
  while (plist[t] not empty) {
    p = best partial path in plist[t];
    pop p off plist[t];
    n = DAG node corresponding to final word in p;

    for (each successor node n' to n in DAG) {
      for (each end time t' for n') {
        p' = new partial path by appending n' to p;
        compute path score for p'; /* add acoustic
                               and LM scores for n' */
        insert p' in plist[t'];
      }
    }
  }
}
```

A pruning *beamwidth* is employed to consider only the more promising paths. The beam is applied to each N-best list independently.

The significant feature in the above algorithm is the compaction step after a segment N-best list is output at an articulation point. It prevents the total number of partial hypotheses from growing exponentially with the length of the utterance. The resulting efficiency allows a greater number of hypotheses to be explored in each segment. Yet, by treating each segment independently, the overall size of the list is kept compact.

This algorithm has been evaluated on the 1994 Hub1 evaluation test set using the standard 20K vocabulary. The lattice error rate on this set is 5.5%. The straightforward N-best list (unsegmented), with a list size of 500, has about 8.4% error rate. This corresponds to an oracle somehow chosing the best hypothesis from each N-best. The segmented N-best algorithm presented here has an error rate of 7.3%, a relative improvement of about 13%. At the same time, the segmented N-best files are about an order of magnitude smaller than the unsegmented ones.

4. Temporal Smoothing of State Scores

One of the observations while testing the Sphinx-3 continuous acoustic models on the H4 development set was that some word instances failed to even show up in the word lattice. They were pruned out completely. However, the same instances were decoded successfully with semi-continuous acoustic models. This basically pointed out a mismatch between the sharp and well-defined continuous HMM models, and the noisy speech in the broadcast news data.

Two obvious ways to improve the recognition were the following:

- Widening the search beamwidth to reduce pruning errors.

- Smoothing the acoustic models, for example, by interpolating context dependent (CD) state scores with corresponding context independent ones ([4]).

The latter showed some improvement in recognition accuracy. However, in both cases, the effect was to increase the number of active models, and slow down the search significantly.

But there is also a third alternative. If, in fact, the input speech is noisy, perhaps smoothing the input data instead of the models can improve performance. The basic idea is to smooth HMM state scores in one frame by interpolating them with corresponding scores from a neighboring frame:

$$P'(s,t) = \lambda P(s,t) + (1-\lambda)P(s,t-1)$$

where, s is any HMM state, $P(s,t)$ is its unsmoothed state score at time t, P' is the corresponding smoothed score, and λ is an empirically determined constant ($0 <= \lambda <= 1$). In a sense, this sends the speech through a low-pass filter, removing noise from it. Interestingly, we found that such smoothing provides some improvement in recognition accuracy. Moreover, the search speed is affected much less than in the alternatives mentioned earlier.

4.1. Experimental Results

We applied each of the alternatives considered above in the Viterbi beam search. The test sets were the F0 and F1 conditions of the 1996 Hub-4 development set. Specifically, four different configurations were tried:

1. *BASE:* Baseline acoustic models with normal pruning beamwidth settings (the same as in the evaluations).
2. *WIDE:* Baseline acoustic models with wider beamwidth.
3. *CD-CI:* Interpolation of context-dependent and independent state scores using baseline acoustic models, and normal pruning beamwidth.
4. *TIME:* Temporal smoothing of state scores using baseline acoustic models, and normal pruning beamwidth.

In Table 3 we show the word error rates and the number of HMM models evaluated per frame in each of the four experiments above. On F0, simply widening the beamwidth is utterly useless. There is about a 75% increase in search complexity for no significant improvement in recognition accuracy. The surprising result is that temporal smoothing does much better, (though not as well as CD-CI smoothing) with less than 10% increase in search complexity. The WER on the F1 condition is unaffected by any of the techniques.

	F0		F1	
	WER	*HMM/Frm*	*WER*	*HMM/Frm*
BASE	18.8	61K	36.2	81K
WIDE	18.6	106K	36.2	143K
CD-CI	17.6	92K	36.5	137K
TIME	18.2	65K	36.1	87K

Table 3: Performance on 1996 H4 Dev. Set With Different Smoothing and Beamwidth Configurations.

We note that we did not employ this heuristic in the evaluations since we did not have sufficient time to evaluate its utility or robustness.

5. Summary

We have described three different techniques that were used during the 1996 Hub-4 evaluations or tested during the development process. We have seen that the shortest path graph search algorithm can be used to efficiently generate a globally optimum hypothesis. On the Hub-4 task, it reduces word error rate consistently by about 3-10% while mostly running at or close to real time. It is also useful in optimizing other global search parameters such as language weight and word insertion penalty. We have also shown that temporal smoothing of state output probabilities results in about 3% reduction in WER on the F0 condition of the development set with less than 10% increase in computational load.

Acknowledgements. This research was sponsored by the Department of the Navy, Naval Research Laboratory under Grant No. N00014-93-1-2005.

I would like to thank Paul Placeway, Roni Rosenfeld, Eric Thayer, and other members of the Sphinx group for their comments on the contents of this paper.

References

1. Placeway, P. *et al*, *The 1996 Hub-4 Sphinx-3 System*, ARPA SLT Workshop, Feb. 1997.
2. Seymore, K., Chen, S., Eskenazi, M., and Rosenfeld, R., *Language and Pronunciation Modeling in the 1996 Hub 4 Evaluation*, ARPA SLT Workshop, Feb. 1997.
3. Parikh, V., Raj, B., and Stern, R. *Sphinx-III: Adaptation and Compensation for the Hub4-96 Task*, ARPA SLT Workshop, Feb. 1997.
4. Huang, X.D., Hwang, M-Y., Jiang, L., and Mahajan, M., *Deleted Interpolation and Density Sharing for Continuous Hidden Markov Models*, ICASSP-96.
5. Siegler, M., Jain, U., Raj, B., and Stern, R. *Automatic Segmentation, Classification and Clustering Broadcast News Audio*, ARPA SLT Workshop, Feb. 1997.
6. Alleva, F., Huang, X., and Hwang, M. *An Improved Search Algorithm for Continuous Speech Recognition.* ICASSP, 1993.
7. Schwartz, R. and Chow, Y.L. *The Optimal N-Best Algorithm: An Efficient Procedure for Finding Multiple Sentence Hypotheses.* ICASSP, Apr. 1990.
8. Woodland, P.C. Leggetter, C.J., Odell, J.J., Valtchev, V. and Young, S.J. *The Development of the 1994 HTK Large Vocabulary Speech Recognition System.*, ARPA SLT Workshop, Jan. 1995, pp 104-109.
9. Murveit, H., Butzberger, J., Digalakis, V. and Weintraub, M. *Large-Vocabulary Dictation Using SRI's Decipher Speech Recognition System: Progressive Search Techniques.* ICASSP, Apr. 1993, vol.II, pp. II-319 – II-322.
10. Leggetter, C.J. and Woodland, P.C. *Flexible Speaker Adaptation Using Maximum Likelihood Linear Regression*, ARPA SLT Workshop, Jan. 1995, pp 110-115.
11. Gopalakrishnan, P.S., Bahl, L.R., and Mercer, R.L., *A Tree Search Strategy for Large-Vocabulary Continuous Speech Recognition*, ICASSP, May 1995, pp 572-575.