TOWARDS A ROBUST REAL-TIME DECODER

Jason Davenport, Richard Schwartz, Long Nguyen
BBN Technologies, GTE Internetworking
Cambridge, MA 02138, USA

ABSTRACT

In this paper we present several algorithms that speed up our BBN BYBLOS decoder. We briefly describe the techniques that we have used before this year. Then we present new techniques that speed up the recognition search by a factor of 10 with little effect on accuracy using a combination of Fast Gaussian Computation, grammar spreading, and grammar caching, within the 2-Pass n-best paradigm. We also describe our decoder metering strategy, which allows us to conveniently test for search errors. Finally, we describe a grammar compression technique that decreases the storage needed for each additional ngram to only 10 bits.

1. INTRODUCTION

Continuous speech recognition with a very large vocabulary requires a large amount of computation. The amount of computation also depends to a large degree on the quality of speech, with the computation increasing by a significant factor for more natural speech. Research systems frequently use 200-500 times real time to achieve the highest possible accuracy. While we can always decrease the computation by using an aggressive beam search pruning strategy [12], our goal here is to decode the speech with the least amount of computation, while still obtaining accuracy as close as possible to that of our best research system.

There have been several algorithms proposed to decrease the search space [10][11][13]. We can also reduce the search space using multiple-pass searches that perform successively more detailed analyses of the sentence, using n-best or lattice structures as intermediate reduced search spaces [5][15]. Finally, there are algorithms dealing with particular parts of the computation, such as the efficient evaluation of large numbers of Gaussian probability densities [3].

The BBN BYBLOS research decoder uses a combination of several techniques to reduce computation. In Section 2, we briefly review the algorithms that we used as of a year ago to obtain high speed. These include the forward-backward search, a single-tree forward fast match, a two-pass n-best algorithm, and a grammar caching algorithm.

During the past year we have developed new algorithms to increase the speed of recognition by another factor of 10, while decreasing the loss for using reduced computation. These new algorithms include: a simple fast Gaussian computation, grammar spreading that reduces the loss due to pruning, metering techniques that allow us to directly and easily measure the loss from pruning without having to greatly increase the computation, and an algorithm that greatly reduces the memory needed to store very large ngram language models in limited space. We will describe these new algorithms in Section 3.

The result of the new algorithms has been an additional factor of 10 speedup over the past year, with a decreased loss due to aggressive pruning.

2. PREVIOUS ALGORITHMS IN BYBLOS

The 1997 BYBLOS decoder used a combination of several strategies. Here we review the previous algorithms used in the decoder.

2.1 Forward-Backward Search

The 1997 BYBLOS research decoder used a combination of several strategies. First, we used the Forward-Backward Search [5][6]. In the forward pass we use an approximate fast match that finds the likelihoods of several ending words in each frame. Then, in the backward pass, we use a more detailed model, but prune the search using the normalized forward-backward score. We have shown that the computation for the backward pass can be sped up by two orders of magnitude without loss even when the models used in the forward pass are considerably less accurate than those used in the backward pass. This allows us to use very coarse or approximate models in the forward pass in order to decrease the search time.

2.2 Single-Phonetic-Tree Fast-Match

In 1992 we developed a very fast forward-pass algorithm [4][14] to go along with the Forward Backward Search. This algorithm allowed us to perform continuous speech recognition for dictation quality speech with large vocabularies. The algorithm uses a single phonetic tree instead of multiple trees. We continually reevalute the grammar probabilities in order to estimate the probability of ending words. To further reduce computation, we use composite triphone models averaged over the various right contexts in the tree, and only consider a few ending words at each frame. We use partial language model probabilities in order to obtain the benefit of the language model as early as possible. Despite the fact that this algorithm by itself results in a much higher error rate, we show here that it almost never causes a search error.

2.3 N-Best

We have developed several algorithms for computing the n-best hypotheses [7][8], where our goal is to reduce the computation but still retain enough good hypotheses so that rescoring with a better model will increase accuracy. The most recent algorithm [1] uses an inexpensive two-pass algorithm that costs very little. It consists in making a lattice of word hypotheses from the

backward pass, inserting trigram probabilities into the lattice, and then writing out the n-best sentence hypotheses without reevaluating acoustic scores.

2.4 Ngram Cache

The search algorithm we use depends heavily on being able to look up thousands of language model probabilities for each frame efficiently. Ideally, we would store the probabilities in a simple array, but the number of language model probabilities is far too large. Typically, one stores the words that have been observed to follow any particular state (i.e., history) in a list of words and their associated probabilities. But searching this list during recognition can be expensive. The average number of words for a bigram state is around 1000. A binary search requires 10 test and branch operations.

Instead, we create the appropriate columns of the array as they are needed. We allocate several (say 100) cache arrays, each the size of the vocabulary, and initialize them to NULL. We create a list, one element per state, of NULL cache pointers that will be used to keep track of the (100) states that are cached. When we need to look up a language model probability for a particular history, we first check the cache pointer for the state of that history. If it has not been cached (cache pointer is NULL), then we find the compressed block of language model probabilities for that history's state, point the history's state cache pointer to one of the empty cache arrays, and set the entries in the array for each of the stored probabilities. Once this is done, we can look up the probability for any following word with a single indirect access.

We typically look up a very large number of words for a given state. Thus, the initial cost of setting several (say 1000) entries is small compared to the savings. We manage this cache on the basis of the last access time. Note that all stored probabilities are represented as scaled log probabilities using just 8 bits. During recognition we look up the floating point value in the appropriate precomputed table with 256 values.

3. NEW ALGORITHMS

Here we present several algorithms that we have used over the past year to further reduce computation, speed up the research process and reduce memory requirements.

3.1 Fast Gaussian Computation (FGC)

Generally we try to avoid speeding up one part of the computation, since it doesn't result in a large factor. However, our research speech recognition systems typically use a very large number of Gaussian probability densities to obtain high accuracy. Thus, the computation can be dominated by Gaussian evaluations. When the beam search is very wide (slow) we spend 93% of decoder computation calculating Gaussian distances in the backward pass and 33% in the forward pass. But with heavy pruning (narrowing the beam) this becomes 94% in the backward pass and 76% in the forward pass, so it is worth spending some effort to decrease this one type of computation.

In the past, we attempted to reduce Gaussian computation by using Linear Descriptive Analysis then choosing the best Gaussians based on the first N dimensions of a feature vector. However, the cost on accuracy was too large and the speedup was minimal, so we abandoned this approach.

We chose to pursue a simpler variation of Padmanabhan's decision tree based FGC [3]. He builds a decision tree based on minimizing the average entropy of allophone distribution. Instead we use a simpler method using binary clustering. We start with the means of all the Gaussians in the whole system and build a decision tree using binary clustering [9]. Each leaf of this tree represents a unique region of the feature space. At each leaf we store a short list of the Gaussians from each codebook that are worth considering. The short lists are made by traversing the tree with labeled training data in a manner similar to that used in [3].

The process is as follows:

- Using a forced alignment, label the training data with codebook and Guassian ids.
- Build the decision tree using binary clustering on the means of the Gaussians.

For each frame in the training data:

- Find the leaf by walking the binary clustering tree.
- Add the Gaussian id to the short list of Gaussians for the codebook with which this data point is labeled.

Thus, this algorithm determines the likely Gaussians for a codebook to be any Gaussian that was ever used within that leaf. If any codebook within a leaf has no samples in the training data, we find the Gaussian that is closest to the mean of the leaf as the sole Gaussian for this codebook.

During decoding, for each feature vector we traverse the decision tree as we do when filling the tree. This requires only an average of 2*depth distance calculations. Then, when we need to find the most likely Gaussians for a codebook associated with a state, we only consider the Gaussians in the short list. In the forward pass, we use phonetically-tied mixtures (PTM) with 256 Gaussians per codebook. We find that we can reduce the number of Gaussians to an average of 28 per codebook. In the backward pass, we use State-Clustered Tied-Mixtures (SCTM), and we can reduce the average number from 64 to 16 Gaussians. The resulting speedup depends on the amount of computation used for Gaussians in the first place. We show the gains in speed and increase in word error rate (WER) in the two tables below. Table 1 shows that for a wide beam, the forward pass does not speed up very much, while the backward pass speeds up by almost the same factor as the reduction in Gaussian computation.

Model	FGC	Gauss/ Codebok	XRT	Speedup Factor	WER
Fw-PTM	No	256	10.1	x	42.3
Fw-PTM	Yes	28	6.6	1.5	42.4
Bw-SCTM	No	64	18.1	x	25.9
Bw-SCTM	Yes	16	5.2	3.4	26.3

Table 1. FGC vs. No FGC using a wide beam.

Table 2 shows that, with a narrow beam, the forward pass is also sped up by a significant factor. Because we spend more time

calculating Gaussians, FGC has a greater effect on speedup. We see an additional 2.6 speedup factor when we add FGC to a narrow beam search with only a loss of 0.1 in WER.

Beam	FGC	xRT	Speedup Factor	Fw Pass WER
Wide	No	10.1	x	42.3
Wide	Yes	6.6	1.5	42.4
Narrow	No	3.6	2.8	42.8
Narrow	Yes	1.4	7.2	42.9

Table 2. Effects of FGC on the forward pass.

3.2 Spreading the Grammar

During a beam search we generally keep any theory active if its score is within some factor of the largest path score at that frame. The beam search is clearly not admissible, because a theory that currently scores worse might later have a better score. But we hope that if the beam is large enough we will not often remove the best scoring global path prematurely. The basic algorithm works well if the different theories each get their scores in a gradual way in a time-synchronous manner. One confounding problem though is that the language model costs come not every frame, but rather at the transition from one word to the next. These language model transitions, which can often be below 10^{-6} for the correct word, occur at different times for each theory. Furthermore, we often exponentiate the language model probabilities by two or more in order to balance them against the acoustic probabilities. Thus a single theory can have its path score decrease in one frame by 10^{-12}, which is comparable to the width of the beam that we use in the search. To avoid search errors we must make the beam considerably larger.

We observed with the single-tree fast match that the use of incremental language model probabilities resulted in a very smooth score profile over time, which in turn allowed us to use a very narrow beamwidth. Even though our backward pass does not use a tree, we can obtain the same benefit explicitly.

We divide each ngram probability in the language model into a fixed component and a history-dependent component. The fixed component is obtained by some "average" ngram probability, *ps(w)*, which will be spread over the whole word. The variable component is obtained by dividing the usual ngram probability, *p(w|h)* by the fixed component for the word.

$$p'(w|h) = p(w|h) / ps(w)$$

This fixed component is spread over the word uniformly, with extra transition costs at each phoneme transition. Each cost, *pc(w)*, is given by

$$pc(w) = ps(w)^{1/np}$$

where np is the number of phonemes in the word. For example, the word ABOUT has four phonemes (AX-B-AW-T). If *ps(w)* for ABOUT were 0.0001, then the four phoneme transitions would each be 0.1.

We tried several weighted averages of the ngrams, but surprisingly, found that the best method for computing the fixed component is just to use the unigram probability of the word.

Table 3 shows the effect of the grammar spreading on the computation/accuracy tradeoff. From the first two rows, we see that when we use a fairly aggressive pruning (medium), the error rate increases significantly (by 3.4% absolute). The next for rows show the result with grammar spreading. If we use the same medium beamwidth, the computation naturally increases slightly, but we see that most of the lost errors are recovered. Additionally, when we reduce the beamwidth further, we see that the computation decreases further with a much smaller loss in error rate.

Comparing the results, we can see that we can either reduce computation by another factor of 2 with no additional loss, or reduce the loss considerably by spreading the grammar.

Spread Grammar	Beam width	xRT	WER
No	Wide	5.2	26.3
No	Medium	1.8	29.7
Yes	Medium	2.0	27.4
Yes	Narrow	1.6	27.9
Yes	Narrower	1.1	28.6
Yes	Even more	0.9	29.6

Table 3. Effect of spreading grammar in backward pass.

3.3 Metering

Our search algorithms discard hypotheses at several stages. Some are simply due to pruning in a beam search, while others are due to hypotheses discarded between stages in a multiple pass search. There is always the concern that we might be prematurely discarding correct answers. The typical approach to test for this is to increase the amount of computation by a large factor, increasing the beamwidth or saving more answers in early passes. We have developed a family of techniques that enable us to test for problems without increasing computation. In each case, we provide an upper bound on how well we could do if we postponed all decisions to the end.

For a particular development set, we can determine the time that each correct word should appear using a forced alignment. Then, during a recognition experiment, we can prevent the search from pruning out any hypothesis that is part of the correct word in the correct time. If running with this option reduces the word error rate, we know there might have been pruning errors.

It is possible that the forward pass, which is very approximate, could discard the correct word. However, it is not sufficient to test whether a word is missing from the forward pass. Often a

word missed from the forward pass would not have been found in the backward pass anyway. To test this, we can cause the backward pass to add the correct word ending at every frame near to the known correct ending. Again, if the error rate is decreased, we know that there was some loss due to the forward pass. Similarly for testing an upper bound on the backward pass, we can force correct words in to the lattice at the appropriate times and measure the effect.

When we tested the forward pass, we found that, even though the error rate for the forward pass is much higher than that of the backward pass, the upper bound loss due to the forward pass was only 0.1% absolute. The backward pass showed a small loss due to using the two-pass algorithm, however, the loss was small compared to other algorithms that require as little computation.

3.4 Ngram Compression

Even with 1-byte probabilities the total storage for a large language model with many millions of ngrams can be very large. Most of the storage is used to represent the words in the model. The incremental storage needed for each transition is two bytes for the destination word plus one byte for the probability. We have implemented an ngram compression strategy using a combination of Huffman coding, quanitzation, and reordering to cut the incremental storage to just 10 bits per transition: 6 bits for the word ID, and 4 bits for the probability.

The technique relies heavily on redefining and reordering the words such that the numbers are much smaller, and then using Huffman coding. The steps are:

1. Sort the words in the vocabulary according to the number of different bigrams where they are used as the destinations. The word used most is given number 1.
2. Use these new numbers as the word Ids.
3. Sort the destinations for each state according to the new word ID.
4. Delta code the sorted destination Ids and code the differences with a Huffman code.

We use the fact that the trigram destinations for history w1,w2 must be a subset of the bigram destinations for history w2.

5. First code the trigram destinations as the position within the corresponding list of bigram destinations.
6. Delta and Huffman code these numbers as with bigrams.

We quantize the ngram probabilities using 4 bits. We found that this results in an increase in WER of only 0.2% absolute.

4. SUMMARY

We have described several new and reviewed several older algorithms that help speed up recognition. We have shown that the BBN BYBLOS system which incorporates these algorithms, can reach near real-time speed with minimal loss in accuracy. Specifically the 2-Pass N-best paradigm along with Fast Gaussian Computation, grammar spreading and pruning have been combined to produce a robust near real time decoder. We have devloped a ngram compression technique that cuts ngram memory usage in half and have described a metering algorithm that directly determines a baseline quickly and provides an upper bound on WER for our decoder.

Acknowledgements

This work was supported in part by the Defense Advanced Research Projects Agency and monitored by Ft. Huachuca under contract No. DABT63-94-C-0063. The views and findings contained in this material are those of the authors and do not necessarily reflect the position or policy of the Government and no official endorsement should be inferred.

5. REFERENCES

[1] L. Nguyen, R. Schwartz, "Efficient 2-Pass N-Best Decoder", EuroSpeech '97, Rhodes, Greece, Sept. 1997, pp. 167-170.

[2] L. Nguyen, T. Anastasakos, et al., "The 1994 BBN Byblos Speech Recognition System", Proc. of ARPA SLS Technology Workshop, Austin, TX, Jan. 1995, pp. 77-81.

[3] M. Padmanabhan, E. E. Jan, L. R. Bahl, M. Picheny, "Decision-tree based feature-space quantization for fast gaussian computation", Proc. of 1997 IEEE Workshop on Automatic Speech Recognition and Understanding, Santa Barbara, CA, Dec. 1997, pp. 325-330.

[4] L. Nguyen, R. Schwartz "The BBN Single-Phonetic-Tree Fast-Match Algorithm", Proc. of ICASSP '99, Phoenix AZ, March 1999.

[5] S. Austin, R. Schwartz, P. Placeway, "The Forward-Backward Search Algorithm", Proc. of IEEE ICASSP-91, Toronto, Canada, May 1991, pp. 697-700.

[6] L. Nguyen, R. Schwartz, et al., "Search Algorithms for Software-Only Real-time Recognition", Proc. of ARPA Human Language Technology Workshop, Princeton, NJ, Mar. 1993, pp. 411-414.

[7] R. Schwartz, S. Austin, "A Comparison of Several Approximate Algorithms for Finding Multiple (N-Best) Sentence Hypotheses", Proc. of IEEE ICASSP-91, Toronto, Canada, May 1991, pp. 701-704.

[8] L. Nguyen, R. Schwartz, et al., "Is N-Best Dead", Proc. of ARPA Human Language Technology Workshop, Princeton, NJ, Mar. 1994, pp. 411-414.

[9] J. Makhoul, S. Roucos, H. Gish, "Vector Quantization in Speech Coding", Proc. of IEEE, Vol. 73, No. 11, November 1985, pp. 1551-1588.

[10] P.S. Gopalakrishnan, L.R. Bahl, "Fast Match Techniques", *Automatic Speech and Speaker Recognition, Advanced Topics*, Kluwer Academic Publishers, Ch. 17, pp. 413-428.

[11] H. Ney, R., et al, "Improvements in Beam Search for 10000-Word Continuous Speech Recognition", Proc. ICASSP '92, San Francisco, CA., Mar. 1992, pp. I.9-12.

[12] B.T. Lowerre, "The Harpy Speech Recognition System", PhD Thesis, Carnegie-Mellon Univ., 1976, Pittsburgh, PA.

[13] F. Alleva, X. Huang, M. Hwang, "An Improved Search Algorithm Using Incremental Knowledge for Continuous Speech Recognition", ICASSP-93, April 1993, pp. 307-310.

[14] R. Schwartz, L. Nguyen, "Single Tree Method for Grammar Directed, Very Large Vocabulary Speech Recognizer", US Patent 5621859, Apr. 1997.

[15] R. Schwartz, L. Nguyen, J. Makhoul, "Multiple-Pass Search Strategies", Speech and Speaker Recognition, Advanced Topics, Kluwer Academic Publishers, Ch. 18, pp. 429-456.