PROGRESSIVE-SEARCH ALGORITHMS FOR LARGE-VOCABULARY SPEECH RECOGNITION

Hy Murveit
John Butzberger
Vassilios Digalakis
Mitch Weintraub

SRI International

ABSTRACT

We describe a technique we call *Progressive Search* which is useful for developing and implementing speech recognition systems with high computational requirements. The scheme iteratively uses more and more complex recognition schemes, where each iteration constrains the search space of the next. An algorithm, the *Forward-Backward Word-Life Algorithm,* is described. It can generate a word lattice in a progressive search that would be used as a language model embedded in a succeeding recognition pass to reduce computation requirements. We show that speed-ups of more than an order of magnitude are achievable with only minor costs in accuracy.

1. INTRODUCTION

Many advanced speech recognition techniques cannot be developed or used in practical speech recognition systems because of their extreme computational requirements. Simpler speech recognition techniques can be used to recognize speech in reasonable time, but they compromise word recognition accuracy. In this paper we aim to improve the speed/accuracy trade-off in speech recognition systems using progressive search techniques.

We define *progressive search* techniques as those which can be used to efficiently implement other, computationally burdensome techniques. They use results of a simple and fast speech recognition technique to constrain the search space of a following more accurate but slower running technique. This may be done iteratively—each progressive search pass uses a previous pass' constraints to run more efficiently, and provides more constraints for subsequent passes.

We will refer to the faster speech recognition techniques as "earlier-pass techniques", and the slower more accurate techniques as "advanced techniques." Constraining the costly advanced techniques in this way can make them run significantly faster without significant loss in accuracy.

The key notions in progressive search techniques are:

1. An early-pass speech recognition phase builds a lattice, which contains all the likely recognition unit strings (e.g. word sequences) given the techniques used in that recognition pass.
2. A subsequent pass uses this lattice as a grammar that constrains the search space of an advanced technique (e.g., only the word sequences contained in a word lattice of pass p would be considered in pass p+1).

Allowing a sufficient breadth of lattice entries should allow later passes to recover the correct word sequence, while ruling out very unlikely sequences, thus achieving high accuracy and high speed speech recognition.

2. PRIOR ART

There are three important categories of techniques that aim to solve problems similar to the ones the progressive search techniques target.

2.1. Fast-Match Techniques

Fast-match techniques[1] are similar to progressive search in that a coarse match is used to constrain a more advanced computationally burdensome algorithm. The fast match, however, simply uses the local speech signal to constrain the costly advanced technique. Since the advanced techniques may take advantage of non-local data, the accuracy of a fast-match is limited and will ultimately limit the overall technique's performance. Techniques such as progressive search can bring more global knowledge to bear when generating constraints, and, thus, more effectively speed up the costly techniques while retaining more of their accuracy.

2.2. N-Best Recognition Techniques

N-best techniques[2] are also similar to progressive search in that a coarse match is used to constrain a more computationally costly technique. In this case, the coarse matcher is a complete (simple) speech recognition system. The output of the N-best system is a list of the top N most likely sentence hypotheses, which can then be evaluated with the slower but more accurate techniques.

Progressive search is a generalization of N-best—the earlier-pass technique produces a graph, instead of a list of N-best sentences. This generalization is crucial because N-best is only computationally effective for N in the order of tens or hundreds. A progressive search word graph can effectively account for orders of magnitude more sentence hypotheses. By limiting the advanced techniques to just searching the few top N sentences, N-best is destined to limit the effectiveness of the advanced techniques and, consequently, the overall system's accuracy. Furthermore, it does not make much sense to use N-best in an iterative fashion as it does with progressive searches.

2.3. Word Lattices

This technique is the most similar to progressive search. In both approaches, an initial-pass recognition system can generate a lattice of word hypotheses. Subsequent passes can search through the lattice to find the best recognition hypothesis. It should be noted that, although we refer to lattices as word lattices, they could be used at other linguistic level, such as the phoneme, syllable, e.t.c.

In the traditional word-lattice approach, the word lattice is viewed as a scored graph of possible segmentations of the input speech. The lattice contains information such as the acoustic match between the input speech and the lattice word, as well as segmentation information.

The progressive search lattice is not viewed as a scored graph of possible segmentations of the input speech. Rather, the lattice is simply viewed as a word-transition grammar which constrains subsequent recognition passes. Temporal and scoring information is intentionally left out of the progressive search lattice.

This is a critical difference. In the traditional word-lattice approach, many segmentations of the input speech which could not be generated (or scored well) by the earlier-pass algorithms will be eliminated for consideration before the advanced algorithms are used. With progressive-search techniques, these segmentations are implicit in the grammar and can be recovered by the advanced techniques in subsequent recognition passes.

3. Building Progressive Search Lattices

The basic step of a progressive search system is using a speech recognition algorithm to make a lattice which will be used as a grammar for a more advanced speech recognition algorithm. This section discusses how these lattices may be generated. We focus on generating word lattices, though these same algorithms are easily extended to other levels.

3.1. The Word-Life Algorithm

We implemented the following algorithm to generate a word-lattice as a by-product of the beam search used in recognizing a sentence with the DECIPHER™ system[4-7].

1. For each frame, insert into the table *Active(W, t)* all words *W* active for each time t. Similarly construct tables *End(W, t)* and *Transitions*(W_1, W_2, *t*) for all words ending at time t, and for all word-to-word transition at time t.
2. Create a table containing the word-lives used in the sentence, *WordLives*(*W*, T_{start}, T_{end}). A *word-life* for word *W* is defined as a maximum-length interval (frame T_{start} to T_{end}) during which some phone in word *W* is active. That is,
   $W \in Active(W, t), T_{start} \le t \le T_{end}$
3. Remove word-lives from the table if the word never ended between T_{start} and T_{end}, that is, remove *WordLives*(*W*, T_{start}, T_{end}) if there is time *t* between T_{start} and T_{end} where *End(W, t)* is true.
4. Create a finite-state graph whose nodes correspond to word-lives, whose arcs correspond to word-life transitions stored in the *Transitions* table. This finite state graph, augmented by language model probabilities, can be used as a grammar for a subsequent recognition pass in the progressive search.

This algorithm can be efficiently implemented, even for large vocabulary recognition systems. That is, the extra work required to build the "word-life lattice" is minimal compared to the work required to recognize the large vocabulary with a early-pass speech recognition algorithm.

This algorithm develops a grammar which contains all whole-word hypotheses the early-pass speech recognition algorithm considered. If a word hypothesis was active and the word was processed by the recognition system until the word finished (was not pruned before transitioning to another word), then this word will be generated as a lattice node. Therefore, the size of the lattice is directly controlled by the recognition search's beam width.

This algorithm, unfortunately, does not scale down well—it has the property that small lattices may not contain the best recognition hypotheses. This is because one must use small beam widths to generate small lattices. However, a small beam width will likely generate pruning errors.

Because of this deficiency, we have developed the Forward/Backward Word-Life Algorithm described below.

3.2. Extending the Word-Life Algorithm Using Forward And Backward Recognition Passes

We wish to generate word lattices that scale down gracefully. That is, they should have the property that when a lattice is reduced in size, the most likely hypotheses remain and the less likely ones are removed. As was discussed, this is not the case if lattices are scaled down by reducing the beam search width.

The forward-backward word-life algorithm achieves this scaling property. In this new scheme, described below, the size of the lattice is controlled by the *LatticeThresh* parameter.

1. A standard beam search recognition pass is done using the early-pass speech recognition algorithm. (None of the lattice building steps from Section 3.1 are taken in this forward pass).
2. During this forward pass, whenever a transition leaving word *W* is within the beam-search, we record that probability in *ForwardProbability(W,frame)*.
3. We store the probability of the best scoring hypothesis from the forward pass, *Pbest*, and compute a pruning value
   Pprune = Pbest / LatticeThresh.

4. We then recognize the same sentence over again using the same models, but the recognition algorithm is run backwards[1].

5. The lattice building algorithm described in Section 3.1 is used in this backward pass with the following exception. During the backward pass, whenever there is a transition between words W_i and W_j at time t, we compute the overall hypothesis probability P_{hyp} as the product of *ForwardProbability(W_j,t-1)*, the language model probability $P(W_i|W_j)$, and the Backward pass probability that W_i ended at time t (i.e. the probability of starting *word* W_i at time t and finishing the sentence). If $P_{hyp} < P_{prune}$, then the backward transition between W_i and W_j at time t is blocked.

Step 5 above implements a backwards pass pruning algorithm. This both greatly reduces the time required by the backwards pass, and adjusts the size of the resultant lattice.

4. Progressive Search Lattices

We have experimented with generating word lattices where the early-pass recognition technique is a simple version of the DECIPHER™ speech recognition system, a 4-feature, discrete density HMM trained to recognize a 5,000 vocabulary taken from DARPA's WSJ speech corpus. The test set is a difficult 20-sentence subset of one of the development sets.

We define the number of errors in a single path p in a lattice, *Errors(p)*, to be the number of insertions, deletions, and substitutions found when comparing the words in p to a reference string. We define the number of errors in a word lattice to be the minimum of *Errors(p)* for all paths p in the word lattice.

The following tables show the effect adjusting the beam width and *LatticeThresh* has on the lattice error rate and on the lattice size (the number of nodes and arcs in the word lattice). The grammar used by the has approximately 10,000 nodes and 1,000,000 arcs. The the simple recognition system had a 1-best word error-rate ranging from 27% (beam width 1e-52) to 30% (beam width 1e-30).

Table 1: Effect Of Pruning On Lattice Size

Beam Width 1e-30

Lattice Thresh	nodes	arcs	# errors	%word error
1e-5	60	278	43	10.57
1e-9	94	541	34	8.35
1e-14	105	1016	30	7.37
1e-18	196	1770	29	7.13
1e-32	323	5480	23	5.65
1e-45	372	8626	23	5.65
inf	380	9283	23	5.65

Beam Width 1e-34

Lattice Thresh	nodes	arcs	# errors	%word error
1e-5	64	299	28	6.88
1e-9	105	613	20	4.91
1e-14	141	1219	16	3.93
1e-18	260	2335	15	3.69
1e-23	354	3993	15	3.69
1e-32	537	9540	15	3.69

Beam Width 1e-38

Lattice Thresh	nodes	arcs	# errors	%word error
1e-14	186	1338	14	3.44
1e-18	301	2674	13	3.19
1e-23	444	4903	12	2.95

Beam Width 1e-42

Lattice Thresh	nodes	arcs	# errors	%wd error
1e-14	197	1407	13	3.19
1e-18	335	2926	11	2.70
1e-23	520	5582	10	2.46

Beam Width 1e-46

Lattice Thresh	nodes	arcs	# errors	%word error
1e-14	201	1436	13	3.19
1e-18	351	3045	10	2.46
1e-23	562	5946	10	2.46

Beam Width 1e-52

Lattice Thresh	nodes	arcs	# errors	%word error
1e-14	216	1582	12	2.95
1e-18	381	3368	9	2.21

The two order of magnitude reduction in lattice size has a significant impact on HMM decoding time. Table 2 shows the per-sentence computation time required for the above test set when computed using a Sparc2 computer, for both the original grammar, and word lattice grammars generated using a *LatticeThresh* of 1e-23.

1. Using backwards recognition the sentence is processed from last frame to first frame with all transitions reversed.

Table 2: Lattice Computation Reductions

Beam Width	Forward pass recognition time (secs)	Lattice recognition time (secs)
le-30	167	10
le-34	281	16
le-38	450	24
le-46	906	57
le-52	1749	65

5. Applications of Progressive Search Schemes

Progressive search schemes can be used in the same way N-best schemes are currently used. The two primary applications we've had at SRI are:

5.1. Reducing the time required to perform speech recognition experiments

At SRI, we've been experimenting with large-vocabulary tied-mixture speech recognition systems. Using a standard decoding approach, and average decoding times for recognizing speech with a 5,000-word bigram language model were 46 times real time. Using lattices generated with beam widths of le-38 and a *LatticeThresh* of le-18 we were able to decode in 5.6 times real time). Further, there was no difference in recognition accuracy between the original and the lattice-based system.

5.2. Implementing recognition schemes that cannot be implemented with a standard approach.

We have implemented a trigram language model on our 5,000-word recognition system. This would not be feasible using standard decoding techniques. Typically, continuous-speech trigram language models are implemented either with fastmatch technology or, more recently, with N-best schemes. However, it has been observed at BBN that using an N-best scheme (N=100) to implement a trigram language model for a 20,000 word continuous speech recognition system may have significantly reduced the potential gain from the language model. That is, about half of the time, correct hypotheses that would have had better (trigram) recognition scores than the other top-100 sentences were not included in the top 100 sentences generated by a bigram-based recognition system[8].

We have implemented trigram-based language models using word-lattices, expanding the finite-state network as appropriate to unambiguously represent contexts for all trigrams. We observed that the number of lattice nodes increased by a factor of 2-3 and the number of lattice arcs increased by a factor of approximately 4 (using lattices generated with beam widths of le-38 and a *LatticeThresh* of le-18). The resulting decoding times increased approximately by 50% when using trigram lattices instead of bigram lattices.

ACKNOWLEDGEMENTS

We gratefully acknowledge support for this work from DARPA through Office of Naval Research Contract N00014-92-C-0154. The Government has certain rights in this material. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the government funding agencies.

REFERENCES

1. Bahl, L.R., de Souza, P.V., Gopalakrishnan, P.S., Nahamoo, D., and M. Picheny, "A Fast Match for Continuous Speech Recognition Using Allophonic Models," *1992 IEEE ICASSP*, pp. I-17-21.
2. Schwartz, R., Austin, S., Kubala, F., Makhoul, J., Nguyen, L., Placeway, P., and G. Zavaliagkos, "New uses for the N-Best Sentence Hypotheses Within the BYBLOS Speech Recognition System", *1992 IEEE ICASSP*, pp. I-1-4.
3. Chow, Y.L., and S. Roukos, "Speech Understanding Using a Unification Grammar", *1989 IEEE ICASSP*, pp. 727-730
4. H. Murveit, J. Butzberger, and M. Weintraub, "Performance of SRI's DECIPHER Speech Recognition System on DARPA's CSR Task," 1992 DARPA Speech and Natural Language Workshop Proceedings, pp 410-414
5. Murveit, H., J. Butzberger, and M. Weintraub, "Reduced Channel Dependence for Speech Recognition," 1992 DARPA Speech and Natural Language Workshop Proceedings, pp. 280-284.
6. H. Murveit, J. Butzberger, and M. Weintraub, "Speech Recognition in SRI's Resource Management and ATIS Systems," 1991 DARPA Speech and Natural Language Workshop, pp. 94-100.
7. Cohen, M., H. Murveit, J. Bernstein, P. Price, and M. Weintraub, "The DECIPHER™ Speech Recognition System," *1990 IEEE ICASSP*, pp. 77-80.
8. Schwartz, R., BBN Systems and Technologies, Cambridge MA, Personal Communication